Approaches and Evaluation for Text Quality Improvement

by

Ankit Vadehra

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2026

© Ankit Vadehra 2026

Examining Committee

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Frank Rudzicz Professor, Faculty of Computer Science, Dalhousie University
Supervisor(s):	Pascal Poupart Professor, School of Computer Science, University of Waterloo
	Olga Vechtomova Professor, Department of Management Science and Engineering, University of Waterloo
Internal Member:	Jesse Hoey Professor, School of Computer Science, University of Waterloo
	Jimmy Lin Professor, School of Computer Science, University of Waterloo
Internal-External Member:	Mark Smucker Professor, Department of Management Science and Engineering, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

I wrote all chapters of this thesis and designed, implemented, and evaluated all the work presented under the supervision of Prof. Pascal Poupart, who provided guidance on the research, the analysis, and the structure of the thesis. Prof. Olga Vechtomova provided feedback on the writing and formatting of the thesis. Parts of Chapter 2, including the neural network architecture background, and Appendix A are modified from content I wrote for my PhD Comprehensive-II Examination. The exceptions are:

- **Chapter 3:** The Industry dataset's use case and properties, and the task were defined through discussions with the Industry Partner team. The human baseline annotations on the development set were produced by the Industry Partner linguists. Gustavo Sutter and I designed the cleaning and normalization steps for the industry dataset. Prof. Pascal Poupart, Gustavo Sutter, and I jointly created the error-type label set and performed the error tagging.
- **Chapter 5:** Bill Johnson and Gene Saunders at Scribendi Inc.[1] organized the data collection, and the post-edit corrections and time annotations were produced by professional editors at Scribendi Inc. This chapter is based on the following publication: Ankit Vadehra et al. "Time Is Effort: Estimating Human Post-Editing Time for Grammar Error Correction Tool Evaluation". In: *Proceedings of the Fourth Workshop on Bridging Human-Computer Interaction and Natural Language Processing (HCI+NLP)*. Suzhou, China: Association for Computational Linguistics, Nov. 2025, pp. 178–196. DOI: 10.18653/v1/2025.hcinlp-1.15. URL: https://aclanthology.org/2025.hcinlp-1.15/

Chapters 3 and 4 have not been published. I plan to submit them for publication after revising the work and expanding the set of experiments.

[1] https://www.scribendi.com/

Abstract

Text generation models have been utilized for various tasks, like conversational systems (chatbots), sentence translation, captioning, summarization systems, creative text generation, etc. In this thesis, I focus on Text Quality Improvement models for the task of Grammatical Error Correction (GEC) and Automatic Speech Recognition (ASR) Transcript Error Correction via Post-Editing (PE). I also investigate and introduce Post-Editing Effort Estimation in Time (PEET) quality estimate for GEC Tool evaluation. The three projects are connected through the correction edit - modelling ASR error edits to generate synthetic training data, decomposing GEC into edit-type-specific sub-tasks, and using the number and type of edits to estimate the human post-editing time.

The ASR Transcript Error Correction task specifically focuses on a low-resource industry setting, where a limited number of source ASR transcripts with corresponding target transcripts (bitext) are available for training and evaluation, alongside a large number of unpaired source ASR transcripts (unitext). This is an under-researched but extremely realistic setting (across many industries), since collecting large-scale (domain-specific) ASR training-target annotations is expensive, and customer data privacy regulations can make it difficult to use the original audio signal for any correction solution. I utilize different semi-supervised synthetic data augmentation approaches to address this low-resource setting and develop Seq2Seq and LLM instruction-based PE models. To better understand the task, human evaluation on the industry dataset was performed to categorize transcription error types. These error type labels are then used to compare the final proposed correction methods at a finer granularity, providing insights into the type and amount of errors each method succeeds at fixing, beyond an overall Word Error Rate (WER) metric score.

Building on the impact of different correction edit types observed in the ASR correction work, I propose an Edit Type Ensembling (ETE) framework for GEC. This framework breaks down the overall Text Quality Improvement task into correction sub-tasks based on the three primary edit operations: insertion, deletion, and substitution. Apart from a regular baseline correction model trained on all connected edits, three independent baseline models are trained on the individual edit types, and their predictions are combined using a simple edit-type-based ensembling approach. The driving hypothesis is that a sequence-to-sequence model trained on all edit operations captures complex syntactic edits like rephrasing and rewrites effectively, but may not optimally learn the structurally simpler, independent edit operations - insertion, deletion, and substitution (like punctuations, spelling etc.); training edit-specific models on these individual operations and ensembling their predictions can utilize the same limited training data more efficiently, resulting in stronger GEC models using the same baseline architecture.

Finally, I focus on some of the limitations of traditional GEC evaluation metrics by considering PE temporal effort, dependent on the number of correction edits and their type. PE effort has been studied across three levels: technical effort (number of edits), cognitive effort (psychological assessment required to identify and correct errors), and temporal effort (total time taken to evaluate and perform post-edits). While PE effort has been explored extensively for Machine Translation, and to a smaller extent for Text Summarization, Natural Language Generation, and GEC, I present the first work to consider PE effort in time (PEET) for quality estimation of a GEC tool. For this project, I collected the first large-scale dataset capturing time-to-correct annotations for two English GEC test sets, BEA19 and CoNLL14, post-edited from two conditions: the original sentence and the output of two strong GEC tools, GECToR and GEC-PD. I further present a new human editor-centric GEC Tool evaluation method, the PEET Scorer, to estimate time-to-correct for GEC Tool predictions, and show that it aligns with human judgment rankings that correlate strongly with the technical post-editing effort. I quantify the editing time saved and improvement in final correction quality when using GEC tools for first-pass text editing, and observe that paraphrasing and punctuation edits have the greatest impact on PE time-to-correct, while determining whether a sentence needs correction also adds considerable editing time.

Acknowledgements

This has been an incredibly long and complex endeavour of my life! It is hard to pinpoint moments I'm thankful for amid all the long periods spent lost, on my own, in some kind of Groundhog Day-like experiment. Now that I have reached the end of this journey, I want to acknowledge the people who helped, guided, and shared stories and jokes with me through this period. It meant a lot, kept me sane and connected, and helped me keep on keeping on.

First and foremost, I'd like to thank my supervisor, Prof. Pascal Poupart, whose advice, support, encouragement, and patience throughout my PhD have been invaluable. I can recall many instances when I was stuck on something, and discussing it with him helped me move forward - so I could get stuck on the next thing. I am extremely grateful for all his insight on research, industry, and life in general. I'm also grateful for the computing resources he gave me access to, which made my research possible (at all hours of the day and night). I'd also like to thank my co-supervisor, Prof. Olga Vechtomova. I didn't get a chance to work with her much during my PhD, but I am thankful for her guidance and comments on my research whenever I reached out to her.

I would like to thank the members of my Examining Committee, Prof. Frank Rudzicz, Prof. Jesse Hoey, Prof. Jimmy Lin, and Prof. Mark Smucker, for taking the time to read my thesis, giving suggestions, and making my PhD defence a pleasant experience. I especially want to thank Jimmy and Frank for helping me shape two of my thesis projects. Jimmy introduced me to the literature on post-editing effort, which helped me formalize the ideas Pascal and I came up with for the PEET Scorer (Chapter 5). My work in Chapter 4 was aided by a talk Frank gave at the Vector Institute NLP Workshop in February 2024 on what drives ASR system improvement - data or model architecture changes. I had been excited about this question for a while, and combining it with my GEC work helped me design the project. I would also like to thank Jesse for his support on a range of topics over the years: suggestions after my seminar presentations, future research directions, and help when I was figuring out what to do after my PhD.

I would like to thank Gustavo Sutter, Ruotian Wu, Prashanth Arun, Ahmad Rashid, and Yue Yan for all the chats, ideas, and discussions about various projects over the years. I'd also like to thank Bill Johnson, Mahsa Teimourikia, and Stephan Goulet for their advice, support, and help with my work and with exploring the industry landscape. I also appreciate the support from UWaterloo alumni Nalin Chhibber, Kshitij Jain, Dhruv Kumar, Hareesh Bahuleyan, Nabiha Asghar, and Catalin Avram, who were happy to help whenever I reached out, whether it was about industry job opportunities, the interview process, or any other questions I had.

Beyond the people I've worked with directly, I genuinely want to thank the amazing CS administrative staff, who have been very helpful and kind throughout my PhD. Thanks to Amy Todd, Paula Roser, Tarannom Haghighi, Greg McTavish, Izabela Rutkowski, and others for their support and random chats, and for going above and beyond at times to help me navigate the ever-changing university procedures and requirements.

I've spent most of my time in graduate school talking to machines and servers, some of them old and tired like me! They've broken down, crashed, and been unresponsive plenty of times. I'd especially like to thank Lori Paniak for being incredibly helpful in handling all my computing and server queries. I also appreciate the help from Tommy Tran in setting up a few computing resources.

I'd like to thank Robin DeMille, Brett Hasted, and Teri-Lyn Hasted for providing a nice house that I ended up calling home for most of my PhD studies. Thank you for being jovial and kind, and for always addressing any issue I had promptly, which made living there for so many years an enjoyable and comfortable experience. A big part of my PhD also happened during the COVID-19 lockdown, which was an insane and rough time! Having a safe and stable place to live and work made a huge difference, and I'm especially grateful for that.

Finally, I'd like to thank my mom and dad, who have been supportive and constantly there for me throughout this graduate school rollercoaster. Thank you for helping me stay connected to the world outside my PhD, and at times helping me escape it. Thanks for the calls, chats, and silly discussions, and for sharing and caring about all the ups and downs. I'm also grateful to the rest of my close family for all their wishes and blessings along the way (now that I'm somehow here, they must have worked!). Over these years, I also lost some family members I really cared about, while I was away doing my thing. I miss them and think about them often - thank you for everything.

Dedication

I spent a long time deciding what to write here, and who it should be for, without being too dramatic.

I couldn't settle on anything.

So, I've decided to do what "Michael" couldn't when he was "learning to rock" -

"In my search for freedom and peace of mind", I'm leaving "the memories behind".

Table of Contents

Examining Committee ii

Author's Declaration iii

Statement of Contributions iv

Abstract v

Acknowledgements vii

Dedication ix

List of Figures xvi

List of Tables xx

1 Introduction 1

1.1 Low-Resource ASR Transcript Correction 2

1.1.1 Contributions 3

1.2 Edit Type Ensembling for Grammatical Error Correction 3

1.2.1 Contributions 4

1.3 Temporal Post-Editing Effort for GEC Evaluation 4

1.3.1 Contributions 4

1.4 Organization 5

2 Background Information **6**

2.1 Machine Learning Paradigms 6

2.1.1 Supervised Learning 6

2.1.2 Unsupervised Learning 7

2.1.3 Self-Supervised Learning 7

2.1.4 Semi-Supervised Learning 8

2.1.5 Reinforcement Learning 8

2.2 Text Generation System 9

2.2.1 Non-Neural Based Techniques 9

2.2.2 Neural Text Generation Techniques 10

2.2.3 Neural Pre-trained Language Models (PLMs) 18

2.2.4 Large Language Models (LLMs) 22

2.2.5 LLM Training Strategies 24

2.2.6 LLM Inference and Prompting Approaches 27

2.3 Automatic Speech Recognition Systems 28

2.3.1 From Statistical Models to End-to-End Systems 29

2.3.2 Multilingual and Low-Resource ASR 30

2.3.3 ASR Transcription Error Types 31

2.4 ASR Transcription Correction 31

2.4.1 Non-Intrusive Transcript Refinement 32

2.4.2 ASR Correction Model Architectures 33

2.4.3 The ASR Over-Correction Issue 35

2.4.4 Synthetic Data Generation for Correction 36

2.5 Grammatical Error Correction Models 37

2.5.1 From Rule-Based to Neural GEC Models 39

2.5.2 Approaches to Grammatical Error Correction Models 40

2.5.3 Edit-Type Specialized GEC Models 42

2.5.4 Ensembling and System Combination . . . 43
2.5.5 The Over-Correction Issue in GEC . . . 45
2.5.6 Dataset . . . 45
2.5.7 Evaluation Metrics and Approaches . . . 46

3 Low-Resource ASR Transcript Correction 51
3.1 Introduction . . . 51
3.2 Motivation and Contributions . . . 54
3.2.1 Contributions . . . 54
3.2.2 Chapter Structure . . . 55
3.3 Dataset Description . . . 55
3.3.1 Industry Dataset . . . 55
3.3.2 Public Datasets . . . 61
3.3.3 Edit Type Distribution . . . 64
3.4 Human Evaluation and Error Analysis . . . 66
3.4.1 Error Categorization . . . 66
3.4.2 Human Annotation Baseline . . . 67
3.5 Evaluation Metrics . . . 69
3.5.1 Word-Level Alignment via KaldiAlign . . . 69
3.5.2 Metric Definitions . . . 70
3.6 Synthetic Data Augmentation for ASR Transcript Correction . . . 71
3.6.1 N-Gram Confusion Matrix Construction . . . 71
3.6.2 Probabilistic Sentence Partitioning . . . 73
3.6.3 Confusion Sampling and Synthetic Pair Generation . . . 74
3.6.4 Application to the ASR Datasets . . . 75
3.6.5 Synthetic Data Generation Configuration . . . 77
3.7 ASR Correction Methods . . . 79
3.7.1 Sequence-to-Sequence Correction Model . . . 79

3.7.2 LLM-Based Few-Shot Correction . . . 80
3.7.3 LLM-Based N-Best Correction with Synthetic Data . . . 82
3.7.4 Post-Processing . . . 82
3.8 Experiments . . . 83
3.8.1 Experimental Setup . . . 83
3.8.2 Synthetic Data Configuration Selection . . . 86
3.8.3 Results . . . 89
3.8.4 Error Category Analysis . . . 94
3.9 Conclusions and Future Directions . . . 96

4 Edit Type Ensemble Models **98**
4.1 Introduction . . . 98
4.2 Motivation and Contributions . . . 100
4.2.1 Contributions . . . 101
4.2.2 Chapter Structure . . . 102
4.3 Methodology . . . 102
4.3.1 Individual Edit-Based Models for GEC . . . 102
4.3.2 Ensembling via Edit-based System Combination (ESC) . . . 105
4.3.3 LLM-Based Edit-Specific Ensembling . . . 107
4.4 Experiments . . . 113
4.4.1 Experimental Setup . . . 113
4.4.2 Results . . . 115
4.5 Parameter-Efficient Edit-Type Models . . . 119
4.5.1 Method . . . 119
4.5.2 Results . . . 119
4.5.3 Inference Cost . . . 120
4.6 Discussion . . . 122
4.7 Summary and Future Work . . . 123

5 Temporal Post-Editing Effort for GEC Evaluation **124**
5.1 Introduction . . . 124
5.2 Motivation and Contributions . . . 125
5.2.1 Contributions . . . 126
5.2.2 Chapter Structure . . . 126
5.3 Background Work . . . 127
5.3.1 Post Editing Effort in Machine Translation . . . 127
5.3.2 Post Editing Effort in Grammatical Error Correction . . . 128
5.4 Dataset Collection and Processing . . . 129
5.4.1 Dataset Source . . . 129
5.4.2 Editor Correction Framework . . . 130
5.4.3 Data Filtering . . . 131
5.4.4 Correction Quality . . . 132
5.4.5 Impact of GEC Tools . . . 136
5.5 Methodology . . . 137
5.5.1 Training Feature Sources . . . 137
5.5.2 ERRANT Edit Feature Extraction . . . 139
5.5.3 PEET Scorer Models . . . 139
5.6 Experiment Results . . . 140
5.6.1 Performance of the PEET Scorer . . . 140
5.6.2 Impact of Error Types on Post-Edit Effort . . . 142
5.6.3 PEET Scorer for GEC Quality Estimation . . . 142
5.7 Limitations . . . 144
5.8 Conclusion and Future Work . . . 144

6 Conclusion and Future Work **146**
6.1 Low-Resource ASR Transcript Correction . . . 146
6.2 Edit Type Ensembling for Grammatical Error Correction . . . 147
6.3 Post-Editing Effort in Time for GEC Evaluation . . . 148
6.4 Summary . . . 149

References **150**

APPENDICES **193**

A Non-Neural Based Text Generation Techniques **194**

B Multiple-Run Evaluation and Statistical Significance **199**

C Supplementary Material for Low-Resource ASR Transcript Correction **200**

C.1 Edit Type Distribution of the ASR Datasets 200

C.2 ASR Transcript Correction LLM Results 200

C.2.1 Per-split results on RED-ACE 204

D Supplementary Material for Edit Type Ensemble Models **207**

D.1 LLM Model Specifications and Implementation Details 207

D.1.1 Implementation Details 208

D.2 LLM Ensembling Task Instruction Prompts 209

D.3 Baseline LLM GEC Prompting Results 213

D.4 Selected Edit-Type Thresholds 214

D.5 Adapter Models Configuration 214

E Supplementary Material for Temporal Post-Editing Effort Estimation **217**

E.1 MAE and Pearson Correlation for PEET Evaluation 217

E.2 Neural Regression Models for PEET Estimation 218

E.3 Predictive Model Parameters 219

E.4 PEET Scorer using SRC, MO and TRG Sentence Features 220

E.5 Feature Impact on Post-Editing Time using Regression Coefficients 221

E.6 PEET Scorer Ranking and Comparison of GEC Tools with Human Judgment Rankings 222

List of Figures

2.1 An un-rolled Recurrent Neural Network (RNN) 11

2.2 An RNN-based Sequence-to-Sequence (Seq2Seq) architecture 13

2.3 Global Attention Model 14

2.4 Transformer Architecture 16

2.5 ERRANT edit operations and types 50

3.1 The ASR transcript correction task. Given a noisy source transcription (from an ASR model), the objective is to generate a corrected version (what was actually spoken). Corrections can be decomposed into substitution, deletion, and insertion edits. 51

3.2 Comparing the task data used by various ASR correction methods against those available in this project. The low-resource, text-only setting of the task means that the audio recordings and other conditional data signals are unavailable. The unannotated ASR transcript is the only resource available in some quantity, which is used for different synthetic data augmentation methods. 52

3.3 The synthetic data augmentation approach proposed in this chapter (Section 3.6). The small bitext and larger ASR source transcripts (unitext) is used to generate synthetic noisy source-target pairs using a confusion model. The correction model is pre-trained on the synthetic pairs before fine-tuning on the bitext. Depending on the type of bitext edit inserted, the unitext can be used as the source (add correction edits) or target (add corruption edits) for the synthetic pair generation. 53

3.4 Example of Alignment-based sentence extraction from a pseudo call source-target transcription. The grey rows mark EQUAL alignment spans that were selected as the left and right boundary spans (labelled ①, ②, ③). All words between two boundaries are extracted as a single source-target sentence pair. For this example, sentence word length was sampled from [12, 17] by the algorithm. The right boundary span of each extracted pair becomes the left boundary of the next, ensuring complete and non-overlapping coverage of all the edits in the transcript. The error words corresponding to each edit type are underlined. 58

3.5 The breakdown of the low-resource training data WER into substitution, deletion, and insertion edits for each dataset. The edits are counted in the regular ASR direction: deletions are reference words the ASR model ignored, insertions are extra words it added. Each edit segment in the bar is that edit type's contribution to the WER, the full bar height denoting that dataset's WER. The industry ASR dataset is deletion-dominant, while the public ASR dataset is primarily substitution edits. The public dataset corresponds to the sampled low-resource bitext (Table 3.4) used in this work. 65

3.6 Overview of the Synthetic Data Augmented (SDA) Seq2Seq ASR transcription correction approach. The pre-trained Seq2Seq model is trained in two stages - pre-training on the synthetic data (Stage 1), then fine-tuning on the supervised bitext (Stage 2). The Seq2Seq Baseline model is only fine-tuned on the bitext. 79

3.7 Overview of the LLM prompting-based ASR transcription correction method. The k in-context examples are selected from the bitext, based on BM25 similarity. For $k = 0$ (zero-shot), the prompt only contains the prompt instruction and the ASR-generated transcription. 80

3.8 Overview of the LLM-based ASR transcription correction using synthetic n-best candidates. The SDA approach is used to generate four candidate corrections for the ASR source transcript at different thresholds θ. The four candidates and the ASR transcript serve as the n-best list for the LLM. . . 83

3.9 Word Error Rate (WER) between the Synthetic Data Augmentation (SDA) output for the starting sentence at different partitioning thresholds θ. A lower WER indicates that the generated text is closer to the desired reference. The plot lines indicate the two directions of SDA - corruption (bitext-target + edits -vs- bitext-source) and correction (bitext-source + edits -vs- bitext-target). The curve values are an average over six runs. 87

4.1 The Grammatical Error Correction (GEC) task. Given a source sentence, the system produces a correction (improved version of the source). Each correction can be represented as atomic edits - substitution, insertion, and deletion - which are used by the Edit-Type Ensemble framework. 98

4.2 Summary of the complete project. Four edit-specific GEC models - M_R (substitution), M_M (insertion), M_U (deletion), and M_{All} (all edits) generate the correction predictions, which are combined by one of three ensembling approaches to produce the improved correction. 99

4.3 Overview of the four edit-specific baseline GEC models (M_i) which generate the correction predictions. The final correction is proposed by analyzing, evaluating and ensembling the four individual predictions. 103

4.4 Overview of the Edit-based Ensemble approach using the ESC framework. Multiple edit-type specific models (M_i) generate correction predictions, which are decomposed into ERRANT-based edits. These edits are evaluated using the ESC regression model. The chosen edits are applied to the source text to get the final correction prediction. 105

4.5 Overview of the LLM-based GEC Ensemble approach. The edit-type model (M_i) predictions are used to design an LLM instruction prompt. The LLM generates the final correction conditioned on the prompt. 109

4.6 Overview of the LLM-edit quality classification GEC Ensemble approach. All candidate edits from all models (M_i) predictions are extracted and evaluated using LLMs on their quality and importance. 111

5.1 The source sentences and the first-pass outputs of two GEC tools are post-edited by professional editors while logging the time-to-correct. The resulting dataset is used to quantify the impact of GEC tools on text editing and to train the PEET Scorer. 125

5.2 Data Collection Setup and Framework for this experiment. 130

5.3 Comparison between the GEC-Tool assisted Text Editing Framework and the Post-Editing Framework. 131

5.4 Survey instructions for the editor to perform post editing, and obtain target corrections for my dataset. 132

5.5 Example source sentence and its first-pass edit from the Survey. The editor can make further improvements in the text box. Submitting the final target correction. 132

5.6 Different PEET Models and their corresponding training features. 137

5.7 The three sentence iterations for each source sentence and the ERRANT sentence edits (A, B and C) extracted between each sentence pairing. . . . 138

5.8 Predicted vs. actual time-to-correct for the Linear Regression PEET Scorer of a single run (seed 200), using the Length baseline and the 4, 25 and 55 edit features. The dashed line represents $y = x$. For better visualization, the axes range is limited to the 99th percentile of the time values (160.5 seconds). The r and MAE values are computed on all test sentences. 141

List of Tables

2.1 *Source*, *Target* and example *Model Output* made by a GEC Tool. 48

3.1 The statistics for the industry dataset after normalization, alignment and sentence extraction from the call transcripts. The Development (Dev) set sentences are extracted from the training data before sentence extraction. The Mean Word is calculated for the source sentences in the bitext. WER is calculated against the manually annotated reference transcripts. 59

3.2 Filler Word and Acoustic Tag Vocabulary ($\mathcal{V}_{\text{FW}}$) 61

3.3 The complete corpus-level statistics and Word Error Rates (WER) for the public ASR datasets, after normalization, for all data splits. 64

3.4 Corpus-level statistics for the sampled Low-Resource (LR) training data (bitext) and the remaining unannotated source text used for synthetic data augmentation (unitext). $4,500$ examples were sampled for the bitext to replicate the low-resource setting of the industry ASR datasets. 65

3.5 The different categories of edits between the ASR transcription source and the annotated target correction for the *Industry English dataset (200 sentences)* obtained during the human evaluation stage. The table presents the error category, examples, and relative percentage distribution. 67

3.6 The different categories of edits between the ASR transcription source and the annotated target correction for the *Industry French dataset (100 sentences)* obtained during the human evaluation stage. The table presents the error category, examples, and relative percentage distribution. 68

3.7 Human annotation correction results on the development sets for English and French. The manual annotation WER is computed against the reference transcripts and reflects the difficulty of the task. 68

3.8 Example Levenshtein alignment between an ASR hypothesis and its reference transcript. Each aligned fragment pair increments the corresponding entry in the confusion matrix C. 72

3.9 Word-level alignment for a liaison error. The ASR system incorrectly transcribes *army issue* as *i miss you*. The word-level alignment maps `army`→`i` and `issue`→`miss`, which is not a meaningful edit transformation on its own. Only the multiple-word fragment captures the error correctly, and it can only be sampled if both target words are assigned to the same partition. . 78

3.10 Naming convention for the correction models reported in the results. The other evaluated LLMs (LLaMA, Mistral and DeepSeek) follow the same naming in Appendix C.2. 85

3.11 Development set WER (%) and WER Corrected (%) for correction models pre-trained on synthetic data generated at ρ word error rate and in both generation directions, with $\theta = 0.05$. All values are obtained before fine-tuning on the supervised bitext using BART-Large and mBARThez. 88

3.12 Impact of removing all insertion edits from the correction model output as a post-processing step. The Baseline model is fine-tuned using only the supervised bitext; the SDA (synthetic data augmentation) model is pre-trained on synthetic data generated with the final configuration (corruption,$\rho = 1\times$, $\theta = 0.05$). Results are reported on the development set using BART-Large (English) and mBARThez (French) for all datasets. 89

3.13 Correction results on the industry English dataset. ↓: statistically significant WER reduction over ASR Normalized ($p < 0.05$). 90

3.14 Correction results on the industry French dataset. ↓: statistically significant WER reduction over ASR Normalized ($p < 0.05$). 91

3.15 Correction results on the public RED-ACE (English) dataset [79], pooled over the *clean* and *other* splits. Results for the other evaluated LLMs are in Appendix C.2. ↓: statistically significant WER reduction over ASR Normalized ($p < 0.05$). Qwen1M-K3 and Qwen1M-K4500 are single runs (Section 3.8.1). 92

3.16 Correction results on the public MLS (French) dataset [225]. Results for the other evaluated LLMs are in Appendix C.2. ↓: statistically significant WER reduction over ASR Normalized ($p < 0.05$). Qwen1M-K3 and Qwen1M-K4500 are single runs (Section 3.8.1). 93

3.17 Percentage of errors corrected for each category on the industry English development set. n is the number of instances per category (Table 3.5). Categories with no corrections by any approach (Hallucination, Anonymization and Contraction) are removed. 94

3.18 Percentage of errors corrected for each category on the industry French development set. n is the number of instances per category (Table 3.6). Categories with no corrections by any approach (Missing long sequences, Contraction, Repeated words, Hallucination, English words, Tag error and Questionable reference) are removed. 95

4.1 Performance of the BART-Base and BART-Large GEC models under different training strategies on the BEA19 (ERRANT) and CoNLL14 (M^2) test sets. Each value is averaged over six runs. ↑: statistically significant $F_{0.5}$ improvement over the Curriculum Learning (All) baseline ($p < 0.05$). 116

4.2 Performance of the BART-based GEC models on the CoNLL14 test set evaluated using ERRANT, reported in addition to the M^2 scores (Table 4.1) for the edit-type (R, M, U) analysis. Each value is averaged over six runs. ↑: statistically significant $F_{0.5}$ improvement over the Curriculum Learning (All) baseline ($p < 0.05$). 117

4.3 Edit-type (R, M, U) Precision, Recall and $F_{0.5}$ on the BEA19 and CoNLL14 test sets, evaluated using ERRANT. Each value is averaged over six runs. ↑: statistically significant $F_{0.5}$ improvement over the Curriculum Learning (All) baseline ($p < 0.05$). 118

4.4 Results of the full fine-tuned and adapter-based edit-specific models after the two-stage training, on the BEA19 (ERRANT) and CoNLL14 (M^2) test sets. The values are the average of three runs. The higher $F_{0.5}$ of each pair is presented in bold. 120

4.5 Storage and inference time for generating all four corrections (All, R, M, U) on a single NVIDIA Tesla P100 GPU with beam size 5. *Per sentence* is the time (batch size 1) for all four streaming corrections; *Full set* is the total time to correct all FCE-Dev 2,191 sentences. The factor value in parentheses is the size saving/time speed-up compared to the four fully fine-tuned models. 121

5.1 The M2 precision and recall quality score for all Bryant and Ng [28] target correction sets for the official CONLL14 competition task. 133

5.2 Quality Scores of the two GEC Tools output predictions, target Editor Corrections (EC) and the State-of-the-Art GEC Tool - GRECO [232] on the official BEA19 and CONLL14 competition. 134

5.3 Quality Scores of the two GEC Tools output predictions and their final target Editor Corrections (EC) using the BEA19 and CONLL14 - Source Sentence EC as target reference. 134

5.4 Inter Annotator Agreement (IAA) scores for the different A, B, C annotation sets using the ERRANT $F_{0.5}$ metric. RAND(n) represents a random selection of "n" items from the respective set. 136

5.5 The average time to correct (**in seconds**) for a sentence and word; correcting the source and after first-pass GEC Tool editing. 137

5.6 Average PEET estimation performance for the Statistical Models over 50 runs (different train-test data seed). The results are presented as the Pearson Correlation (r), Mean Absolute Error (MAE) loss. The 4, 25 and 55 edit feature models also include the Length and Edited Words features. 141

5.7 The standardized regression coefficients of the LR model trained on the medium (25) edit features to measure the impact of each feature on PEET estimation. 142

5.8 The correlation of the PEET model ranking with human-judgment rankings (HJR). I also provide the correlation of the HJR with the Word Error Rate (WER) metric. Spearman (ρ) and Pearson (r) correlation scores are used for comparison. A high negative correlation indicates lower time-to-correct and WER score corresponding to a higher human judgment ranking. 143

C.1 The edit type breakdown for all four ASR datasets. The Train (LR) split for the public ASR datasets refers to the sampled low-resource bitext (Table 3.4). 201

C.2 Edit type breakdown for the No Insertion (NI) training data, where all ASR deletion edits have been removed from the source–target pairs. The residual deletion rate is nonzero due to random WER alignment edit labelling. . . . 201

C.3 Correction results of all four evaluated LLMs on the public RED-ACE (English) dataset, pooled over the *clean* and *other* splits. The model naming convention is described in Table 3.10. 202

C.4 Correction results of all four evaluated LLMs on the public MLS (French) dataset. The model naming convention is described in Table 3.10. 203

C.5 Correction results of all evaluated models on the *clean* split of the public RED-ACE (English) dataset. Columns are as in Tables 3.15 and C.3, which report the same models pooled over both splits. 205

C.6 Correction results of all evaluated models on the *other* split of the public RED-ACE (English) dataset. Columns are as in Tables 3.15 and C.3, which report the same models pooled over both splits. 206

D.1 GEC Ensemble LLM Models and Specifications 207

D.2 Performance of LLAMA, QWEN, DEEPSEEK and MISTRAL under different prompting strategies on BEA19 Dev (B19D), BEA19 Test (B19T) and CoNLL14 Test (C14T). B19D and B19T are evaluated with ERRANT; C14T is evaluated with the M2 scorer. 213

D.3 Edit-type thresholds selected on BEA-19 Dev for the edit quality classification ensemble (τ_R: replacement, τ_M: insertion, τ_U: deletion). An edit is kept if its label score is $\geq \tau$; $\tau = 0.0$ keeps all edits of that type. Dev $F_{0.5}$ is the exact edit-match score used for selection, not the ERRANT score. The obtained thresholds are used to generate predictions for B19T and C14T. 214

D.4 Best configuration of each adapter method for the BART-Base All model after Stage 1 training. The FCE development set was used for checkpoint selection. Full fine-tuning is averaged over three runs (FCE Dev $F_{0.5}$ range 51.65-52.43). 215

E.1 MAE and Pearson correlation (r) for different prediction patterns on a toy example with target times $T = [20, 25, 30, 35, 40]$ seconds. The correlation for the constant mean predictor is undefined, since its predictions have zero variance. 217

E.2 The training data format for the BERT and RoBERTa LM. The example considers a sentence pair - ⟨mo⟩ and ⟨trg⟩, where "mo" is the Model Output correction made by a GEC Tool and "trg" is the post-edited target correction for "mo". ⟨mo-parse⟩ and ⟨trg-parse⟩ denote the constituency parses, and ⟨trg-pos⟩ the part-of-speech tags of the target. The special tokens [MO], [TRG] and [TO] denote sentence breaks in the input. #EW denotes the number of edited words between mo and trg. 218

E.3 Performance of Neural PEET models using different sequence model features over 5 runs. The results are shown as Pearson Correlation (r) and Mean Absolute Error (MAE) loss. 219

E.4 Neural PEET model performance over 5 runs using the source (SRC), GEC Tool Model Output (MO) and Target Correction (TRG) sentence features. The results are shown as Pearson Correlation (r) and Mean Absolute Error (MAE) loss. 220

E.5 PEET Statistical Model performance over 50 runs (different train-test data seed) using Incorrect and Ignored separated Edit features (Section 5.5.1) extracted from SRC, MO and TRG sentence triples. The results are presented as the Pearson Correlation (r), Mean Absolute Error (MAE) loss. 220

E.6 The standardized regression coefficients of the LR model trained on all the big (55) edit features to measure the impact of each feature on PEET estimation. 221

E.7 The standardized regression coefficients of the LR model trained on the small (4) edit features to measure the impact of each feature on PEET estimation. 221

E.8 The correlation of the features used to train the small-edits(4) Linear Regression (LR) model in Table 5.6. I also list the standardized regression coefficients to measure the impact of each feature on PEET estimation. . . 222

E.9 PEET Scorer estimated average time-to-correct per sentence and ranking for 7 GEC Tool corrections on the FCE dataset (1936 Sentences), along with their Human Judgment Ranking (HJR), presented in *Napoles-FCE* [197] (Section 5.6.3). The 7 GEC Tools consist of Seq2Seq Neural Models. 222

E.10 PEET Scorer estimated average time-to-correct per sentence and ranking for 7 GEC Tool corrections on the WikiEd dataset (1984 Sentences), along with their Human Judgment Ranking (HJR), presented in *Napoles-Wiki* [197] (Section 5.6.3). The 7 GEC Tools consist of Seq2Seq Neural Models. 223

E.11 PEET Scorer estimated average time-to-correct per sentence and ranking for 12 GEC Tool corrections on the CONLL14 dataset (1312 Sentences), along with their Human Judgment Ranking (HJR), presented in *Grundkiewicz-C14(EW)* [90] (Section 5.6.3). The 12 GEC Tools consist primarily of rule-based and statistical machine translation architecture. 224

E.12 PEET Scorer estimated average time-to-correct per sentence and ranking for 15 GEC Tool corrections on the CONLL14 dataset (1312 Sentences), along with their Human Judgment Ranking (HJR), presented in *SEEDA-C14-All(TS)* [136] (Section 5.6.3). The 15 GEC Tools consist of strong SOA Neural Models. 225

Chapter 1

Introduction

Text generation systems have been utilized for a variety of tasks, like conversational systems (chatbots), sentence translation, captioning, summarization, and creative text generation [11, 162]. However, the output of these systems can contain various issues. Apart from factual and task-specific inconsistencies, they can also contain sentence discourse-level issues like grammatical errors, awkward phrasing, text hallucinations, and stylistic inconsistencies, among others [112]. This has led to a growing need for Text Quality Improvement models that can take imperfect generated (or human-produced) text as input and improve it, either automatically or in collaboration with a human editor through Post-Editing (PE).

In this thesis, I focus on Text Quality Improvement in two specific tasks: Grammatical Error Correction (GEC) and Automatic Speech Recognition (ASR) Transcript Error Correction. Both tasks share a common underlying goal - given an imperfect source text, produce an improved target text - but differ in their error distributions, available training data, and evaluation approach. Beyond improving the correction models themselves, I also examine how the quality of these tools should be evaluated, particularly from the perspective of the human editor who must post-edit their output. With this in mind, I introduce the Post-Editing Effort Estimation in Time (PEET) scorer, a human-centric evaluation approach for GEC tools.

Considering these motivations, this thesis explores the following directions:

- correcting ASR transcription errors in a low-resource, industry setting using synthetic data augmentation-based Post-Editing models,
- training edit-type-specific GEC models (insertion, deletion, and substitution edits)

and ensembling their predictions to build models stronger than regular GEC baselines, and

- evaluating GEC tools using a temporal post-editing effort metric that estimates the human editing time better than the number of edited words (technical effort), which edit-based metrics such as WER rely on.

These three directions are connected through two common themes. First, the correction edit and its type are used across all three projects - to generate synthetic ASR training data (Chapter 3), to decompose GEC into edit-type-specific sub-tasks (Chapter 4), and to estimate the human post-editing time of GEC tool output (Chapter 5). Second, each project focuses on a low-resource setting, in terms of either the available training data or the computation resources. Together, the three projects follow the Text Quality Improvement pipeline - building correction models with limited supervised data, utilizing the same data more efficiently, and evaluating the model output by the human post-editing effort it requires.

I describe each of these directions and the contributions of my thesis in the following sections.

1.1 Low-Resource ASR Transcript Correction

Automatic Speech Recognition (ASR) systems are widely used to transcribe spoken conversations across various applications and domains [239, 2], but their output frequently contains errors that must be corrected before the transcripts can be efficiently used downstream. Developing Post-Editing models for ASR correction is challenging in an industry setting, where only a limited number of source ASR transcripts with corresponding target transcripts (bitext) are available for training and evaluation, without the original audio data. This low-resource, text-only setting is prevalent across many industries, since collecting large-scale domain-specific ASR training-target annotations is expensive, and customer data privacy regulations can make it difficult to use the original audio signal for developing corrective models. Despite being a realistic and common industry setting, it has received limited attention in the ASR transcript correction literature. Most existing low-resource research instead focuses on building ASR solutions for new languages [254], rather than addressing low-resource constraints within a specific industry or application environment.

In Chapter 3, I address ASR transcript correction in this low-resource setting using a real-world industry dataset of anonymized customer support call transcripts.

1.1.1 Contributions

1. I propose various semi-supervised synthetic data augmentation approaches to develop Seq2Seq and LLM instruction-based ASR Post-Editing correction models, and evaluate these models on both an industry dataset and public ASR datasets for English and French.

2. I conduct a human evaluation on the industry dataset to categorize different transcription error types, and use these labels to evaluate the different correction methods at a finer granularity than the overall Word Error Rate (WER) score.

1.2 Edit Type Ensembling for Grammatical Error Correction

Various approaches have been proposed for Grammatical Error Correction (GEC) models, including sequence-to-sequence (Seq2Seq) architectures, edit-based tagging models, and more recently, LLM-based correction methods [29, 204]. In this thesis project, I focus on a straightforward and popular Seq2Seq GEC Baseline model [128], which uses the BART Neural Architecture [158], and is trained in an end-to-end manner to generate an improved prediction for an erroneous source input. Corrections made by a GEC model correspond to three main edit types - insertion, deletion, and substitution. Since the source text can contain many issues of different kinds, the Seq2Seq Baseline model training tries to learn the correction patterns for all edit types together. This allows it to perform complicated corrections like fluency rewrites, rephrasing, and word reordering. However, this joint-training approach may cause the model to give less priority to the structurally simpler, independent edit operations.

In Chapter 3, as part of the analysis of ASR transcript correction models, I observed that training models with and without specific edit types (like including or excluding insertion edits) impacts correction performance. This observation, that Seq2Seq models can be trained to selectively perform certain kinds of edits, along with the goal of utilizing the same limited training data more efficiently, motivated the Edit Type Ensembling (ETE) framework for GEC proposed in Chapter 4.

1.2.1 Contributions

1. I perform multi-stage curriculum training to get baseline GEC models for each individual edit type (insertion, deletion, and substitution), in addition to the regular baseline model trained on all joint edits.

2. I utilize the predictions of all four models as inputs to a regression-based GEC ensembling method [231] to obtain the final corrected output.

3. I show that this approach utilizes the same limited training data better, resulting in a stronger and more precise GEC model than the original published BART baseline [128], using the same baseline architecture.

1.3 Temporal Post-Editing Effort for GEC Evaluation

Text editing can involve multiple rounds of revision, and utilizing a strong GEC tool to perform corrections in the initial round can significantly impact further human editing effort and the final text quality. Since GEC tool-assisted editing is an iterative process, an editor makes post-edits to the tool output to obtain the desired final-correction, and the post-editing (PE) effort required can be used as a quality estimate for that tool. PE effort has been studied across three levels [142]: technical effort (number of edits), cognitive effort (psychological assessment required to identify and correct errors), and temporal effort (total time taken to evaluate and perform post-edits). While PE effort has been explored extensively for Machine Translation, and to a smaller extent for Text Summarization, Natural Language Generation, and GEC, temporal PE effort specifically has not been researched for the GEC tools and task.

In Chapter 5, I transition from improving GEC models (Chapter 4) to evaluating GEC tools. This thesis project focuses on measuring their performance from a human editor's perspective, by proposing the Post-Editing Effort in Time (PEET) Scorer as a human-centric metric for GEC tool usability.

1.3.1 Contributions

1. I collect the first large-scale dataset with time-to-correct annotations for two English GEC test sets, BEA19 and CoNLL14, post-edited from two conditions: the original sentence and the output of two strong GEC tools, GECToR and GEC-PD.

2. I quantify the editing time saved and improvement in final correction quality when using GEC tools for first-pass text editing, and identify which edit types have the greatest impact on PE time-to-correct.

3. I propose the PEET Scorer, a new human-centric GEC tool evaluation method that estimates time-to-correct for GEC tool predictions, and show that its GEC tool rankings align with human judgment rankings that correlate strongly with the technical post-editing effort.

1.4 Organization

This thesis is organized in the following chapters:

- In Chapter 2, I present the information and theory for all methods, architectures, and related work for the tasks explored in this thesis, including the machine learning concepts and background on text generation systems, ASR systems, and GEC models.
- In Chapter 3, I focus on ASR transcript correction in a low-resource industry setting, utilizing semi-supervised synthetic data augmentation approaches to develop Seq2Seq and LLM instruction-based Post-Editing models. I also conduct a human evaluation to categorize the transcription error types.
- In Chapter 4, I propose an Edit Type Ensembling framework that decomposes GEC into the three edit-specific sub-tasks (insertion, deletion and substitution), and show that it utilizes the limited training data better, resulting in a stronger and more precise GEC model than the baseline.
- In Chapter 5, I introduce Post-Editing Effort in Time (PEET) and the PEET Scorer, a human-centric evaluation method for GEC tools, along with a new large-scale time-to-correct annotations dataset.
- In Chapter 6, I summarize my conclusions and discuss directions for future work.

Chapter 2

Background Information

In this chapter, I present some of the underlying methods, architecture, models and related work for the different tasks explored by me in this thesis. An overview of these techniques will support and help understand the work I present and describe in the subsequent chapters.

2.1 Machine Learning Paradigms

Machine Learning (ML) is a subset of Artificial Intelligence (AI) focused on creating algorithms, models and functions that can identify and learn patterns from data and make task-specific predictions. ML models consist of classical statistical techniques, which work well for structured data (when the training features are discrete and known), as well as modern Neural Network (NN) based Deep Learning (DL) techniques that can learn from vast amounts of training data and high-dimensional complexity found in Computer Vision and Natural Language Processing (NLP) tasks.

2.1.1 Supervised Learning

Supervised learning utilizes labelled training data to learn a functional mapping $f : X \rightarrow Y$ to estimate an output/prediction Y for an input X. During the learning phase (training), a loss function is defined to measure how well the model estimation aligns with the true label value. The goal of the training process is to optimize the model to minimize the loss value.

Various classification and regression models can be trained using labelled data.

Regression Estimates a prediction value in a continuous numerical range (e.g., housing price, temporal values, etc.).

Classification Choosing the output from a fixed and discrete set of categories/labels (e.g., Sentiment Prediction, Spam Detection, etc.)

Models Supervised learning techniques can be applied to both classical statistical and modern deep learning models.
The Classical statistical models include Linear Regression (regression), Logistic Regression and Naive Bayes (classification), Decision Trees and Random Forests (regression and classification), and Support Vector Machine (SVM), which finds an optimal separating hyperplane between classes (classification and regression). These models are well-established techniques. A comprehensive review of their theory and applications can be found in most Machine Learning introductory literature [96, 19]. Deep Learning models such as Multi-layer Perceptrons (MLP), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformers can be modified for both regression and classification tasks by specifying the training objective (loss) and the output-layer architecture. I describe these models in greater detail in Section 2.2.2.

2.1.2 Unsupervised Learning

Unsupervised learning techniques estimate groupings and intrinsic patterns from unlabelled data (X) based on different features. Clustering methods like Hierarchical Clustering, K-Means and Dimensionality Reduction approaches like Principal Component Analysis (PCA) can be used to organize data into similar groupings without the need for discrete labels. Unsupervised learning methods focus on similarity and distance metric between data items rather than a “loss" function/value while learning.

2.1.3 Self-Supervised Learning

Self-Supervised Learning (SSL) is a specific type of Unsupervised Learning approach that makes use of unlabelled data to train a model, by having the model generate its own labels. SSL can be useful in scenarios where obtaining labelled data is challenging, costly or not possible, but a large amount of unlabelled data is available [94]. SSL is often used during the pretraining phase to learn the underlying structured representation of data for subsequent "downstream task" learning. Some examples of SSL tasks in Natural Language Processing

are - Sentence Masked Word Prediction [61, 158], Next Sentence Prediction [61], Next Word Prediction for Text Generation [235], etc. The SSL pre-training tasks are categorized as pretext tasks.

2.1.4 Semi-Supervised Learning

Semi-Supervised Learning (Semi-SL) makes use of supervised and unsupervised learning approaches by jointly using labelled and unlabelled data to train a model [321]. Similar to SSL, Semi-SL methods utilize a large amount of unlabelled data for model learning. However, unlike SSL, which is completely unsupervised, the Semi-SL framework makes use of some labelled data to augment the unlabelled data with synthetic/pseudo labels for model training. This can help the model learn downstream task-specific information during the pre-training stage. Techniques combining an SSL pretext task with a supervised downstream task can be considered as Semi-SL as well.

2.1.5 Reinforcement Learning

RL focuses on goal-oriented learning through interaction with an environment. An agent learns to make a sequence of decisions by maximizing the cumulative reward. The framework is defined by four components:

State (S) The current situation of the agent.

Action (A) The set of possible moves.

Reward Function ($R : S \times A \to \mathbb{R}$) A mapping from state-action pairs to real numbers, interpreted as the immediate feedback received (reward) for taking an action in a given state.

Transition Distribution ($P : S \times A \to \Delta(S)$) A mapping from state-action pairs to a probability distribution over next states denoting the likelihood of moving to a new state given an action.

The objective is to derive an optimal policy ($\pi^* : S \to A$), defined as the policy that maximizes the expected cumulative reward:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid \pi\right], \tag{2.1}$$

where $\gamma \in [0, 1)$ is a discount factor that controls the importance of future rewards, and s_t, a_t denote the state and action at time step t.

This approach has led to the development of complex strategic game (e.g., Go, Dota 2, Chess, etc) solvers [265, 264, 15] and the alignment of Large Language Models with human instructions and preferences (e.g., InstructGPT) [208, 35, 307].

2.2 Text Generation System

Text Generation or Natural Language Generation (NLG) systems have been explored and researched for a long time. The goal of an NLG system is to produce understandable text in English or other languages. NLG systems can be designed to generate text conditional on different types of inputs and tasks, like conversational systems (chatbots), translating sentences, captioning, summarization systems, data-to-text summarization, and generating creative text like stories, jokes, etc. [32, 78]. The NLG task can be broken into two categories: data-to-text and text-to-text generation.

The text-to-text generation task takes some text as input and generates a text output in response. Examples include - Machine Translation systems [113], Summarization Systems [45], Text Spelling and Correction systems [144], and text style transfer [119, 72]. The data-to-text task utilizes structured data like Tables or Knowledge Graphs as input and generates text conditional on that. For example, weather and financial report systems [81, 242, 220], hospital patient report [141], and news article headline generation [157].

Historically, most of the NLG systems utilized rule-based generation techniques. Upon the introduction of readily available corpora and supporting computing power, statistical techniques and Neural Network models for NLG were introduced.

2.2.1 Non-Neural Based Techniques

In my work, I utilize and design Deep Learning based Neural Network (NN) models to accomplish different tasks. However, before delving into the NN background, I want to acknowledge the vast literature on Non-NN based techniques. These approaches rely on careful model design, feature extraction and handcrafted rules for text processing and generation.

I provide a comprehensive review of the Non-NN Based Techniques in Appendix A.

2.2.2 Neural Text Generation Techniques

The work done by Bengio et al. [13] compares the performance of NN and Statistical language models (LM). The improved performance obtained using NN-LM was pivotal in ushering in the popularity of NN based generative models. The recent trend in utilizing NN architecture to tackle a variety of NLP tasks is inspired by the ease of availability of high-performance computing resources (GPU/CPU) and massive amounts of task specific training data. As a result, most of the newer NN inspired models have achieved State-of-the-Art (SOTA) results in almost all NLP tasks.

NN inspired architectures also eliminated the need to experiment with different feature selection and extraction techniques which are often time-consuming and expensive. Based on the granularity of the selected feature - word, character, byte-pair, etc. the model can learn a low dimensional representation of that feature in continuous vector space. Utilizing several techniques, NN models can learn to generalize over training data much better and as a result obtain better results.

An interesting fact to consider is that representing the discrete features in a continuous vector space allows the possibility for the NN model to learn a wide variety of features that might not be possible to encompass in a rule based or predefined feature-set statistical model. As a result, reducing the feature complexity while increasing the model complexity and utilizing low dimensional representational space (thanks to high capacity GPUs) the model can generalize and learn a much better and richer low-level-task-invariant representation like Language Models [61, 215] and Feature Embeddings [189, 143].

However, task-specific models for lower-level applications often benefit from an inductive bias about the task as I explain in the subsequent sections. As a simple example, Language Models utilize the bias that word ordering matters in a text phrase. As a result, unigram models don't perform as well as context dependent models.

There has been a lot of work in improving NN architectures for text generation like feed-forward neural language models that utilize $n-1$ previous words as context similar to statistical LMs. But most of the recent work and the SOTA models are dependent on NN architectures that utilize variable-length context history. The NN models that utilize this approach are the Recurrent Neural Network (RNN) and Transformer architectures. The RNN model transforms, updates and propagates the input context at each time step into a lower-dimensional representation space called history. As a result, a history obtained at any given time-step can be considered as a complete representation of the text phrase till that time step. Transformer models generate a representation for each feature in parallel and as a result are much more time-efficient than RNNs. To incorporate the sequential

ordering information of text they append a positional and importance weighting of the low dimensional representation of each feature. I expand and explain the newer NN architectures in this section as these models have become the baseline and SOA for present research in the NLP applications I focus on in my work.

Recurrent Neural Networks (RNNs)

RNNs are a class of neural network models which can handle variable length data. Each RNN unit has a self loop and an internal state which is helpful in processing sequential and time-series data like audio waveform, stock value, text and video. The RNN self-loop is responsible for the recurrence relation and the internal state acts as the memory representation of all information seen till that time-step. RNNs are helpful in processing sequential data because they can utilize each individual data point as well as the relation of that data point with the preceding data points. This results in generating a more comprehensive and rich textual representation. As a result of their usefulness, RNNs are used in NLP for various tasks like text classification, language generation, named entity recognition, etc. [330].

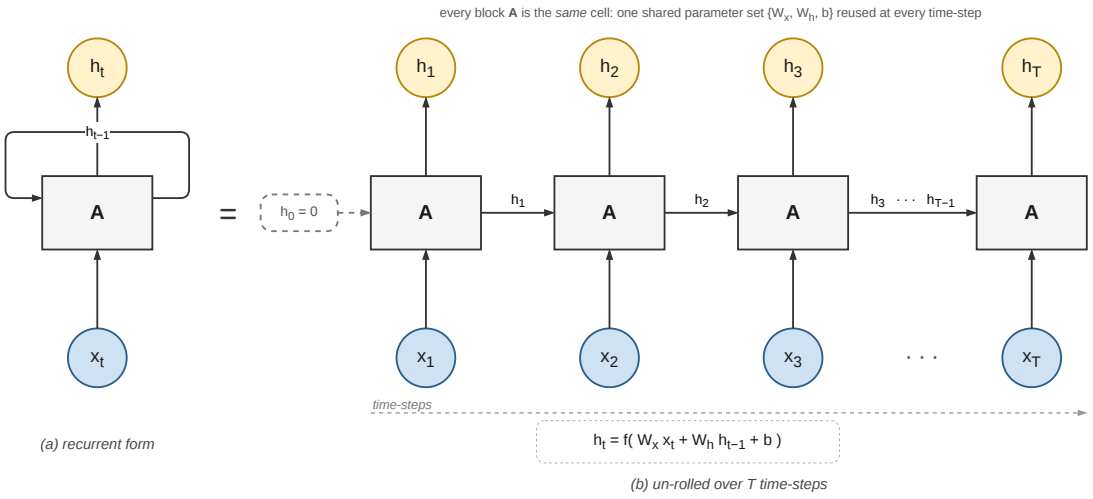

Figure 2.1: An un-rolled Recurrent Neural Network (RNN). The recurrent form (a) and its un-rolled equivalent (b) describe the same network with the same parameters.

Figure 2.1 shows a simple RNN in both its recurrent and un-rolled forms. x_t is the the embedding representation of the model input (word, character, byte-pair) at time-step t. For every step the cell A combines the input representation with the hidden state from the

previous step, h_{t-1}, to generate the new hidden state $h_t = f(W_x x_t + W_h h_{t-1} + b)$. The h_t vector is used for both the output at step t and as the state passed forward to step $t+1$. In RNN models h_t is as a fixed-size, and necessarily lossy, context summary of all the inputs up to timestep t.

There are many different and complex RNN models proposed in the literature. Even though the operation and state transfer approach is same; they differ in their internal state architecture. The most popular RNN models are the Long Short Term Memory (LSTM) [104] and Gated Recurrent Unit (GRU) [39] models. The models consist of internal states and gates combined with non-linearity to allow better generalization and training on complex function mappings.

Although RNN models are effective at learning sequential representations, certain text generation systems require the output to be conditioned on an input sequence. For instance, translating a source sentence (Machine Translation [77]) or generating a response to a user query (Question-Answer[18]). To address this, Sutskever, Vinyals, and Le [277] proposed combining two RNN networks into an encoder-decoder framework, referred to as the Sequence-to-Sequence (Seq2Seq) architecture, which is described in the next section.

RNN-based Sequence-to-Sequence Models and Attention

The Seq2Seq architecture consists of an Encoder and Decoder component. The encoder is responsible for producing the sentence representation for the input/conditional text, which is utilized by the decoder to generate the output text. The whole architecture learns the mapping from the input to the output in an end-to-end approach.

The Seq2Seq architecture was designed to solve the Neural Machine Translation (NMT) task [277]. Given a source and target language pair, the encoder block/sub-module generates the low-dimensional representation for the source text and the decoder block utilizes the representation to generate the target sentence. There has been a lot of work that has utilized different NN architectures for the Encoder/Decoder block and various hybrids of them as well. I focus on the Transformer architectures for the Seq2Seq task as they are the most common standard in the recent literature. I use this section to explain the Seq2Seq Model using RNNs for the Encoder and Decoder, and the next section to expand on Transformer-based Seq2Seq architectures.

Figure 2.2 shows an example of how a Seq2Seq architecture works for a conversational system with input $X = (x_1, x_2, x_3, x_4) = (How, are, you, ?)$ and the desired target $Y = (y_1, y_2, y_3) = (I, am, good)$.

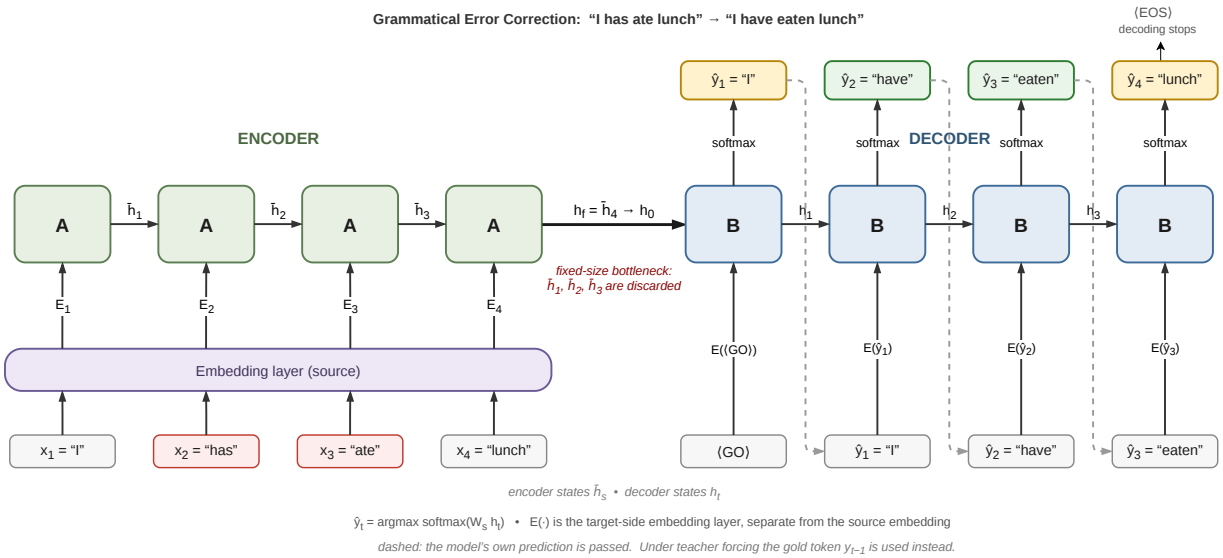

Figure 2.2: An RNN-based Sequence-to-Sequence (Seq2Seq) architecture, shown on the GEC task example. The encoder represents the source as h_f, which initializes the decoder.

The Encoder block consists of an embedding layer that can be pre-trained and initialized or trained end-to-end with the model. The embedding wights can also be frozen during training. At each time-step s, the source token x_s is passed through the embedding layer which produces the low-dimensional continuous representation E_s. This acts as the input to the Encoder-RNN at that step. The RNN combines E_s with the hidden state from the previous step, $\bar{h}_{s-1}$, to get $\bar{h}_s$. The state after the final step ($\bar{h}_f = \bar{h}_4$ in Figure 2.2) represents the whole source input context and is used to initialize the decoder, $h_0 = \bar{h}_f$. The decoder generates the prediction token one at a time. At step t it embeds the previous target token, updates its state to h_t, and outputs $\hat{y}_t = \arg\max \text{softmax}(W_o h_t + b_o)$ over the target vocabulary. The decoding process is initialized with a special start ⟨GO⟩ token and terminates when the ⟨EOS⟩ token is generated.

Figure 2.2 shows the case where the token predicted at step t is used as the input for the $t+1$ step. In practice, the ground-truth token y_t is often provided as the input instead, an approach known as teacher forcing, which helps stabilize training by reducing the downstream impact of an incorrect token prediction. However, only utilizing the ground-truth token can introduce an unwanted exposure bias - mismatch between training and inference conditions. Scheduled Sampling [12] helps reduce this issue by sampling between the ground-truth token and the model's prediction as the next input.

Another issue to observe in Figure 2.2 is that the entire sentence representation is compressed into a single low-dimensional final hidden state h_f. This causes a significant loss of information at the Encoder side and also leads to long-range dependency issues in the decoder block. The context information obtained from the source tokens is diluted in h_f, which the decoder must rely on throughout generation. To overcome this issue, the Attention module was proposed between the Encoder and Decoder [175, 6]. The attention layer allows all the Encoder hidden states ((h_1, h_2, h_3, h_4) in Figure 2.2) to be considered during decoding and also learns an importance weighting for each hidden state during each decoding time-step.

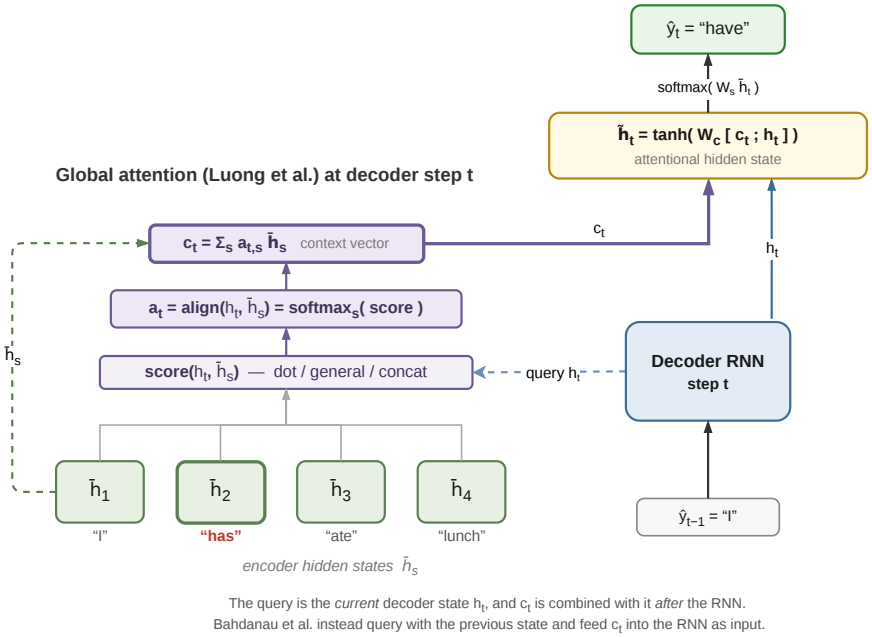

Figure 2.3: Global attention at decoder step t, following Luong, Pham, and Manning [175].

There are differences between the attention approaches presented by Bahdanau, Cho, and Bengio [6] and Luong, Pham, and Manning [175]. I choose the Global Attention approach suggested by Luong et al. for explanation since it derives ideas from both attention approaches. Figure 2.3 shows the Global Attention Model proposed by Luong et al.

The global attention model computes the importance of each hidden state generated by the Encoder for the decoder generated output at any time step t. Using Figure 2.3 as reference for notation, h_t represents the hidden representation generated by the decoder at

time-step t and $\bar{h}_s$ represents the set of hidden representations generated by the Encoder for all input time steps. I calculate the scoring for this decoder output h_t with all encoder outputs $\bar{h}_s$ using one of the following functions:

$$\text{score}\left(\boldsymbol{h}_t, \overline{\boldsymbol{h}}_s\right) = \begin{cases} \boldsymbol{h}_t^\top \overline{\boldsymbol{h}}_s & \text{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W_a} \overline{\boldsymbol{h}}_s & \text{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_a}\left[\boldsymbol{h}_t; \overline{\boldsymbol{h}}_s\right]\right) & \text{concat} \end{cases} \tag{2.2}$$

This provides an importance weighting over the encoder outputs for any given h_t. Since the scoring is arbitrary, I convert the scores into a probability distribution using the softmax function:

$$\text{align}\left(\boldsymbol{h}_t, \overline{\boldsymbol{h}}_s\right) = \frac{\exp\left(\text{score}\left(\boldsymbol{h}_t, \overline{\boldsymbol{h}}_s\right)\right)}{\sum_{s'} \exp\left(\text{score}\left(\boldsymbol{h}_t, \overline{\boldsymbol{h}}_{s'}\right)\right)} \tag{2.3}$$

Finally, the importance weighting score is multiplied with the encoder outputs to generate a weighted context vector:

$$\text{context } = \text{align}(\text{ score })^T \overline{h_s} \tag{2.4}$$

This context vector is appended to the h_t to generate a modified decoder representation:

$$h_{t'} = [h_t;\ \text{context }] \tag{2.5}$$

This combined vector is utilized to predict the decoder output at time-step t.

Experimental results indicate that attention is beneficial and makes the Seq2Seq model perform much better. The RNN-based Seq2Seq architecture suffers from the complexity of processing very long inputs. The Transformer architecture [296], described next, replaced the Seq2Seq recurrence with a new, self-attention mechanism, retaining the encoder-decoder Seq2Seq framework.

Transformer-based Sequence-to-Sequence Architecture

The Transformer architecture proposed by Vaswani et al. [296] is one of the most important papers in the Neural Network inspired NLP architectures domain. Even with attention and other improvements like bi-directionality the RNN model is incredibly slow with the

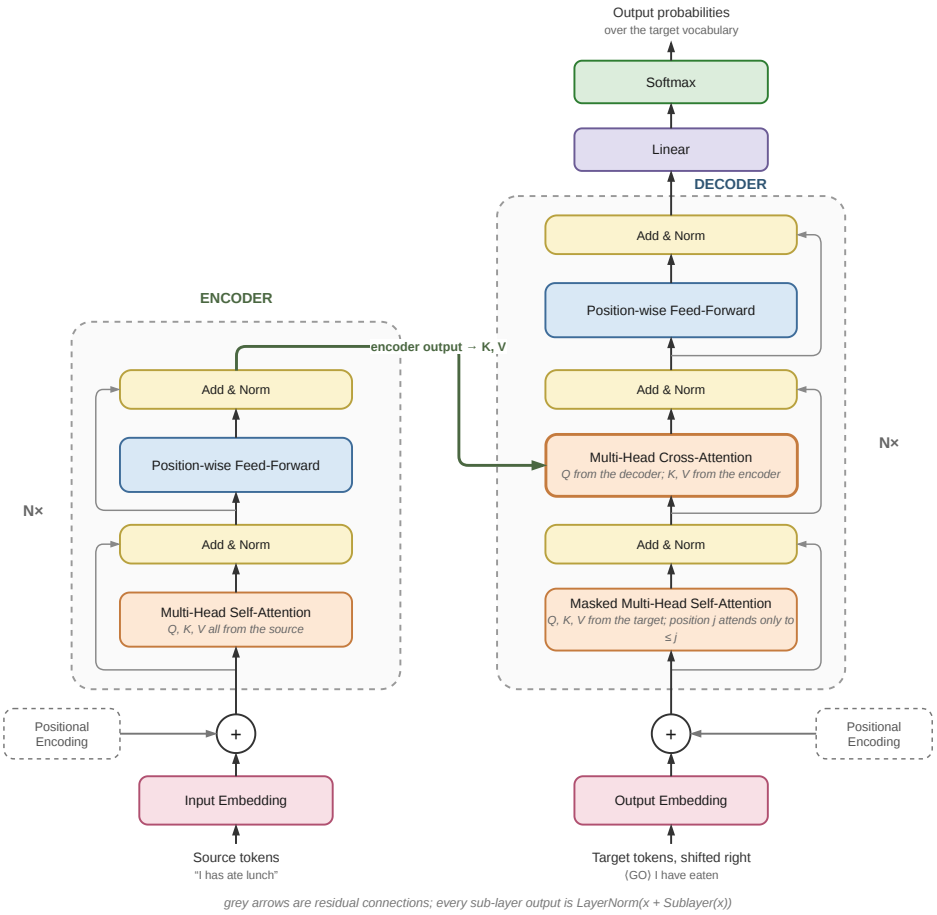

Figure 2.4: The Transformer encoder–decoder architecture [296].

recurrent un-rolling of the network. The model is further impacted by long length text input. The Transformer architecture bypasses this issue by removing all recursive calls and utilizes the attention and linear layers to produce a text representation. The original model presented the Transformer Encoder-Decoder architecture (Seq2Seq) for the task of Neural Machine Translation. However, the transformer encoder and decoder has achieved great success separately. Transformer architectures like BERT [61], RoBERTa [172] are based on just transformer-encoders and trained on a massive amount of text data. These models have had huge success in learning rich higher level representations of text for Language Understanding tasks and have provided a lot of information for different low level NLP tasks. Transformer Decoder models like GPT2 [235] are auto-regressive in nature and as a result can be used for tasks requiring text generation.

The core of the Transformer architecture is the Attention and Positional Embedding layer that is used to incorporate the intra-dependence and sequential information of text, respectively. Utilizing these components, the model can be trained to generate a sentence representation for multiple input features in parallel, providing a significant increase in training speed.

Figure 2.4 shows the Transformer architecture proposed by Vaswani et al. The number of layers for the Encoder and Decoder can be varied. Each layer consists of two components. A multi-head self-attention layer and a position-wise fully connected feed-forward layer. Residual connections and layer normalization are performed around each of the layer components. In addition to these two components, the Decoder has another multi-head attention layer to attend to the encoder outputs. The decoder block modifies the self-attention layer by adding masking capability to ignore subsequent tokens. This ensures that the generation of any token only observes the previous tokens.

The self-attention layer creates Query (Q), Key (K) and Value (V) vectors of dimension d ($d = d_q = d_k = d_v = d_{model}/A$) for each input vector, in each attention head (A = number of attention heads). Then the self-attention is calculated using the three vectors by scoring each word in the sequence with the rest. As the key/query dimension d_k gets larger, the dot products in QK^T grow in magnitude. Dividing by $\sqrt{d_k}$ keeps the attention logits stable so the softmax does not become too sharp, which can cause vanishingly small gradients.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.6}$$

The self-attention layer determines how much weight to be applied to the other parts of the sequence when encoding a specific word. The obtained vector is then passed to the feed forward neural network layer. The transformer linearly projects queries, keys and values into different vectors with the multi-head attention component.

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}\ (\ \text{head}_{\ 1}, \ldots,\ \text{head}_{\ h})\, W^O \\ \text{where head}_{\ i} &= \text{Attention}\ \left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{2.7}$$

W_Q, W_K, W_V and W_O are projection matrices that are learnt from the multi-head architecture computed by parallel heads simultaneously. By doing more projections the model can focus on different parts of the input sequence which is expressed via different vector spaces.

Finally, as mentioned previously, the Transformer architecture avoids any recurrence and stores the text sequence order information in a Positional Embedding. This position information is added to the input vector, which provides the updated input vectors.

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos/10000^{2i/d_{\text{model}}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(pos/10000^{2i/d_{\text{model}}}\right) \end{aligned} \tag{2.8}$$

Sine and Cosine function is used to estimate the relative position based on the other tokens. *pos* denotes the token position and i denotes the embedding dimension position to be encoded. d_{model} is the embedding dimension size. The value of i belongs to the interval $[0, d_{model}/2]$.

The self-attention layer has been shown to be quite effective in calculating global information dependency in text sequences. The Encoder self-attention block learns a mapping from the same Query and Value, while the Decoder Attention can be used to learn a mapping between the input token Query to the target Values.

2.2.3 Neural Pre-trained Language Models (PLMs)

The success of the Transformer neural network architecture and its ability to parallel process data faster (utilizing the self-attention mechanism) allowed for large-scale training on massive unlabelled corpora using different self-supervised objectives (Section 2.1.3). These models, called *pre-trained language models* (PLMs), learn rich contextual representations of language from text, and can be *fine-tuned* on many downstream tasks with relatively smaller labelled datasets. The pre-train $\rightarrow$ fine-tune training setup has led to State-of-the-Art performance improvements on many NLP tasks [229]. There have been many different text-based PLMs proposed [190, 162]. In this section, I focus on the PLMs I use for my projects.

BERT and RoBERTa

The **BERT** (Bidirectional Encoder Representations from Transformers) [61] model makes use of bidirectional pre-training for language understanding and text representation. BERT's self-attention mechanism allows each input token to attend to the full context (in both left and right directions), making it particularly powerful for language understanding tasks.

The BERT model utilizes two self-supervised objectives on two large datasets (BooksCorpus - 800M words [348] and the English Wikipedia - 2,500M words) for pre-training.

Masked Language Modelling (MLM) A subset of the input tokens (15% for BERT) is randomly masked (substituted with a special `[MASK]` token), and the model is trained to predict the masked tokens from bidirectional context. For a masked token at position i, the MLM objective minimizes the cross-entropy loss:

$$\mathcal{L}_{\text{MLM}} = -\sum_{i \in \mathcal{M}} \log P(w_i \mid \mathbf{w}_{\setminus \mathcal{M}}; \theta) \tag{2.9}$$

where $\mathcal{M}$ is the set of masked token positions, w_i is the original token at position i, $\mathbf{w}_{\setminus \mathcal{M}}$ is the bidirectional context tokens that are unmasked, and θ are the BERT model parameters.

Next Sentence Prediction (NSP) Sentence pairs (A, B) are sampled from the corpus, where 50% of the time B is the actual sentence following A (labelled `IsNext`) and 50% of the time B is a randomly sampled sentence from the corpus (labelled `NotNext`). The model is trained to predict the correct label for each pair using a binary classification objective:

$$\mathcal{L}_{\text{NSP}} = -\left[y \log P(\text{IsNext} \mid A, B; \theta) + (1 - y) \log P(\text{NotNext} \mid A, B; \theta)\right] \tag{2.10}$$

where $y \in \{0, 1\}$ is the label that indicates if the sentences are a sequential pair from the corpus. The BERT pre-training objective is the combination of the two objectives:

$$\mathcal{L}_{\text{BERT}} = \mathcal{L}_{\text{MLM}} + \mathcal{L}_{\text{NSP}} \tag{2.11}$$

The pre-trained BERT model can be further fine-tuned on different downstream NLP task datasets by attaching task-specific layers. BERT has demonstrated strong performance on various NLP tasks, including question answering, named entity recognition, natural language comprehension, and text classification [1, 246, 76].

RoBERTa (Robustly Optimized BERT Pre-training Approach) [172] explored different hyperparameters and training objective optimization for BERT pre-training and identified several modifications that improved downstream performance:

- Removed the NSP training objective as it didn't improve downstream task performance.
- Performed training on longer input sequences and in larger batches.
- Used dynamic masking, where the input masking is performed for each batch.

- Pre-trained on a significantly larger dataset (160GB of text compared to BERT's 16GB dataset) - CommonCrawl News and Stories, OpenWebText, and BookCorpus.

These modifications indicated that the BERT model was under-trained, and choosing better hyperparameter values resulted in RoBERTa consistently outperforming the BERT model on multiple benchmarks, making it a stronger baseline for encoder-based tasks. In this thesis, I use the BERT and RoBERTa encoders for training different neural PEET Scorer metric models in Chapter 5.

BART

BART (Bidirectional and Auto-Regressive Transformers) [158] is a Transformer-based encoder-decoder model (Seq2Seq - Section 2.2.2) that uses the bidirectional encoding from BERT and the autoregressive decoding approach of GPT [235]. The autoregressive decoder can generate output sequences conditioned on the encoder representation of the input sequence and the previously generated decoder outputs. This makes the BART model incredibly suitable for Seq2Seq conditional text generation tasks.

The BART model is pre-trained using a denoising autoencoder objective (Self-Supervised). The input text is corrupted using one or more pre-specified noise functions, and the training objective is to predict the original uncorrupted text. The pre-training is performed using the same 160GB corpora (CommonCrawl News and Stories, OpenWebText, and, BookCorpus) used by the RoBERTa model [172]. The BART model uses five different noise functions to corrupt the input text:

Token Masking Similar to the BERT MLM objective, 15% of the input tokens are randomly replaced with a special `[MASK]` token.

Token Deletion Random tokens are deleted from the input sequence.

Text Infilling A block of consecutive input tokens (a text span) is replaced with a single `[MASK]` token. Span lengths are drawn from a Poisson distribution ($\lambda = 3$).

Sentence Permutation A document is split into sentences. The sentences are shuffled and used as input to the BART model.

Document Rotation A random token is selected as the beginning of the document and the document is rotated so it begins at that token.

For each of these noisy transformations, the BART model is trained to reconstruct the original text from the corrupted input. This makes the BART model robust to noise and effective at understanding input inconsistencies and generating coherent and improved output sequences. For each corrupted input sequence $\hat{\mathbf{x}}$ and the original uncorrupted sequence $\mathbf{x}$, the BART pre-training minimizes the Seq2Seq cross-entropy reconstruction loss:

$$\mathcal{L}_{\text{BART}} = -\sum_{t=1}^{T} \log P(x_t \mid x_{<t}, \hat{\mathbf{x}}; \theta) \tag{2.12}$$

where x_t is the original token at decoding step t, $x_{<t}$ are all the previously generated tokens, $\hat{\mathbf{x}}$ is the noisy input sequence to the encoder, and θ are the BART model parameters. Comparing the results of the BART model pre-trained with different noise functions, Lewis et al. [158] use a combination of Token Masking, Text Infilling and Sentence Shuffling for the final pre-training.

The denoising pre-training objective naturally aligns BART with tasks that require mapping a noisy or imperfect input to a clean output sequence. As a result, BART and its variants have been widely adopted as strong baselines for Seq2Seq text generation and evaluation tasks [338, 332]. In this thesis, I focus on two text improvement tasks: Grammatical Error Correction (GEC) and automatic speech recognition (ASR) transcription error correction. The BART model has shown strong baseline performance for both GEC [128, 275] and ASR transcription error correction [64] tasks, making it a suitable model for my projects. As a result, I use BART and its variants for both tasks in this thesis.

Model Variants and Size The BART model is available in two sizes - **BART-Base** (6 encoder and 6 decoder Transformer layers, 140M parameters) and **BART-Large** (12 encoder and 12 decoder Transformer layers, 400M parameters).

Multilingual BART (mBART) To extend the BART model to multilingual settings, Liu et al. [171] proposed **mBART** (Multilingual BART), a sequence-to-sequence denoising autoencoder pre-trained on large-scale monolingual corpora across 25 languages. mBART follows the same denoising training approach as BART. The pre-training corpus is the CC25 subset of the CommonCrawl dataset [311], covering many diverse languages. The multilingual pre-training allows mBART to be fine-tuned for both monolingual and cross-lingual Seq2Seq tasks. Unlike the BART model, mBART is available in a single size - **mBART-Large** (12 encoder and 12 decoder layers, 610M parameters).

BARThez and mBARThez While mBART supports input from many languages, models pre-trained on a single language often achieve better performance for that language compared to multilingual models, due to better language-specific training data (example, mBERT vs CamemBERT for French [186]). **BARThez** [124] is a French BART model pre-trained from scratch on a large French text corpus utilizing the same denoising autoencoder objective as BART. BARThez is available in a single model size comparable to BART-Base (6 encoder and 6 decoder layers, 165M parameters). Since BARThez is pre-trained entirely on French text, it learns better linguistic representations than mBART for French.

mBARThez [124] extends BARThez by initializing the model from the mBART-Large checkpoint and further pre-training it on the same French corpus used for BARThez. This allows mBARThez to leverage the multilingual representations learnt during mBART pre-training and improve on the French language through further pre-training. Kamal Eddine, Tixier, and Vazirgiannis [124] show that mBARThez outperformed both BARThez and mBART on French Seq2Seq generation tasks.

In this thesis, I make use of BART-based models for different tasks and languages. I utilize the BART-Base and BART-Large as the base generation model for English language tasks - Grammatical Error Correction in Chapter 4 and Automatic Speech Recognition (ASR) Transcription Correction in Chapter 3. In Chapter 3, I also work on French ASR Transcript Correction using the BARThez and mBARThez models.

2.2.4 Large Language Models (LLMs)

The significant scaling of computational power, Transformer PLM parameters and training data has introduced a new class of models referred to as *Large Language Models* (LLMs) [346]. LLMs are autoregressive Transformer-decoder models trained on massive web-scale corpora using the next-token prediction objective (similar to the GPT model [235]). The next-token prediction objective (Self Supervised - Section 2.1.3) trains the model to predict the next token x_t given all the preceding tokens $x_{<t}$, minimizing the cross-entropy loss over the full sequence:

$$\mathcal{L}_{\text{LLM}} = -\sum_{t=1}^{T} \log P(x_t \mid x_{<t}; \theta) \tag{2.13}$$

where T is the sequence length and θ are the model parameters. Training the LLM on massive and diverse web datasets allows the model to learn a wide range of linguistic, factual and reasoning capabilities across many different domains and writing styles.

At sufficient scale, LLMs are able to display *emergent capabilities*, ie., abilities that are not present in smaller models but appear as the model size increases [309]. These include complex multi-step reasoning, in-context learning, instruction following and NLP task generalization with and without task-specific training [16, 342, 219]. Different LLMs can handle different input and output modalities, like text, images, audio and video [209], as well as multimodal models that combine several of these modalities [313]. In this thesis, I focus primarily on text-based LLMs for text generation and correction tasks. I describe audio and speech-based LLM in Section 2.3.1 in the context of an Automatic Speech Recognition (ASR) system background.

Open-Source and Closed-Source LLMs

LLMs can be categorized as *closed-source* (proprietary) and *open-source* models. Closed-source LLMs are proprietary models accessible only through APIs and product websites, with model weights and full training details not publicly available. Some popular closed-source LLMs include the GPT models [25, 205, 206], Claude [1] and Gemini [80] [80]. Open-source LLMs make their model weights publicly available, allowing the community to use, fine-tune and adapt them for specific tasks without the computational cost of pre-training from scratch. Well-known open-source LLMs include the LLaMA models [188, 290, 291, 173], Mistral [117, 118] and the more recent DeepSeek [56, 58, 59, 60] and Qwen [7] model families. A comprehensive survey of LLM architectures, training approaches and capabilities is presented by Zhao et al. [343, 343].

Pre-training and Alignment

LLMs are pre-trained similarly to PLMs - self-supervised training on large unlabelled corpora to learn general language representation. However, unlike BERT and BART, which are trained on datasets comprising hundreds of GBs of text, LLMs are trained on trillions of tokens sourced from diverse web crawls, books, code repositories and other sources [343, 290]. Training LLMs on this massive amount of data requires a significant computational infrastructure and various training optimizations.

A major difference between LLMs and earlier PLMs is the subsequent *alignment* step performed after pre-training. Despite capturing vast amounts of factual knowledge [34], pre-trained LLMs do not necessarily follow instructions or generate outputs that align with human preferences. To address this, a further fine-tuning stage is performed - *Instruction*

[1] http://claude.ai/

Tuning and *Preference Optimization*, which I describe in Section 2.2.5. Many modern LLMs such as LLaMA and Qwen [290, 7] release both the pre-trained and instruction-tuned weights, allowing researchers to utilize the pre-trained weights for downstream tasks and to further fine-tune the instruction-tuned weights for specific applications.

LLMs for Text Generation Tasks

LLMs have shown strong performance across various text generation tasks like open-ended dialogue systems, summarization, machine translation, code generation, question answering, and text correction [262, 319, 11]. Their ability to generate fluent, coherent and context-specific text across many domains has made them a natural baseline for most NLP tasks, and benchmarking against LLMs has become a standard evaluation practice in recent NLP research [154, 263]. The two primary approaches used to apply LLMs to specific tasks are:

Prompting The LLM (instruction-tuned variant) is provided with a natural language instruction or a set of examples describing the task (from the user), and the model generates the task output during inference without any model weight updates. This leverages the in-context learning capabilities of LLMs and is described in detail in Section 2.2.6.

Fine-tuning The pre-trained LLM weights are further updated on a task/domain specific labelled dataset. Full fine-tuning updates all model parameters (which is compute-expensive for large models), while parameter-efficient fine-tuning (PEFT) methods like LoRA [110] and Prefix Tuning [165] update only a small subset of parameters, making task fine-tuning easier.

In this thesis, I explore the use of LLMs for Automatic Speech Recognition (ASR) Transcription Correction and Grammatical Error Correction (GEC) in Chapter 3 and 4, utilizing prompting-based approaches to evaluate their effectiveness for my project.

2.2.5 LLM Training Strategies

The pre-trained LLM described in Section 2.2.4 does not necessarily follow task instructions or generate outputs that align with human preferences and expectations. Several training approaches have been proposed to align and improve LLMs for specific tasks and prompt instruction-following behaviour. In this section, I describe the training strategies most relevant to my thesis work - Instruction Tuning (and Preference Optimization), and Curriculum Learning.

Instruction Tuning

Instruction Tuning [308] is a supervised fine-tuning method that trains a pre-trained LLM on various NLP tasks described as natural language instructions. Each training example consists of a textual instruction for the task (e.g., *"Correct the grammatical errors in the following sentence:"*, *"Summarize the following paragraph:"*), any other optional task-specific input and the desired output response. The model is fine-tuned to minimize the standard cross-entropy loss over the output response tokens:

$$\mathcal{L}_{\text{IT}} = -\sum_{t=1}^{T} \log P(y_t \mid y_{<t}, \mathbf{c}; \theta) \tag{2.14}$$

where y_t is the target output token at step t, $y_{<t}$ are the previously generated prediction tokens, $\mathbf{c}$ is the instruction and input context, and θ are the model parameters.

By fine-tuning on a large and diverse collection of instruction-template task examples, the LLM learns to follow general instructions and is able to solve other tasks framed as natural language instructions at inference time. Wei et al. [308] show that instruction tuning on ∼60 tasks significantly improves the performance of the LLM on other unseen tasks (zero-shot). As a result, instruction tuning has become a standard post-pretraining step for almost all modern LLMs to improve task generalization and instruction-following capability. A comprehensive survey of instruction tuning approaches, datasets, and evaluation is presented by Zhang et al. [342].

Preference Optimization (PO) methods are often applied after the instruction tuning stage to align the LLM outputs with human preferences better. While instruction tuning trains the model to generate task responses in a question-answer template format, it does not optimize the model to prefer higher-quality outputs over lower-quality ones. PO tries to solve this by training the LLM on ranked human preference judgments between pairs of model outputs (y_{win}, y_{loss}) for the same input x, where y_{win} is the preferred response. The most widely used PO approach is **Reinforcement Learning from Human Feedback** (RLHF) [44, 208] where a reward model trained on human preference pairs is used to fine-tune the LLM via reinforcement learning. **Direct Preference Optimization** (DPO) [237] is a simpler PO alternative that eliminates the need for a separate reward model and optimizes the LLM on preference pairs using a binary cross-entropy loss. PO approaches significantly improve the quality and instruction-following capability of modern LLMs. A comprehensive survey of different preference optimization methods is presented by Wang et al. [307].

In this thesis, I use various instruction-tuned LLMs for Automatic Speech Recognition

(ASR) Transcription Correction and Grammatical Error Correction (GEC) in Chapter 3 and 4.

Curriculum Learning

Curriculum Learning (CL) [14] is a PLM/LLM training strategy in which the training examples are provided to a model in a specific order. The model is shown the easier examples first and harder examples are provided as the training proceeds. The main intuition for this approach is that learning easier examples initially allows better initialization, improved convergence and generalization for harder examples [269].

In the context of NLP tasks, the difficulty of a training example can be defined in several ways:

Length-based Shorter sequences are presented before longer ones, as longer sequences may contain more information and complex dependencies for the model to learn.

Frequency-based Examples with common and frequent vocabulary, edit operations, entities or sentence structures are presented before examples with rare or domain-specific vocabulary or structures, allowing the model to learn general patterns before handling exceptions.

Loss-based The model's own prediction loss for an example is used as a candidate for difficulty mapping - examples with a higher loss are introduced later in training [337].

Task-based For a multi-task setting (standard for most modern LLMs), simpler tasks or auxiliary tasks are trained first before training on the primary task, allowing the model to handle the task complexity incrementally [274].

Formally, given a training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, curriculum learning uses a difficulty scoring function $\phi(x_i, y_i)$ to rank training examples, and a pacing function $\lambda(t)$ to control which examples are available to the model at training step t:

$$\mathcal{D}_t = \{(x_i, y_i) \in \mathcal{D} \mid \phi(x_i, y_i) \leq \lambda(t)\} \tag{2.15}$$

where $\mathcal{D}_t$ is the subset of training data available at step t, with $\lambda(t)$ gradually increasing over training to introduce more difficult examples. Curriculum learning has been shown to improve convergence speed and model performance on various NLP tasks [269]. In this thesis, I utilize curriculum learning for edit-specialized GEC model training in Chapter 4, where the model is trained in progressive stages on GEC datasets of increasing formality and complexity, with the final stage fine-tuning on the most task-relevant data to optimize model performance.

2.2.6 LLM Inference and Prompting Approaches

As discussed above, one of the main benefits of using instruction-tuned LLMs is their ability to solve multiple tasks during inference using natural language task description prompts, without requiring any additional training or model weight updates. The design and content of the input prompt have an impact on the quality of the LLM output [170]. In this section, I describe the prompting approach relevant to this thesis: Zero-Shot, Few-Shot, In-Context Learning and Retrieval-Augmented Generation.

Zero-Shot and Few-Shot Prompting

In **Zero-Shot prompting** [25, 308], the LLM is provided with a natural language instruction describing the task, without any task input-output example pairs.

$$y = \text{LLM}(\text{Instruction} + x; \theta) \tag{2.16}$$

where x is the input and y is the LLM generated output. The model's task-solving ability depends on the instructions and the knowledge captured in its parameters during the pre-training and instruction-tuning stages. Zero-shot prompting is incredibly helpful when there's limited (or no) task-example data.

Few-Shot prompting [25] extends zero-shot prompting by adding a small number of k input-output demonstration examples $(x_1, y_1), \ldots, (x_k, y_k)$ to the prompt, before the actual test input x:

$$y = \text{LLM}(\text{Instruction} + (x_1, y_1), \ldots, (x_k, y_k) + x; \theta) \tag{2.17}$$

The model observes the task format and expected output style from the examples without any explicit parameter updates. This ability to learn from in-context demonstrations is referred to as **In-Context Learning** (ICL) [63]. Brown et al. [25] showed that the GPT-3 LLM can match or improve upon the performance of fine-tuned smaller models on many NLP benchmarks with few-shot prompting. The selection of demonstration examples for any task, their ordering and the structured format of the prompt have been shown to impact LLM task performance significantly [191, 169], making prompt design an important step when using LLMs for downstream tasks.

Retrieval-Augmented Generation (RAG)

A key limitation of LLMs is that their factual knowledge is fixed at training time (depending on the training dataset) and can become incorrect, outdated and lack domain-specific

detail. Retrieval-Augmented Generation (RAG) [159] handles this issue by adding a retrieval component that fetches the most relevant documents/text from an external knowledge base and adds it to the input context, providing the LLM with helpful and domain-specific information. RAG has been applied to open question answering, dialogue assistants, summarization, etc. and has been shown to improve factual information and domain specificity of the LLMs output. A detailed survey of RAG methods, strategies and applications is presented by Gao et al. [75].

In this thesis, I utilize Zero-Shot and Few-Shot In-Context Learning approaches for GEC and ASR correction tasks, described in the following section.

2.3 Automatic Speech Recognition Systems

The Automatic Speech Recognition (ASR) task aims to design models that can map an input audio signal to its corresponding textual representation (transcript). ASR is a heavily researched, widely applicable and complex task; with model performance varying significantly depending on the type of audio, number, language and accent of the speakers, and environmental/background noise of the audio recording [182, 245]. Over the past decades, ASR models have progressed from rule-based phonetic matching systems to domain-adaptive, multilingual data-driven techniques trained using modern deep learning architectures. Recent years have also seen ASR systems being integrated in various commercial products, including virtual assistants (e.g., Siri, Alexa, Google Assistant) [130], call-centre support systems, and real-time captioning [295].

The work done in Chapter 3 focuses on correcting transcription errors in customer support call recordings in both English and French, where the ASR transcription output is generated using a proprietary web speech-to-text service with no access to the model internals, and the original audio recordings are unavailable due to privacy restrictions. The supervised training data is limited to a small number of manually annotated transcript pairs (∼4000 bitext), alongside a large pool of unannotated ASR outputs. Here *bitext* refers to paired data: (ASR hypothesis, aligned annotated target transcript), which can be utilized for direct supervised training. Similarly, *unitext* refers to monolingual, unpaired, or unlabelled data. I use it to represent the raw ASR transcript set with no corresponding reference.

The goal is to improve transcription quality under these low-resource, noisy data constraints using a synthetic data augmentation approach. The remainder of this section provides the background information for ASR Systems and my task.

2.3.1 From Statistical Models to End-to-End Systems

Early ASR systems, beginning with the Bell Lab Audrey system in the 1950s for digit recognition [55], relied on handcrafted template matching against manually defined phonetic patterns [193]. These approaches were engineered specifically for the data/domain and usually failed to generalize across speakers or acoustic environments. The application of *Hidden Markov Models* (HMMs) to design ASR systems in the 1970-1980s provided a general statistical framework for modelling speech as a sequence of probabilistic events. The combination of HMMs with *Gaussian Mixture Models* (GMMs) provided an ASR Model architecture that was popular and used for decades [62].

In a GMM-HMM system, acoustic features (spectral representation like MFCCs - *Mel-Frequency Cepstral Coefficients*) are extracted from short overlapping audio frames. The GMM estimates the likelihood of the acoustic features at each HMM state and the HMM captures the temporal structure of the speech signal. During decoding, the acoustic model is combined with a separately trained language model and pronunciation lexicons to output the most likely word sequence. However, GMM-HMM systems required a large amount of labelled speech, hand-engineered pronunciation lexicons, and domain-specific language models, making adaptation to new languages or domains expensive.

The application of deep neural networks in the early 2010s to ASR introduced the *Deep Neural Network–HMM* (DNN-HMM) hybrid model, where the GMM model was replaced with a DNN acoustic feature classifier, yielding substantial WER reductions [103]. This was followed by using various Neural Network architectures: *Recurrent Neural Networks* (RNNs) and *Long Short-Term Memory* (LSTM) (Section 2.2.2) networks for modelling the sequential audio dependencies better, and *Convolutional Neural Networks* (CNNs) for understanding the local spectro-temporal structure in the audio spectrogram [226].

One of the major advancements in ASR System architecture is the introduction of *end-to-end* (E2E) ASR, in which a single neural network is trained to map the input audio features directly to text. There are two main E2E-ASR decoding approaches. *Connectionist Temporal Classification* (CTC) [86] trains the model by considering all possible ways the output tokens can be aligned to the input audio frames, removing the need for manually labelled frame-level alignments between the speech audio and text. *Attention-based encoder-decoder* (AED) models learn a soft alignment between encoder states and decoded tokens during training, allowing for slower but longer-range context autoregressive decoding [160]. Recently, the transformer architecture [296] has become the standard for both approaches, and hybrid CTC-attention training is now common practice.

E2E systems have simplified the ASR training and inference pipeline significantly. However, this simplification has a direct consequence for correction. Because linguistic

information is now captured within the E2E network weights rather than isolated in a separate language model, it can't be updated or replaced without retraining. Peyghan et al. [216] note that E2E architectures jointly learn acoustic, pronunciation, and language information but cannot exploit large unpaired text corpora without explicit integration strategies. This is why ASR Transcription post-editing correction has become an active research area; to utilize linguistic and domain knowledge to improve an ASR system's (with fixed weights) predictions.

2.3.2 Multilingual and Low-Resource ASR

One of the primary factors dictating the performance of neural ASR systems is the amount of transcribed speech data available for training. For high-resource languages like English, there are a lot of speech transcribed datasets with thousands of hours of aligned audio. For many other languages, comparable resources do not exist. For instance, the Multilingual LibriSpeech (MLS) [225] dataset contains roughly 44,500 hours of English audio but distributes only 6,000 hours of audio across the other seven languages combined.

To overcome this data sparsity for low-resource languages, three main strategies have been explored in the literature: (1) *cross-lingual transfer* where a model trained on one or more high-resource languages is adapted to the target low-resource language [106, 131]. (2) *self-supervised pretraining* used by models such as wav2vec 2.0 [5] and HuBERT [109] in which they learn general speech representations by pre-training on large quantities of unlabelled audio data, requiring only a small amount of labelled data for downstream task fine-tuning. (3) *large-scale weak supervision*, adopted by the transformer encoder-decoder ASR-LLM Whisper [234] which is trained on 680,000 hours of multilingual audio collected from the web. The Whisper model trades annotation quality for data quantity, achieving strong zero-shot transfer across languages and domains without the need for fine-tuning.

I use the Whisper model in this thesis to generate the French ASR source transcripts for the MLS dataset (Section 3.3.2), since the MLS dataset does not provide pre-generated ASR transcripts. Whisper's ability to perform ASR across multiple languages and domains should not be mistaken for correctness in all conditions. The model frequently hallucinates or ignores speech content during long silences, noisy or low-volume audio, and makes transcription errors on disfluent speech and rare domain vocabulary. Most of these transcription issues were also observed in the industry customer support call recording transcripts examined in Chapter 3, and I wanted to replicate these conditions for the MLS-French dataset. For English, I use the RED-ACE dataset [79], which provides the LibriSpeech audiobook recordings [210], the target text and ASR transcriptions generated using the Google Cloud Speech-to-Text API.

2.3.3 ASR Transcription Error Types

Depending on the type of audio recording, ASR models can frequently make erroneous transcriptions in noisy conditions - environmental noise [214], multi-speaker and overlapping conversation [33], or domain-specific terminology with low coverage in the training distribution. ASR errors arise when the system incorrectly transcribes the speech input, and understanding their type is helpful for designing corrective strategies.

Substitution Errors (S) where one word/phrase is replaced with another (e.g., "four" → "for").

Deletion Errors (D) when the system ignores words or parts of utterances (e.g., quickly-spoken words like "to" or "the").

Insertion Errors (I) when the ASR model hallucinates or predicts extra words not present in the input speech.

The amount of these three errors, together, is used to compute the ASR evaluation metric: *Word Error Rate* (WER).

$$\text{WER} = \frac{S + D + I}{N} \tag{2.18}$$

where N is the total number of words in the target transcript. The word counts S, D, and I are obtained by computing the minimum edit-distance alignment (Levenshtein algorithm) between the prediction and target word sequences.

2.4 ASR Transcription Correction

There are two ways to improve the quality of an ASR system's output. (1) improve the ASR model itself, or (2) post-process its transcriptions. The first approach - designing better architectures or fine-tuning an existing model on additional paired speech-text data - is effective but requires significant computational resources and, more importantly, access to the model weights. Many ASR systems are deployed as proprietary cloud services and cannot be modified. Also, fine-tuning on limited domain data can degrade performance due to overfitting. Due to these constraints, there has been a lot of work on trying to improve the ASR transcript directly without modifying the base ASR system.

In this thesis, I adopt the terminology proposed by Peyghan et al. [216] for ASR output refinement techniques. *Intrusive* methods that change the ASR model's architecture, parameters, or training process and *Non-intrusive* methods that focus only on the output of an ASR System - the 1-best transcription, n-best hypothesis list, or a word lattice - and sometimes the raw input audio. Their survey provides a comprehensive review of many non-intrusive correction approaches for ASR transcripts, which they group into five classes - fusion, rescoring, distillation, training adjustment, and correction.

2.4.1 Non-Intrusive Transcript Refinement

Fusion augments the ASR decoding step with an external language model [335]. Mainly in low-resource and domain adaptation situations with unpaired text data [223]. *Shallow fusion* approaches interpolate the ASR and LM scores during the inference decoding step and require no architecture modification or training, whereas *deep* and *cold fusion* techniques combine the ASR and LM hidden states, requiring semi or full training of the ASR decoder. A comprehensive survey and comparison of these techniques is provided by Toshniwal et al. [289]. All fusion-based approaches require access to the iterative decoding loop.

Rescoring evaluates and ranks the hypotheses produced during the ASR decoding using an external scoring model; it can be done on the word lattice while pruning or on a generated n-best transcription list [314]. Rescoring-based prediction selection performance is upper-bounded by the n-best predictions, and it requires the ASR system to generate multiple scored hypotheses.

Distillation transfers linguistic knowledge from a large external LM into the ASR model during training, improving the ASR Model architecture, resulting in no additional inference cost [73]. **Training adjustment** uses various carefully planned ASR training objectives and model optimization techniques to improve the ASR baseline model [224, 43]. Both approaches require complete retraining of the ASR model and a significant amount of training data.

Correction refinement approaches generate a new transcript from the initial ASR output. This is the only class that does not require access to the ASR model after its final text prediction, and the only method applicable to my setting when the ASR system is a black box returning a single transcription. Correction models generate rather than select (like rescoring methods), and can produce words that were absent from the ASR hypothesis.

The low-resource data constraints in this thesis make *correction* refinement the only feasible technique to design an ASR Transcript Correction system. The industry ASR transcripts are generated using a proprietary web service that provides a single transcription

with no lattice, no n-best list, no confidence scores, and no access to modifying the iterative decoding step. I also do not have access to the original audio signal that is being transcribed. This excludes a large portion of the ASR correction methods since many of the strongest reported correction results condition on multiple scored hypotheses [347, 176] or fine-tune on the audio-text data for low-resource domain adaptation [223]. I only have access to a few thousand supervised annotated sentence pairs and a large amount of unpaired source transcripts. With no additional conditioning knowledge available at inference time, I focus on designing a correction approach that makes use of synthetic data augmentation described in Section 3.6.

2.4.2 ASR Correction Model Architectures

Rule-Based and Statistical Approaches

The initial correction systems were rule-based or built on n-gram heuristics. For instance, ROVER [71] aligns the single hypotheses from multiple ASR systems into a word transition network and selects the target word using a voting scheme at each position. Bassil and Semaan [10] make use of a large n-gram dataset to detect ASR word errors, generate correction candidates and select replacements based on occurrence frequency. Approaches like this are not able to capture the diversity of error patterns across domains, and n-gram context cannot capture the long-range dependencies.

Neural Network Correction Models

Tanaka et al. [279] introduced an encoder-decoder RNN based error correction model trained on N-best hypothesis lists and ASR confidence scores. Hrinchuk, Popova, and Ginsburg [108] utilized a Transformer model with the encoder and decoder initialized using BERT (Section 2.2.2) to correct the ASR 1-best transcript. Zhao et al. [345] showed that the BART model[158], which is pre-trained to reconstruct clean text from corrupted input, outperforms BERT-based models on Mandarin ASR correction. The similarity between the BART denoising pre-training objective and the ASR correction task dictates my choice of using the BART model for English and the BARThez model [124] for French ASR Correction in Section 3.7.1.

The work closest to my own is done by Dutta et al. [64], where the BART model was trained for accented speech transcript correction. Due to a limited amount of training data, synthetic data was generated, and additionally the training was also conditioned on

grapheme-to-phoneme sequences. My approach differs in two ways. First, their synthetic errors are generated by transcribing the *training audio* with weaker ASR models, which is not possible in my setting. Second, introducing phoneme-based synthetic substitutions and model input is less helpful for the disfluent conversational speech and multilingual transcripts examined in Chapter 3.

There have been many solutions that deal with the low-resource data setting of ASR Correction. Mai and Carson-Berndsen [181] transcribe the input audio with a weak and a strong ASR model, using the weaker output as source and the stronger output as target. Jia et al. [116] use LLM prompting to generate training data based on real ASR error patterns extracted from classroom recordings. Nanayakkara et al. [196] work on clinical dialogue correction with no paired data by using proxy objectives on domain text - summarization, paraphrasing, and mask-filling. They show mask-filling to be the most effective due to its similarity to the correction task. This differs from my approach since they use the masked language modelling objective to teach the model medical domain vocabulary, whereas the synthetic pre-training stage in Chapter 3 trains the model directly to perform corrections as a sequence-to-sequence task, using data that replicates the error distribution of the deployed ASR system.

A consistent finding in recent low-resource and domain-specific ASR correction work is that the compact task-specific fine-tuned models outperform general-purpose LLMs, even much larger ones. Bhandari and Harit [17] show that a simple character-level sequence-to-sequence model for the Rajasthani language outperforms both a fine-tuned mBART-50 model, zero-shot GPT-4o and Gemini 2.5 Pro. Kumar et al. [145] observe the same conclusion for Hindi, Marathi, and Telugu language ASR correction models- fine-tuned mT5 and ByT5 models [315] outperform substantially bigger LLMs under both fine-tuning and in-context learning settings. This makes sense since a smaller model trained directly on the ASR target domain's error patterns can perform more focused and relevant corrections than a generalized instruction-following LLM at inference time. These results are consistent with what I observe in Chapter 3, where the synthetic data augmented BART model outperforms the instruction-tuned LLMs on the low-resource ASR correction task.

LLM-Based Correction

Initial work on using LLMs for ASR transcript correction produced mixed results. Min and Wang [192] found that asking an LLM to correct the 1-best hypothesis *increased* the WER compared to the source transcript, since the LLM rephrases text that was already correct. The performance improves when the LLM was shown multiple hypotheses. This has motivated a new direction of LLM-based ASR correction - *Generative Error Correction*

or *Generative Speech Error Correction* (formalized as the GenSEC task at IEEE SLT 2024 [318]). Unlike N-best rescoring (Section 2.4.1), which selects or re-ranks a fixed set of candidates, the GenSEC approach relies on instruction-tuned LLMs to *generate* a new transcript conditioned on all N hypotheses, thus being able to produce edits that may appear in none of them (unconstrained generation). There has been a lot of recent work in this area focusing on prompting and in-context examples [317, 176, 253], parameter-efficient fine-tuning on N-best data [36, 149], and conditioning the LLM on additional input signals like as acoustic embeddings, or retrieved domain entities [194, 227]. Radhakrishnan et al. [236] extend this further by introducing a fusion-based (Section 2.4.1) Whispering-LLaMA refinement model that augments Whisper-generated audio representation features into a frozen LLM model via adapters, achieving a significant improvement over the N-best generation approach by grounding the LLM generation with the true speech signal.

These approaches represent an upper bound for the ASR correction task when there is access to the ASR decoder and the audio signal. My task has neither. The industry ASR dataset only consists of a single 1-best transcription through a proprietary web ASR service, no confidence scores and no call audio data for privacy reasons. This rules out using the full GenSEC framework as it is. Since the N-best hypotheses used for standard ASR error correction are fixed, they can limit the error distribution the LLM is exposed to. This limitation has been explored in recent work by generating hypotheses dynamically [194] and through LLM prompt addition [292]. The approach proposed in Section 3.7.3 addresses both constraints by sampling the Synthetic Data Augmentation confusion model described in Section 3.6.3 with different thresholds to generate synthetic 5-best correction candidates for each source transcript input, allowing me to utilize LLM-based GenSEC correction without access to the ASR decoder, audio and confidence scores. The resulting LLM corrections achieve a better WER score compared to the prompting and in-context examples LLM baseline for ASR correction.

2.4.3 The ASR Over-Correction Issue

The generative capability of ASR correction models that makes them expressive also makes them unsure. A model that can freely edit or transform its input can substitute a semantically similar word, normalize valid transcription into another, or, specifically with LLMs, hallucinate or paraphrase content. This is one of the primary risk of using the correction paradigm, and it is damaging when the initial WER is low, or, when the audio signal is unavailable [216].

The problem is worsened by a data distribution mismatch. Correction models are initialized using language models pre-trained on large amount of written text. This is

helpful to reduce the training time and allow better domain adaptation (Section 2.2.3). However, the ASR output transcribed for conversational speech contains disfluencies, repetitions, partial words, and pauses. A model trained on the denoising objective - like BART, whose prior prefers fluent text will tend to transform the input even when the transcript is accurate for the input audio. The overcorrections issue is very common when using LLMs for text correction tasks [212, 174]. Various strategies have been explored to handle overcorrection edits, such as error detection then correction modules that restrict correction spans [341, 329] and constrained decoding that limits the allowed edit operations [320]. The edit filtering strategy proposed in Section 3.7.4 follows a similar motivation to reduce the overcorrections and is applied as a post-processing step rather than architectural modifications to the correction model.

2.4.4 Synthetic Data Generation for Correction

To overcome the limited ASR supervised correction data in low-resource and domain-specific settings, a natural approach is to generate synthetic training data that mimics the domain error distribution of the deployed ASR system. There are three main approaches to producing synthetic ASR error data.

The first approach uses a text-to-speech (TTS) system to generate the audio from various text inputs, re-transcribes the resulting audio using an ASR model, producing different ASR errors on arbitrary text [302, 92]. This produces realistic error patterns, but requires a TTS and ASR system (compute expensive) and the generated data may not match the error characteristics of the low-resource target domain (like background noise, and overlapping speakers).

The second method inserts errors into clean text using a trained neural model. Leng et al. [156] use BERT based token masking and sample replacement preferring homophone words. Hrinchuk, Popova, and Ginsburg [108] apply random token deletion, replacement, and shuffling for the same purpose. These methods are cheaper than the TTS route, but the synthetic errors do not reflect the actual confusion of the deployed ASR system.

Using neural based models to introduce errors in the first two approaches also make the error introduction approach uncontrollable. The third approach focuses on learning a model of the ASR system's error behaviour from available data and samples edits from it. Schatzmann, Thomson, and Young [257] formulated this as a fragment-to-fragment translation approach for generating ASR errors in spoken dialogue systems: an utterance is divided into contiguous fragments, and for each fragment a possible edit is drawn from the

confusion model. Fazel-Zarandi et al. [68] and Wang et al. [300] subsequently validated and extended this framework for training dialogue policies under noisy input conditions.

The third approach is the one adopted in Chapter 3, for two reasons. First, since the confusion distribution is estimated directly from the limited supervised bitext, the synthetic generated data mimics the error rate and error-type distribution of the proprietary ASR system on domain data. Second, it does not require the audio data. Although, there are a few limitations of this approach - the confusion matrix can only generate error patterns that were observed in the small annotated sample, and cannot extend to unseen vocabularies.

A further benefit of using this confusion distribution directly is that it can be sampled repeatedly. Varying the fragment partitioning and then sampling substitutions can easily produce many distinct synthetic corrections/corruptions, which is used as a synthetic n-best list. Apart from Seq2Seq pre-training on synthetic data, generating multiple candidates from the confusion model is used for the N-best LLM correction method evaluated in Section 3.7.3.

2.5 Grammatical Error Correction Models

Grammatical Error Correction (GEC) is a popular and interesting task in the field of NLP. The objective of automated GEC (as the name suggests) is to design models that can correct different grammatical and discourse-level inconsistencies in text (retaining all the content information) [29]. A model that can sufficiently perform this task can benefit a large portion of the community. Non-native language speakers can use it as a proofreader and gain significant insights regarding how they are learning the language. Native speakers can also benefit from such a system to ensure that they do not introduce inconsistencies by mistake. Even though the scope of my work focuses on GEC systems for the English language, there is significant work in GEC systems for other languages (example Chinese GEC [228]). Such systems are important for different reasons, such as treating the GEC task as a low-resource translation task. However, I don't focus on systems designed for other languages in this work.

GEC is a difficult task due to various reasons. Depending on the writer and the area, GEC encompasses a wide variety of correction possibilities. While some of them are universal and straightforward, like spelling, tense, plurality-singularity consistency, etc., others might be much more complex, like style or formality (discourse level). Errors and issues like these don't have a fixed set of rules to follow while performing corrections. As a result, there is a diverse set of possible hypothesis predictions of a GEC system that could

be considered appropriate and correct. Considering this in addition to the inherent property of language being diverse on its own (since the same thing can be phrased in multiple ways), GEC is definitely a non-trivial task. Apart from these issues, the automated GEC task, like most supervised learning tasks, is dependent on the datasets they are being trained and tested on [278, 199]. Datasets and systems can also have sentence length restriction and as a result might not be able to capture inter-sentence and long-range dependency errors [339]. Depending on the task which provided the training dataset there can also be a difference between the distribution of possible errors. As a result, models trained on one dataset might not be able to generalize and provide good corrections for another dataset.

There has been a lot of work in the field of GEC utilizing various approaches and models. The GEC models can often incorporate the tangential task of Grammatical Error Detection (GED) and Grammatical Error Explanation (GEE). GED deals with localizing and determining if an issue exists in the source text and can be further expanded if error corrections are also tagged by the type of grammatical issue [324]. The GED task can make use of this and try to predict the location and type of grammatical inconsistency and determine sentence quality [256]. However, GED and GEC are not completely independent. For example, the work by Omelianchuk et al. [203] utilizes a technique called tagged corrections, where the grammatical inconsistencies are identified first and then corrected. Li and Wang [164] used template Masking for error detection and subsequent Mask-Filling for performing GEC. Similarly, another recent task called Grammatical Error Explanation (GEE) has emerged [69, 267]. GEE focuses on providing a description or category label for each corrective edit made to a sentence. Similar to GED, recent work has explored integrating GEE in the framework for solving GEC [326, 327], observing better performance than GEC baselines. In this background section, I primarily focus on GEC-only approaches, i.e., producing a final rewrite of the source sentence that can be evaluated by a gold-set target for grammatical consistency and coherence.

The work presented in Chapter 4 focuses on improving a GEC baseline model by decomposing the correction task into the three atomic edits - insertion, deletion, and substitution correction task. Four sequence-to-sequence models are trained with the same base architecture (BART - Section 2.2.3) and edit-specific training text pairs (bitext) from the same publicly available GEC training data, and their predictions are combined using edit-level ensembling. Unlike most recent GEC work that improves performance using large synthetically generated pre-training corpora or complex model architectures, my focus is on utilizing the available training data and the same baseline architecture more effectively to keep the correction inference cost low. The remainder of this section presents the background for the GEC task, the model architectures, the edit-type and ensembling approaches relevant to my work, and the GEC datasets and evaluation metrics used for

evaluation.

2.5.1 From Rule-Based to Neural GEC Models

Historically, due to various factors like computational and model complexity of supervised learning, unavailability of public training corpora, etc., most of the GEC models utilized rule-based corrections. A rule-based GEC model tries to conform the incorrect sentence to a set of predefined rules for appropriate grammatical syntax. Macdonald et al. [177] designed a system that utilized string matching rules for error corrections. The model utilizes regular expressions and word-lists for grammatical correction and improvement. Other works in this area utilize similar structure rules derived from the target application. Some examples of similar systems are - ALEK [40, 152] and ESL Assistant [74]. These models were often improved by utilizing linguistic parsers for intra-sentence structural coherence and sophisticated grammar [98, 243]. Models like these are often restricted in the type of errors they correct and dependent on the performance of external tools like parsers. After the introduction of public GEC corpora the rule based models were improved to automatically extract string correction rules based on the errors observed in the training corpora [183, 252, 30, 322, 3]. Models like these are often limited by the frequency and type of issues observed in the training corpus.

Recently, due to the introduction of parallel training corpora, the GEC task can be thought of as a variant of the Machine Translation task. As a result, various models and techniques designed for the task of Statistical Machine Translation (SMT) and Neural Machine Translation (NMT) have been integrated in GEC models. SMT models are based on a parallel source-target corpus and rely on a phrase mapping table between the source and target and a language model to choose the best possible translation for the phrase [24, 70, 120]. Other SMT approaches for GEC utilize techniques like round-trip translation where an English input sentence is converted to another language and then back to English. The hypothesis being that double-translation will remove the grammatical ambiguity while retaining sentence content information [101, 178]. Other approaches utilize techniques like beam search output from other SMT models.

NMT models are based on a similar idea as SMT models, but they utilize Neural Network Architecture for training. NMT approaches have become widely popular in the field of GEC due to the readily available training corpora as well as the introduction of fast and strong computing power. The NMT architecture makes use of the Seq2Seq learning architecture proposed by Sutskever, Vinyals, and Le [277]. Various models, relying on hybrid NN architecture, have been proposed for the GEC task. The use of massively

pretrained transformers have also gained popularity for the GEC downstream task. Models like BERT [61] and GPT2 [235] have been fine tuned for GEC [203, 125]. The methods presented in my work utilize an architecture similar to the NMT model.

For a detailed summary and history of the GEC task, I recommend reading the comprehensive review prepared for the COLING-2020 GEC Tutorial[2] by Grundkiewicz, Bryant, and Felice [88] and the survey by Wang et al. [306].

2.5.2 Approaches to Grammatical Error Correction Models

In my work, I focused solely on GEC models that are trained and designed using neural networks or deep learning architecture, as they provide significant improvement and are considered to be state of the art with regard to their performance.

GEC models can be broadly divided into supervised-trained, LLM-based, and ensemble-ranked models [204]. The supervised models follow two architectural approaches: the first makes use of the popular (and now standard) Neural Machine Translation (NMT) architecture, which is built on the sequence-to-sequence (Seq2Seq) model. The second approach, sequence tagging, is based on an editing or tagging mechanism. It uses a tagged correction technique in which the model first tries to identify inconsistencies and then provides corrections for a localized text span (i.e., word-level correction).

Sequence-to-Sequence Models for GEC

The Seq2Seq architecture described in Section 2.2.2 has been used for the GEC task directly: the incorrect sentence is the source and the corrected sentence is the target, similar to the NMT source–target language pair [41, 91]. Figure 2.2 shows this on a GEC example.

The GEC task is different from translation, which has an impact on how GEC Seq2Seq models are built. The source-target in GEC are in the same language and most of the source tokens are copied unchanged to the target, with only a few spans edited (depending on the quality of the source). This leads to a few specific design choices for GEC models. First, model architecture with a focused copying mechanism, like the copy-augmented Transformer by Zhao et al. [344], which works better than plain Seq2Seq models. Second, denoising pretrained encoder-decoder models are shown to perform well for GEC, since reconstructing corrupted text is similar to the GEC task goal. The fine-tuned BART model obtains competitive performance with no GEC-specific synthetic data pretraining [128].

[2] https://github.com/grammatical/coling2020-tutorial

Third, an extension to the previous point motivates GEC-specific synthetic data pretraining to further improve performance, as shown by GEC-PseudoData (GEC-PD) [133] and Rothe et al. [247].

All models in Chapter 4 use BART (Section 2.2.3) as the Seq2Seq model.

Sequence Tagging Models

As mentioned previously, the sequence tagging approach is trained to detect inconsistencies in shorter, localized text spans. These models differ from the Seq2Seq models as they do not generate the complete target sentence. Rather, they go over each token in the input and generate labels such as "KEEP", "EDIT", and "DELETE" for them. As a result, they perform GED on each word or token to perform the overall GEC.

Some examples of this architecture can be observed in the GECToR [203] and PIE tagged-editing GEC models [4]. These models utilize the specific editing labels for each word in the source sentence to edit them.

Tagging models are fast at inference, but are restricted to the edit operations prediction vocabulary. I use Seq2Seq models in this thesis, and the edit-decomposition method in Chapter 4 considers the sentence-level targets rather than token tags.

LLM-Based GEC Methods

Recently, Instruction-tuned Large Language Models (LLMs) (Section 2.2.5) like Llama [290, 204] and ChatGPT [127] have been used to tackle GEC using different settings like: Zero-Shot, Few-Shot and Fine-Tuning [66, 140, 48, 54, 238].

LLMs are highly capable at generating fluent and natural rewrites of the source text and score well on fluency-oriented GEC evaluation. However, they consistently underperform smaller task-specific fine-tuned neural models on the standard minimal-edit GEC benchmarks. Fang et al. [66] show that ChatGPT scores lower than supervised baselines on the CoNLL-14 and BEA-19 test sets, even though human editors may prefer its corrections. Kobayashi, Mita, and Komachi [136] quantify this behaviour in the SEEDA GEC dataset, where GPT-3.5 makes two to three times as many edits as the fine-tuned neural systems on the CoNLL-14 test set. The primary reason for this issue is that LLMs tend to rewrite and paraphrase spans of text beyond the required grammatical corrections, which I discuss in Section 2.5.5. In Chapter 4, I do not use LLMs as standalone GEC models, but as ensemblers that evaluate and combine the edits proposed by the edit-specific baseline models.

2.5.3 Edit-Type Specialized GEC Models

The three atomic edit operations (insertion, deletion, and substitution) and the different error categories built on top of them have been used in various GEC works. The ERRANT GEC toolkit [27] is usually used to obtain these edit categories. ERRANT extracts edits between a bitext and classifies them into 54 linguistic/syntactical categories built on the three atomic operations. This has been used for GEC evaluation, synthetic data generation, training objective design, and system combination. It has also been explored how different GEC models have different performances across error types [231, 203].

The primary use of edit-label information is in GEC synthetic data generation. Choe et al. [41] design type-specific noising functions that mimic the error type distribution of the GEC corpus using ERRANT and use it to generate pre-training data from clean text. Recently, Wang et al. [305] extended this approach with contextual data augmentation, generating errors conditioned on the error type and the context sentences to better match the GEC data error distribution.

Another area of work uses the edit type information in the model training objective. Bout et al. [23] design a set of auxiliary sequence-to-sequence tasks derived from GEC source-target alignment like - predicting the sequence of edits and applying a given edit sequence to the source text, and train their GEC model on the whole dataset in a multi-task setting. Lai et al. [150] trains the GECToR [203] sequence tagging model (which uses edit-type labels in its output vocabulary) in a Type-Driven Multi-Turn Corrections (TMTC) approach, where multiple edit-specific bitext are constructed from each source-target pair, each containing a single error type. The base model is then trained in turn, so that it becomes aware of the gradual correction procedure and the interdependence between different correction types. TMTC proposes a curriculum-learning approach to teach GECToR the edit labels at separate training stages; however only improves marginally over the baseline GECToR model. Tarnavskyi, Chernodub, and Omelianchuk [280] obtain comparable improvements on GECToR by scaling up the sequence tagger and ensembling multiple large variants, which suggests that the benefit of edit-type staged training on its own is limited. Training models on restricted and strategic training input subsets has been explored in the curriculum learning [269, 303] and multi-task learning [37, 336] literature (Section 2.2.5), where filtering or re-weighting the training data is shown to improve specialized task performance.

A related direction separates Grammatical Error Detection from Correction, using the detected edit labels to focus where corrections should be applied. Li et al. [166] propose TemplateGEC, which transforms the detection output into a template that constrains the correction model. Li and Wang [164] use a general language model where the detection step generates the edit labels used for localized correction, evaluating it as a single model

and as separate detect-then-correct approaches. The GECToR [203] and PIE [4] sequence tagging models can also be considered as similar models, since they predict edit operation labels for each input token instead of generating the complete output text.

All of the approaches above make use of a specialized model architecture or a large synthetically generated pre-training corpus, such as the 200Million sentence C4 or the 9M sentence PIE dataset. In contrast, the work in Chapter 4 focuses on keeping the simple baseline architecture and using the same publicly available GEC BEA-2019 training data, and obtains improvements by decomposing the task by edit type across multiple models and then ensembling their predictions. The objective is to make better use of the limited annotated GEC data available, rather than to add more data or parameters that make inference expensive.

2.5.4 Ensembling and System Combination

The current state-of-the-art GEC tools all rely on the approach of ensembling multiple strong GEC Tools, aggregating them with methods like majority votes [280] and logistic regression [232, 231]. Combining the predictions obtained from multiple GEC models is one of the most reliable ways to improve overall correction performance. Bryant et al. [29] distinguish two combination approaches in their survey. *Ensembling* averages the output probabilities of multiple individually trained models when predicting the next token or edit tag. *System combination* uses a separate combination approach over the individual outputs of all the base systems, and is preferable when the systems being combined differ substantially.

System combination is most effective when the base systems have complementary strengths. Qorib, Na, and Ng [231] report that state-of-the-art (SOTA) GEC models can often be less accurate at correcting certain error types than other weaker models, so combining these system outputs can produce a better correction prediction. Susanto, Phandi, and Ng [276] first demonstrated this for GEC by combining multiple systems using MEMT [97], a combination method proposed for machine translation that aligns system outputs and scores candidates using n-gram and length features. Kantor et al. [126] developed the IBM method, which groups edits from two systems at a time and learns to select among them per model-error type efficacy. Lin and Ng [168] extended this using integer programming (GEC-IP), which allowed multiple system prediction combinations in a single iteration. Qorib, Na, and Ng [231] proposed the Edit-based System Combination (ESC) framework, which trains a logistic regression model using ERRANT edit-type labels as features to score each candidate edit from the set of all system predictions for selection, outperforming the prior approaches while remaining simple and interpretable.

Two related approaches score the set of possible edits using different features. Sorokin [268] proposed EditScorer, which scores each elementary edit using different estimated features of the edit and its surrounding context, obtained from a RoBERTa (Section 2.2.3 representation, rather than the edit type. Qorib and Ng [232] propose GRECO, a quality estimation model that scores the complete edited sentence and can be used to select the best subset of edits for combination. A limitation of the ESC framework is that it does not consider the content of the edits or their surrounding context. To overcome these limitations, I also propose an LLM-based edit quality classification approach in Chapter 4, which evaluates each candidate edit using its content and sentence context rather than its edit type alone.

The main drawback of system combination is its computational cost, since inference has to be run on every base system before combination. Qorib, Aji, and Ng [230] address this with MoECE, a mixture-of-experts GEC model in which different sub-networks specialize in correcting different error types, removing the need for multiple inference passes. Their motivation is close to mine - obtaining the benefits of error-type specialization without paying the full cost of system combination - although the approach is considerably different. The approach proposed in Chapter 4 does not avoid this cost - all four edit-specific models require a forward inference before the edits can be combined. My focus is instead on whether decomposing the task by edit type improves correction quality when the training data and base architecture are held fixed. However, because all four models are fine-tuned from the same pre-trained checkpoint and differ only in the edit-specific training data, the setup is suited for parameter-efficient adapter training [107, 217], where a single shared base model attaches four lightweight edit-specific adapters. This would reduce both the storage requirement and, with parallel adapter inference, the cost of generating the four predictions. This is discussed as a direction for future work in Section 4.7. Using an LLM to judge or combine candidate system outputs has also bee explored in machine translation [111] and ASR N-best ranking [292], which motivates the LLM-based ensembling approaches I evaluate apart from ESC.

I use the ESC framework [231] to combine the edit-specific model predictions in Chapter 4 for three reasons. (1) it only requires the four system predictions. (2) it scores and selects the combination edits in a single pass, which keeps the combination cost negligible compared to the base model inference. (3) it uses the ERRANT edit-type labels as features, which aligns with the model training done on edit-type filtered data.

2.5.5 The Over-Correction Issue in GEC

Like ASR transcript correction (Section 2.4.3), GEC models can also introduce additional errors while attempting to correct the existing ones. The GEC task focuses on minimal edits - correcting the grammatical inconsistencies while retaining the source content and structure. This requirement is also observed in other text generation tasks such as style transfer, where the model must change some attributes of the source text while keeping similar content [119, 72]. However, a generative correction model can rewrite the input entirely. This is the main reason that the GEC evaluation metrics weigh precision higher than recall. The $F_{0.5}$ score used by the M2 [51] and ERRANT [27] scorers reflects the human preference that introducing a new error is worse than failing to correct an existing one.

This issue is very common with LLM-based correction approaches. Instruction-tuned LLMs tend to paraphrase, reorder and rewrite text beyond the required corrections [66, 48, 174, 212], generating fluent output that scores poorly against the minimal-edit references. Park, Do, and Lee [212] characterize this as the main limitation of LLM-based GEC, and also point that supervised fine-tuned models - are precise but have low recall, only making the corrections they are confident about.

Several strategies have been proposed to handle LLM over-correction. Detect-then-correct pipelines restrict the model correction to error-detected spans [166, 164], and sequence tagging models constrain the model to a fixed vocabulary of edit operations instead of free text generation [203, 4]. For LLM-based correction, constrained prompting instructs the model not to introduce any edits beyond the provided candidate set. Both LLM ensembling approaches proposed in Chapter 4 (No-Extra (NE) prompt and the edit quality classification method) follow this method, restricting the LLM to only evaluate and combine the edits already proposed by the edit-specific base models rather than generating a correction freely.

2.5.6 Dataset

There are many available public datasets for the task of GEC. All the datasets are obtained from different areas and accumulated, annotated, corrected and shared with the community for research and evaluation of GEC systems. In this section I present some of the datasets that I utilize to evaluate my methods. One of the most popular datasets is the CONLL-14 presented by Ng et al. [200]. This is a test dataset provided primarily for evaluation of

models in the 2014 CONLL Task on GEC[3]. This dataset was an extension to the 2013 CONLL shared task on GEC. The models in this task were trained on the NUCLE dataset presented by Dahlmeier, Ng, and Wu [52]. The NUCLE dataset was prepared at the Centre for English Language Communication (CELC) at NUS. It uses English essays submitted by around 1400 undergraduate students at NUS and have the error type tags and the corrections mentioned. The other popular datasets for GEC were prepared and presented in the 2019 BEA Shared Task on GEC[4]. The dataset is referred as the W&I [322] and LOCNESS [85] dataset. The W&I dataset was accumulated from the Write and Improve web platform to assist non-native English speakers by providing correction suggestions on submitted text. The LOCNESS dataset is a collection of essays by native English speakers. For the BEA-2019 Shared Task, annotators manually went through the two datasets and provided a significant annotated and corrected dataset for training and evaluation purposes. The FCE dataset [323] is another corrected dataset present for the task of GEC. It is based on the Cambridge Learner Corpus[5] obtained from submissions by non-native English speakers taking the Cambridge ESOL exam[6]. The last dataset is known as the JFLEG Corpus [199]. The JFLEG Corpus extends the GUG Corpus [99] which was gathered from English language learners. The JFLEG corpus provides several rewritten and fluent corrections on the GUG corpus by multiple annotators.

2.5.7 Evaluation Metrics and Approaches

Even though Human Evaluation of NLP models is incredibly beneficial and insightful, it is very expensive and time-consuming. As a result, depending on the NLP task, the model is evaluated using different automatic metrics. For instance, the BLEU score [211] is one of the most popular metrics for evaluating text-generation tasks like Summarization and Machine Translation. For the task of GEC, various automated evaluation approaches have been researched and used in practice.

Early work on evaluating grammaticality relied on a combination of language model scores and linguistic features. N-gram language models were used to assign grammaticality scores, with the assumption that ungrammatical word phrases would get lower scores (being less probable) using a model trained on a large corpus of correct text [299, 99]. Apart from this, hand-crafted Grammatical templates and rules designed to accept structurally

[3] https://www.comp.nus.edu.sg/~nlp/conll14st.html

[4] https://www.cl.cam.ac.uk/research/nl/bea2019st/

[5] https://www.languagesciences.cam.ac.uk/news/cambridge-english-corpus-available-academic-use

[6] http://www.cambridgeesol.org/

well-formed sentences were also used to detect grammatical errors by checking whether a sentence could be parsed [299]. Okanohara and Tsujii [202] proposed a discriminative alternative, training a model to directly classify sentences as correct or incorrect using pseudo-negative examples sampled from a probabilistic language model, rather than relying on raw probabilities alone. However, most of these approaches have specific limitations for the GEC evaluation setting: n-gram probabilities are sensitive to sentence length and word frequency, making cross-sentence comparison unreliable; parser-based methods struggle with out-of-vocabulary word errors and degrade for longer sentences; and none of these approaches focus on the validity of the specific edits made by a GEC system against a reference correction. As a result, metrics designed specifically for evaluating the GEC edits were introduced. The most widely used among these are the GLEU, M2, and ERRANT scores, each of which I describe below.

The **GLEU** score metric is an extension and modification of the BLEU score [198]. It utilizes different n-grams present in the original sentence, the predicted output, and the reference ground truth sentence. Unlike standard BLEU, GLEU removes the dependence of scoring on edits alone. Specifically, n-gram spans that match the reference sentence but not the original source sentence are rewarded, while spans that match the original source sentence but not the reference ground truth are penalized in the overall sentence score. This ensures that the metric rewards genuine correction edits rather than simply preserving the original text.

The **M2 scorer** [51] evaluates GEC output by computing precision, recall, and $F_{0.5}$ score based on the edits between the original sentence and the GEC model prediction or ground truth sentence. Rather than comparing complete sentences directly, the M2 scorer aligns the sentence pair, extracts the set of atomic edits from the source-hypothesis and source-reference pairs, and then compares these edit sequences. The edits are stored in the M2 file format (Section 2.5.7). The $F_{0.5}$ score puts greater emphasis on precision than recall, reflecting the intuition that introducing new errors is worse than failing to correct existing ones.

The **ERRANT scorer** [27] improves and builds on the M2 scorer. It makes use of the M2 file format and follows a similar precision-recall-$F_{0.5}$ scoring setup, but uses syntactic and linguistic features to extract better-aligned edits and additionally performs automatic error type classification for each edit (Section 2.5.7). This allows for a more fine-grained evaluation of GEC systems by breaking down performance details across different grammatical error categories (e.g., noun number, verb tense, spelling). ERRANT uses a specific set of linguistic rules to determine if sequential atomic edits are related, to combine them and assign error type labels, making it particularly useful for detailed analysis.

While these automated metrics are practical and widely adopted, they have specific issues. All three metrics rely on comparisons to a fixed set of reference corrections, and since GEC is a subjective one-to-many task (multiple valid corrections can exist for a single erroneous sentence), reference-based metrics can underestimate the quality of valid unseen corrections. Furthermore, these metrics do not account for the downstream usability of a GEC tool from a human editor's perspective. For instance, a system that makes various small, confident corrections may score well on reference-based metrics but still require significant human post-editing effort to reach the desired final output. This gap between automatic metric scores and real-world GEC post-editing cost motivates the development of human-centric evaluation approaches, such as measuring Post-Editing (PE) effort. PE effort has been studied across three levels [142]: technical effort (number of edits), cognitive effort (mental effort of identifying and correcting errors), and temporal effort (total time taken to evaluate and perform post-edits). I discuss this further in Chapter 5, where I introduce the PEET Scorer as a temporal post-editing effort metric for GEC tool evaluation.

GEC Task Evaluation Example and the M2 Format

Source (S)	Surrounded by such concerns, it is very likely that we <u>are</u> distracted to worry about these problems.
Target (T)	Surrounded by such concerns, it is very likely that we <u>will be too</u> distracted to worry about these problems.
Model Output (MO)	Surrounded by such concerns, it is very likely that we are distracted <u>from worrying</u> about these problems.

Table 2.1: *Source*, *Target* and example *Model Output* made by a GEC Tool.

Evaluating a GEC Tool requires the Source (S), Target (T) and Model Output (MO) text. Table 2.1 provides an example GEC triple. The M2 and ERRANT scorers generate an M2 file for a pair of sentences (S-T, S-MO), which contains the edits to transform S into T/MO and the positions of those edits. In the M2 file, the `S` line is the tokenized source sentence and each `A` line is a single edit. The evaluation process generates two M2 files: (Source - Target) and (Source - Model Output) and then compare the edits to evaluate the MO quality.

Source - Target M2 File

```
S Surrounded by such concerns , it is very likely that we are distracted to
worry about these problems .
A 13 14|||R:OTHER|||and|||REQUIRED|||-NONE-|||0
A 11 12|||R:VERB:TENSE|||will be|||REQUIRED|||-NONE-|||1
A 12 12|||M:ADV|||too|||REQUIRED|||-NONE-|||1
```

Source - Model Output M2 File

```
S Surrounded by such concerns , it is very likely that we are distracted to
worry about these problems .
A 13 14|||R:PART|||from|||REQUIRED|||-NONE-|||0
A 14 15|||R:VERB:FORM|||worrying|||REQUIRED|||-NONE-|||0
```

ERRANT Edit Types

The ERRANT tags each edit with an edit *operation* and the edit *type* label (Figure 2.5). The operation is one of the three atomic edits - Replacement (R, substitution), Missing (M, insertion) and Unnecessary (U, deletion). The type is one of 24 linguistic categories based on the part-of-speech, morphology and orthography of the edited tokens. The full edit label has both, for example `R:VERB:TENSE` in the Source - Target M2 file above. Some edit types don't combine with the all three operations - the morphological (e.g., VERB:SVA) and orthographic (e.g., SPELL, WO) types only occur as replacements, which results in 54 edit labels.

I make use of these edit labels at all three levels in this thesis. In Chapter 4, the three edit operations (R, M, U) are used to create the edit-specific training datasets (Section 4.3.1), and the ESC ensemble uses the 54 edit labels as its edit features (Section 4.3.2). In Chapter 5, I use the ERRANT edit counts as features to train the PEET Scorer model (Section 5.5.2).

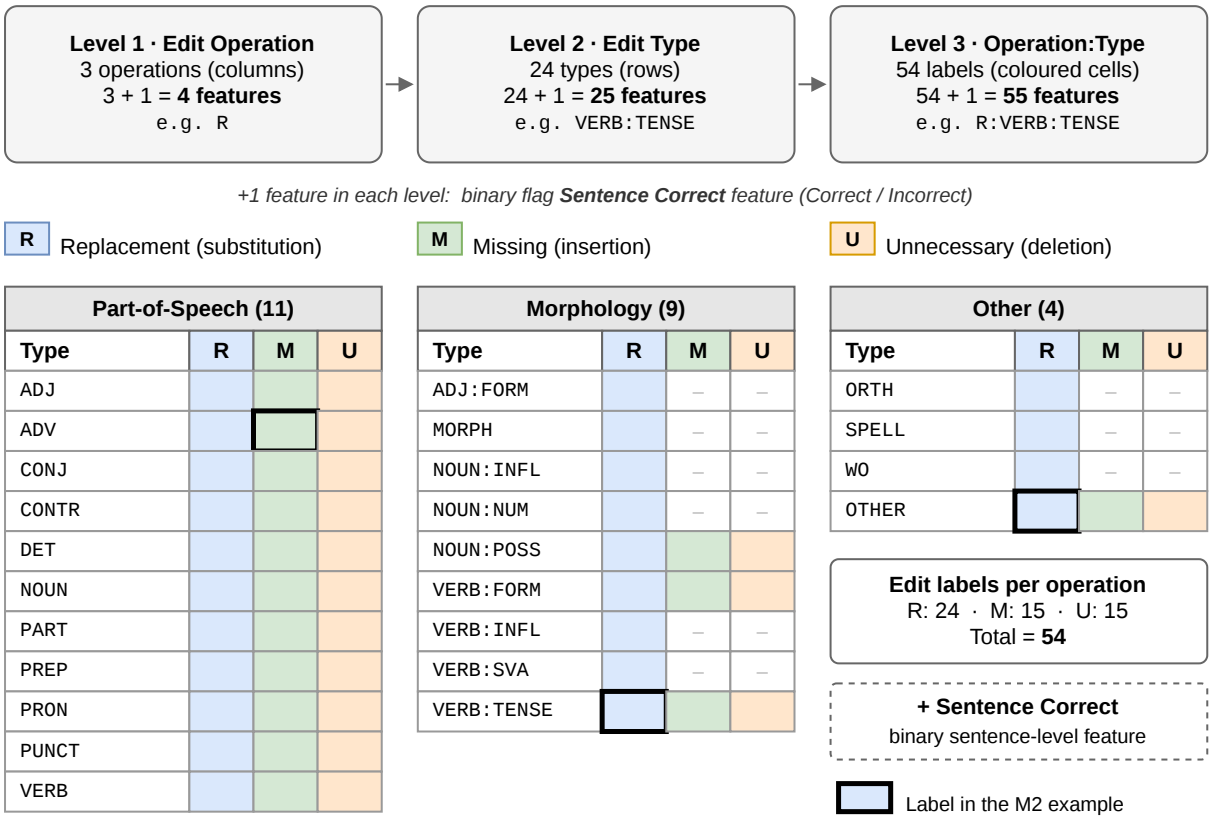

Figure 2.5: ERRANT edit operations and types. The three edit operations (columns) and 24 edit types (rows) are combined to get the 54 edit labels (coloured cells).

Chapter 3

Low-Resource ASR Transcript Correction

3.1 Introduction

In this chapter, I focus on Post-Editing models for correcting speech conversation transcripts generated by an Automatic Speech Recognition (ASR) model. The task focuses on generating the correction (what was actually spoken) for a noisy source transcript obtained from the ASR model (Figure 3.1).

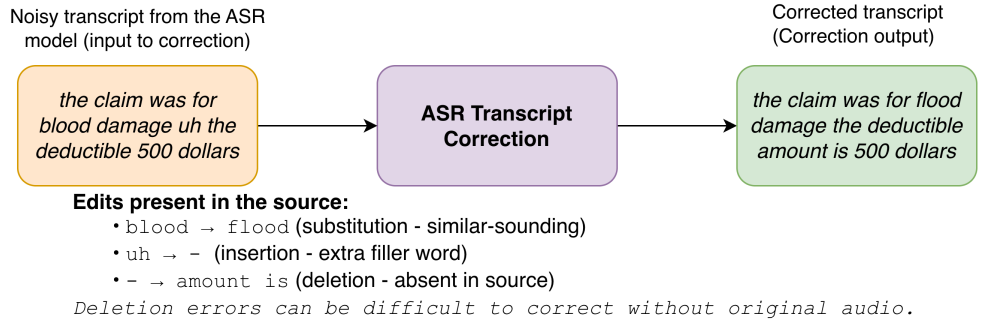

Figure 3.1: The ASR transcript correction task. Given a noisy source transcription (from an ASR model), the objective is to generate a corrected version (what was actually spoken). Corrections can be decomposed into substitution, deletion, and insertion edits.

The corrections can be decomposed into three atomic edit operations - substitution,

deletion, and insertion (Section 2.3.3). As shown in Figure 3.1, correcting deletion errors (where the edit is missing in the noisy source) can be significantly difficult without the original audio since the data was lost during transcription and has to be inferred from context alone.

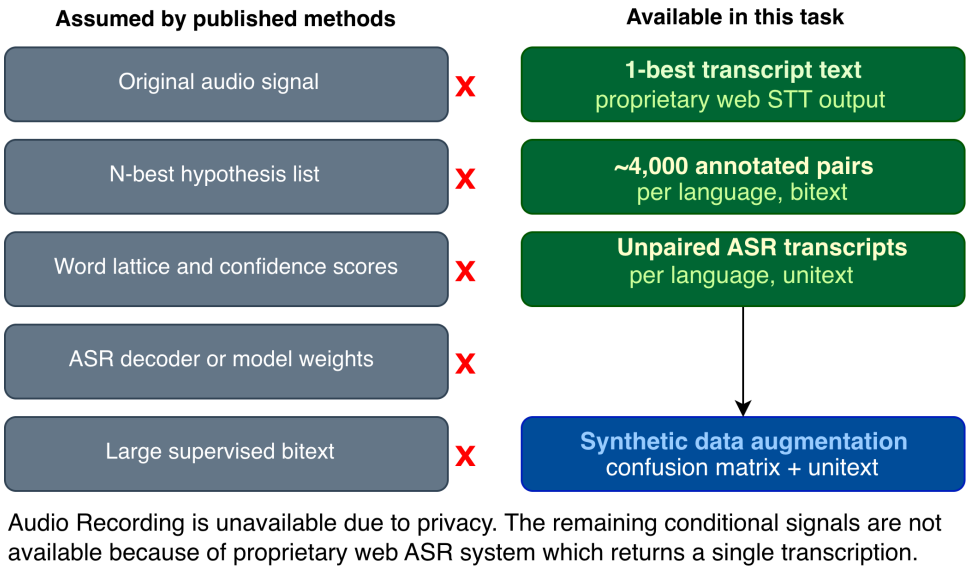

Figure 3.2: Comparing the task data used by various ASR correction methods against those available in this project. The low-resource, text-only setting of the task means that the audio recordings and other conditional data signals are unavailable. The unannotated ASR transcript is the only resource available in some quantity, which is used for different synthetic data augmentation methods.

I focus on a low-resource, text-only data setting for this task. The main data for this project, provided by an industry partner, consists of anonymized customer support call transcripts in English and French, generated by a proprietary web-based Speech-to-Text (STT) service. The dataset consists of a limited number of source ASR and the corresponding manually annotated target transcript for training (bitext - $\sim$ 4000 pairs per language), alongside a large number of unpaired source ASR transcripts (unitext). To make sure the work is reproducible beyond the proprietary industry dataset, I replicate the low-resource setting on two public ASR datasets (Section 3.3.2). This setting is realistic and highly relevant across many industries, since collecting target annotations is expensive and customer data privacy regulations [67] can make it difficult to use the original audio signal in any correction solution. Despite this importance, it has received limited attention in the

ASR Transcription correction literature (Section 2.4), where most solutions have access to large and rich conditional data that is not available for my task. Figure 3.2 summarizes the difference between various ASR solution methods and the low-resource setting: the audio call recordings are unavailable due to privacy, and the remaining conditional signals (multiple hypotheses, confidence scores) are unavailable since the proprietary web ASR system returns a single transcription.

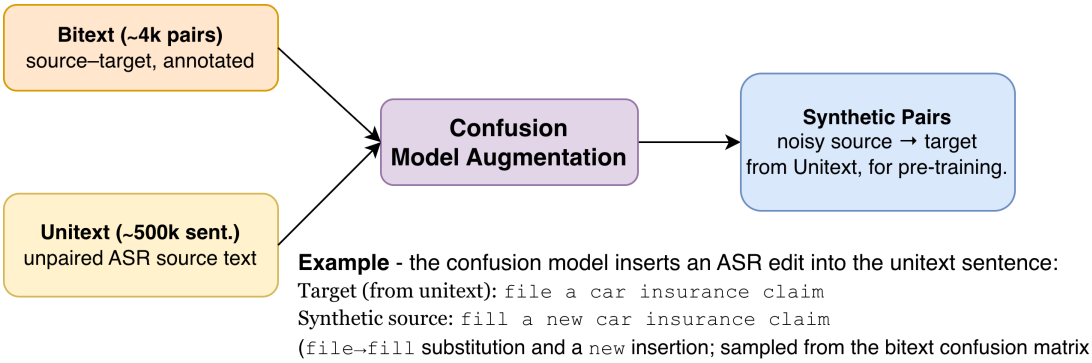

Figure 3.3: The synthetic data augmentation approach proposed in this chapter (Section 3.6). The small bitext and larger ASR source transcripts (unitext) is used to generate synthetic noisy source-target pairs using a confusion model. The correction model is pre-trained on the synthetic pairs before fine-tuning on the bitext. Depending on the type of bitext edit inserted, the unitext can be used as the source (add correction edits) or target (add corruption edits) for the synthetic pair generation.

Since it is relatively cheap to obtain a large amount of unpaired ASR transcripts (compared to annotated bitext pairs), the main contribution of this chapter is a data augmentation-based solution. The solution makes use of the small bitext and the larger unitext transcripts to generate a large synthetic source-target pair dataset that replicates the edits and errors of the ASR system (Figure 3.3). A confusion model is constructed from the bitext alignment that represents the ASR system transcription errors at the phrase level. This confusion model is used to probabilistically partition unitext sentences into fragments and sample possible edits/errors for each fragment, transforming the unpaired transcripts into a synthetic training pair dataset. Depending on the type of confusion matrix edit, the unitext can be used as the synthetic source (add correction edits) or target (add corruption edits). The post-ASR correction solution - a BART-based Seq2Seq model (Section 2.2.3) uses the synthetic dataset for pre-training and then fine-tunes on the bitext (Section 3.7.1).

Apart from the Seq2Seq synthetic data solution, I also explore LLM-based correction as a complementary method. This includes a few-shot approach using in-context examples

(Section 3.7.2), and a confusion model-generated synthetic n-best ASR Transcription correction (Section 3.7.3). For both solutions, the baseline approach only uses the limited bitext and is compared with the synthetic data-augmented methods. To evaluate correction performance beyond WER and better understand the task, I also conduct a human evaluation on the industry dataset to categorize the different transcription error types (Section 3.4), which allows for a better analysis of each solution at the error type level.

3.2 Motivation and Contributions

The ASR transcription correction approaches explored in this chapter follow from the constraints of the task. Most of the conditional ASR data that is utilized by recent ASR correction methods - audio features, multiple scored hypotheses, ASR model access, and a large training corpus is not available here (Figure 3.2; Section 2.4). Of the five non-intrusive ASR correction approaches surveyed by Peyghan et al. [216], only output-based *correction* makes use of the predicted text alone, making it the feasible approach for my task.

Focusing on the correction approaches, my constraint is data - only ∼4,000 annotated pairs per language (bitext). The resource available in larger quantity is unpaired ASR transcription output (unitext), so I extract the ASR system's transcription edits from the bitext and use them to transform the unitext transcripts into synthetic training data. A BART-based Seq2Seq model is well-suited for this task, since its denoising pre-training objective aligns with text correction tasks (Section 2.2.3). It can also be trained and run locally (privacy constraints) on older GPU hardware, making it suitable for the low-latency inference required in the industry production environment. Apart from this, multiple recent low-resource ASR correction works show that task-specific smaller models perform better than much larger general-purpose LLMs on the minimal-edit correction task [17, 145]. I also evaluate LLM-based correction as an in-context prompting baseline and in a setting that uses data augmentation (synthetic n-best candidates), which performs better than the prompting baseline in most settings.

3.2.1 Contributions

I present the following contributions in this chapter:

- An alignment-based sentence extraction and normalization pipeline that transforms industry call transcripts into supervised bitext pairs for training and evaluation.

- Adaptation and evaluation of a confusion-matrix based synthetic data augmentation method [257] for a text-only low-resource ASR Seq2Seq correction model, evaluated on the industry and public ASR datasets for English and French.
- ASR model correction score (WER) evaluated across WER Corrected and WER Introduced (Section 3.5) providing a detailed insight into model performance.
- A synthetic ASR n-best candidate LLM correction method, generated by varying the partitioning threshold of the confusion model, that outperforms the in-context LLM prompting baseline in most settings.
- A human annotation on the industry dataset to label different transcription error types and an error analysis of the model output.

3.2.2 Chapter Structure

Section 3.3 introduces the industry and public datasets, the alignment-based sentence extraction method, and the normalization pipeline. Section 3.4 presents the human annotation study and error analysis, and Section 3.5 defines the evaluation metrics. Section 3.6 describes the confusion-matrix synthetic data augmentation method and the configurations, with Section 3.7 describing the different correction methods built using it. Experimental Setup and the results are presented in Section 3.8, followed by the conclusion and some future directions in Section 3.9.

3.3 Dataset Description

In this project, I use data from two sources: a proprietary task dataset provided by my industry collaborator and 2 publicly available ASR transcript datasets to ensure reproducibility. Both are described in this section, along with the data normalization steps and the statistics.

3.3.1 Industry Dataset

The industry dataset consists of anonymized customer support call transcripts in both English and French, generated by a proprietary web-based Speech-to-Text (STT) service. The calls are real-world multi-speaker conversations between customers and support agents,

discussing various topics like claim data, policy information and account details. All personally identifiable information (PII) was tagged and removed from the raw text transcripts before starting this project.

The supervised low-resource training dataset - source ASR transcripts paired with manually annotated reference transcriptions (bitext), comprises 62 English calls, split into 50 training calls and 12 test calls, generating approximately 4,900 sentence pairs in total. The French data consists of 84 calls, split into 68 training and 16 test calls, yielding approximately 4,900 sentence pairs as well. In addition to the supervised bitext data, I also have roughly 50,000 unpaired source ASR call transcripts (unitext) available for each language that are used for synthetic data augmentation (Section 3.6).

Unlike most publicly available ASR datasets, which provide transcriptions for shorter chunked audio samples [248, 36], the industry dataset consists of complete source and reference transcripts for the entire audio call. After pre-processing and normalizing the transcripts (described below in this Section 3.3.1), I perform alignment-based transcript splitting to extract the supervised bitext. Since the calls consist of overlapping multi-speaker conversations and have no audio timestamps, it is difficult to determine fixed sentence boundaries. To overcome this issue, I use the JiWER[1] package to calculate the word-level alignment between each source-target transcript pair, obtaining a sequence of edit operations (EQUAL, SUBSTITUTE, DELETE, or INSERT) and their word index spans. A left and right boundary span is selected from the contiguous EQUAL spans (based on the desired sentence length bounds - 30 and 40 words), and the source-target substring between the two equal-alignment boundaries is extracted as a single sentence pair (bitext). The right boundary span of an extracted pair is chosen as the left boundary span for the next, ensuring complete coverage of all edits from the whole transcript. The procedure for transcript alignment and sentence extraction from the Industry Dataset is described in Algorithm 1 along with an example illustration in Figure 3.4. The statistics for the industry dataset after sentence extraction are reported in Table 3.1. I refer to this dataset as the low-resource ASR transcription correction dataset.

Apart from the low-resource supervised training dataset (bitext), I also have the ASR transcription source generated from $\sim 50,000$ Calls for both French and English. These transcripts, along with the bitext data, are used to generate synthetic pairs for ASR transcription correction model training (Section 3.6).

The industry dataset has two levels of granularity - utterance and call level. The training bitext is extracted at the utterance (sentence) level using Algorithm 1, while the confusion matrix transformations (Section 3.6.4) and the test set use the complete call transcripts.

[1] https://jitsi.github.io/jiwer/

Algorithm 1 Alignment-Based Bitext Sentence Extraction

Input: Source transcript H (ASR hypothesis); Target transcript R (reference); Lower bound $L_B = 30$; Upper bound $U_B = 40$
Output: List of sentence pairs $\mathcal{P} = \{(h_k, r_k)\}_{k=1}^{K}$
{Step 1: Compute word-level alignment between source and target}
$\mathcal{A} \leftarrow$ JIWER.PROCESSWORDS(R, H)
{Step 2: Extract all "equal" alignment span index information.}
$\mathcal{E} \leftarrow \{a \in \mathcal{A} \mid a.\text{type} = \text{EQUAL}\}$
{r = reference span, h = hypothesis span, start/end = index value}
{Each span $e \in \mathcal{E}$ stores $(r_{\text{start}}, r_{\text{end}}, h_{\text{start}}, h_{\text{end}})$}
{Step 3: Greedily select boundary span pairs}
$\mathcal{P} \leftarrow \emptyset$
$i \leftarrow 0$
while $i < |\mathcal{E}|$ **do**
  $e_{\text{start}} \leftarrow \mathcal{E}[i]$ // Left boundary span
  $\ell_{\text{target}} \leftarrow$ RANDOM.RANDINT(L_B, U_B) // Sample target chunk length
  {Advance right boundary until chunk length is satisfied}
  for $j = i + 1$ **to** $|\mathcal{E}| - 1$ **do**
    $e_{\text{end}} \leftarrow \mathcal{E}[j]$
    $\ell \leftarrow e_{\text{end}}.r_{\text{end}} - e_{\text{start}}.r_{\text{start}}$ // Span length in reference words
    if $\ell \geq \ell_{\text{target}}$ **then**
      break
    end if
  end for
  {Step 4: Extract aligned source and target substrings}
  $r_k \leftarrow R\big[e_{\text{start}}.r_{\text{start}} : e_{\text{end}}.r_{\text{end}}\big]$
  $h_k \leftarrow H\big[e_{\text{start}}.h_{\text{start}} : e_{\text{end}}.h_{\text{end}}\big]$
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{(h_k, r_k)\}$
  {Step 5: Advance left anchor to right anchor}
  $i \leftarrow j$
end while

return $\mathcal{P}$

During inference, each test call ASR transcript is split into $\sim$ 35-word chunks, and a lookup dictionary maps each chunk (utterance id) to its call id. The corrected chunks are then stitched back together for each call, and the test WER is computed at the call level against

Edit type	Reference (target)	ASR hypothesis (source)
EQUAL ①	the policy covers	the policy covers
SUBSTITUTION	flood damage	blood damage
EQUAL	and the deductible	and the deductible
DELETION	amount is	—
EQUAL ②	five hundred dollars	five hundred dollars
SUBSTITUTION	per calendar year	per calendar ear
INSERTION	—	uh
EQUAL	before we reimburse	before we reimburse
SUBSTITUTION	any repair costs	in a repair costs
EQUAL ③	you will need to	you will need to

Extracted pair 1 (boundary ①→②, ∼15 words):

Ref	*the policy covers flood damage and the deductible amount is five hundred dollars*
Src	*the policy covers blood damage and the deductible five hundred dollars*

Extracted pair 2 (boundary ②→③, ∼16 words):

Ref	*five hundred dollars per calendar year before we reimburse any repair costs you will need to*
Src	*five hundred dollars per calendar ear uh before we reimburse in a repair costs you will need to*

Figure 3.4: Example of Alignment-based sentence extraction from a pseudo call source-target transcription. The grey rows mark EQUAL alignment spans that were selected as the left and right boundary spans (labelled ①, ②, ③). All words between two boundaries are extracted as a single source-target sentence pair. For this example, sentence word length was sampled from $[12, 17]$ by the algorithm. The right boundary span of each extracted pair becomes the left boundary of the next, ensuring complete and non-overlapping coverage of all the edits in the transcript. The error words corresponding to each edit type are underlined.

the complete reference transcript. The development sets (200 English and 100 French sentences) are kept at the utterance level and are used for the human annotation study (Section 3.4). The public ASR datasets are only available as utterance splits of 30-35 words (Section 3.3.2), and are evaluated at the utterance level.

Lang.	Split	# Calls	# Sentences	Mean Words/Sent.	WER (%)
English	Train	50	4,060	33.9	10.68
	Test	12	638	35.2	10.63
	Dev	—	200	35.2	16.80
French	Train	68	3,825	37.6	31.14
	Test	16	976	35.8	28.71
	Dev	—	100	34.9	31.33

Table 3.1: The statistics for the industry dataset after normalization, alignment and sentence extraction from the call transcripts. The Development (Dev) set sentences are extracted from the training data before sentence extraction. The Mean Word is calculated for the source sentences in the bitext. WER is calculated against the manually annotated reference transcripts.

Data Pre-Processing and Normalization

Due to the noisy environment of the customer support audio calls, the multi-person overlapping conversation and the PII anonymization, cleaning the Industry dataset transcripts is essential. Based on manual inspection of recurring error patterns in the source-target transcript alignment, a consistent normalization stage is applied to both the ASR source transcripts and the reference target transcripts (unitext + bitext) before using the dataset for training and evaluation. The primary goal being to reduce variations between the source and target that might not be true transcription errors. The following steps are applied sequentially to the Industry Dataset:

1. Speaker Switching Removal
The source and target transcript, for each call, is initially provided as sequences in which both the ASR model and the human annotator attempt to segment and construct utterances based on speaker switching. However, due to the overlapping, multi-speaker nature of customer support calls, speaker switches are frequently ignored or mistaken by the ASR model, causing poor utterance-level alignment between the source and target. To overcome this issue, I discard all utterance tagging and splits, combining them and generating a single source and target transcript for each call. Sentence-level segmentation is performed (after normalization) on the concatenated pair using the alignment-based extraction method described in Algorithm 1.

2. Tag and Punctuation Removal
The source and target transcripts contain bracket-tagged annotations that were removed

in this step. The ASR source transcripts contain PII word spans tagged and replaced with their type (`[EMAIL_1]`, `[ADDRESS_1]`, etc.) by a proprietary anonymization service. Similarly, all PII elements were tagged and removed from the target transcripts by the human annotators. The difference in the naming type convention and location of the tags, between source and target, added extra alignment-based errors. The human annotators also added extra tags in the target transcripts for non-speech acoustic occurrences, like `[silence]`, `[laugh]`, and `[unk]` (listed in Step 4. below). To reduce tagging-based errors, all the tagged terms were removed from both source and target transcripts.

After the tag removal, all punctuation was stripped, and the transcripts were converted to lowercase to get a cleaner and better aligned transcription. I retain the dollar sign (`$`) and the percent symbol (`%`) character, which is important for my dataset and is handled explicitly in normalization Step 6 below.

3. Numerical Terms Normalization

The ASR system and human annotators can transcribe numerical entities in speech differently. For instance, the ASR output may produce *twenty five* while the reference contains *25*, or vice versa. To resolve this, all numerical terms present as alphabetic words were converted to their digit representation using the `text2num` Python package[2]. Both the source and target transcripts were normalized to ensure a consistent digit-based representation of all numerical terms.

4. Filler Word Removal

Often transcriptions can include multiple occurrences of filler words (FW) and disfluencies due to background noise, pauses and interruptions in the audio call recording. The ASR model and the human annotators can't handle these issues consistently, adding to alignment inconsistency and increasing the WER. To remove the FW terms, a manual inspection stage was performed. A frequency-dictionary mapping of all words (with fewer than five letters) present in all transcriptions was created and then checked. This provided me with the most commonly occurring FW: *uh*, *um*, *uhm*, and *mhm*. The bitext alignment showed that these FW were mostly present in the ASR Transcription and were aligned with the non-speech acoustic-tagged terms in the reference (Step 2 above). A vocabulary of all FW terms was created, and tokens matching any vocabulary item were removed from both source and target transcriptions. The established vocabulary $\mathcal{V}_{\text{FW}}$ is presented in Table 3.2.

Even though I added the non-speech tags in the vocabulary $\mathcal{V}_{\text{FW}}$ for completeness, all those terms were already removed in Step 2 of normalization. The partial normalization

[2] https://pypi.org/project/text2num/

Table 3.2: Filler Word and Acoustic Tag Vocabulary ($\mathcal{V}_{\text{FW}}$)

Disfluencies	**Acoustic / Non-Speech Tags**		
uh	[unk]	[pause]	[throat]
uhm	[silence]	[noise]	[mumble]
um	[laugh]	[cough]	
mhm			

in Step 2 increased alignment inconsistency between the source and target transcriptions. This makes removing the disfluencies found in this step from both transcriptions an important step to increase transcription consistency.

5. Repeated Word Removal.
A common source of transcription token alignment mismatch is the presence of word repetitions, a common occurrence in conversational speech (e.g. *I I I think*, *okay okay okay*). These can also occur due to ASR Model transcription hallucination. This can cause an inconsistency in the number of word repetitions between aligned transcripts (e.g., `okay okay` → `okay`, or `I I I` → `I I`). Since the repeated words can't be consistently aligned, I remove all repeated occurrences of any word from both transcriptions, reducing them to a single occurrence.

6. Currency, Units, and Symbol Normalization.
In the final normalization stage I fix the currency term, percentages, and measurement unit inconsistencies. Specifically:
• dollar signs and amounts are expanded to their word form (`$50` → *50 dollars*, `$` → *dollars*).
• percentage symbols and its word variations are normalized (`%` → *percentage*, *percentages* → *percentage*).
• distance unit abbreviation is expanded to its word form(`km` → *kilometers*, *kilometer* → *kilometers*).
These substitutions are applied using regular expression rules and ensure that similar term representations are consistent between the source and target transcription.

3.3.2 Public Datasets

Since the industry dataset for this project is proprietary and cannot be publicly released, all methods and experiments are additionally evaluated on publicly available ASR transcript datasets. This ensures that the work is reproducible and can be readily extended by the

research community without requiring access to the industry data. For English, I use the RED-ACE dataset [79] and for French, I use the French subset of the Multilingual LibriSpeech (MLS) dataset [225].

RED-ACE (English)

The RED-ACE dataset [79] was introduced for the task of ASR Error Detection and Correction and is built on top of the English LibriSpeech corpus[3] [210]. The ASR output was generated by transcribing the LibriSpeech audio using the Google Cloud Speech-to-Text API. For my experiments, I use only the source ASR hypothesis and the reference transcriptions, discarding the confidence scores and error labels since this task focuses on text-only ASR transcription correction.

The RED-ACE dataset follows the standard LibriSpeech train/dev/test splits. Each split of the LibriSpeech dataset is further divided into a "clean" and "other" subset. The $\sim$ 1000 hours of LibriSpeech dataset consists of audiobook recordings from the LibriVox Project[4] aligned with the corresponding book text. As part of the dataset collection project, Panayotov et al. train an acoustic ASR model on the *Wall Street Journal (WSJ) si-84* audio data [213] and use it to transcribe the LibriSpeech audio data. The Word Error Rate (WER) was calculated between the WSJ Model transcription and the reference text. Based on the WER, the LibriSpeech dataset for all splits was divided into "clean" and "other" splits, where the clean data split has a lower WER, denoting a cleaner audio recording for the corresponding reference text.

The English LibriSpeech dataset consists of the following training data subsets - *train-clean100*, *train-clean-360* and *train-other-500*, with roughly 100, 360 and 500 hours of audio, respectively. Apart from the training data subset, $\sim$ 5 hours of audio data is provided for each dev/test data split (for both clean and other subsets). Using a multi-stage alignment algorithm, Panayotov et al. align the full audio recording with the reference text and split it into up to $\sim$ 35 seconds of audio-text example pairs. For instance, the 363.6 hours of audio recording in the *train-clean-360* subset was split into 104,014 training examples. For the RED-ACE dataset, Gekhman et al. [79] select the *train-clean-360*, *train-other-500*, and all the dev/test data splits and transcribe it using the Google Cloud Speech-to-Text models. The authors mention that there was $\sim$ 2% data loss per split while transcribing. As a result, the RED-ACE transcription dataset is a subset of the full LibriSpeech dataset corpus.

[3]Dataset available at: https://huggingface.co/datasets/google/red_ace_asr_error_detection_and_correction

[4]https://librivox.org/

For this project, I combine the RED-ACE *train-clean-360* and *train-other-500* transcriptions for my low-resource training dataset and the unitext source-transcript for training. The data normalization and splitting process, along with the final data statistics, is provided below (Section 3.3.2).

Multilingual LibriSpeech (French)

The Multilingual LibriSpeech (MLS) dataset [225] is a large multilingual audio corpus generated from LibriVox audiobook recordings. It contains audio-and-transcription pairs for eight languages: English, German, Dutch, French, Spanish, Italian, Portuguese, and Polish. I only utilize the MLS-French data subset for this project[5]. The MLS-French dataset consists of $\sim$ 1300 hours of audio recordings split into Train/Dev/Test, with the Dev and Test data splits containing $\sim$ 10 Hours of data each. The chunked audio corresponds to $263,055$ examples in total.

Unlike the RED-ACE (English) ASR dataset, the MLS dataset only contains the chunked audio recordings (10-20 seconds) with the corresponding reference transcription. The dataset does not contain ASR predictions that can be directly utilized as training data for an ASR Transcription correction model. To obtain the ASR source transcripts for my project, I manually perform ASR inference on the entire MLS-French dataset using a strong public ASR LLM. I choose Whisper-Large-v3-Turbo [234], a fast, multilingual, and open-license ASR model. The Whisper ASR predictions are combined with the MLS-French reference transcripts to form the French language source-target bitext used for training and evaluation. The data normalization and splitting process, along with the final data statistics, is provided below (Section 3.3.2).

Public Dataset Normalization and Statistics

Unlike the Industry Dataset, which requires a multi-step data normalization process, both the public ASR datasets consist of single-speaker audiobook recordings, which are significantly cleaner than the customer support call recordings. As a result, I only perform a minimal amount of pre-processing normalization on the public dataset. For the RED-ACE (English) dataset, I only convert the source and target transcripts to lowercase. For the MLS (French) dataset, apart from the lowercase transformation, an added step of punctuation removal is performed. The punctuation characters: `,!?".*` were removed from both source and target transcriptions. Finally, for both datasets, I reduce consecutive spaces to a

[5]Dataset available at: https://huggingface.co/datasets/facebook/multilingual_librispeech

single space. I use the training data selection approach from Grammatical Error Correction research (Section 2.5) by filtering for training examples that contain errors (source $\neq$ target). Because the RED-ACE training set (#Examples = 503,136) is substantially larger than the MLS training data (#Examples = 258,213), I only extract the erroneous subset from RED-ACE while keeping the full MLS dataset. The complete corpus-level statistics and baseline Word Error Rates (WER) for the normalized public ASR datasets across all data splits are provided in Table 3.3.

Dataset	**Split**	**# Sentences**	**Mean Words/Sent.**	**WER (%)**
RED-ACE (English)	Train	435,311	34.04	14.0
	Dev (Clean)	5,400	20.08	8.94
	Dev (Other)	5,673	17.52	18.95
	Test (Clean)	5,235	20.02	9.04
	Test (Other)	5,864	17.55	19.30
MLS (French)	Train	258,213	36.95	6.18
	Dev	2,416	35.3	7.76
	Test	2,426	38.47	7.05

Table 3.3: The complete corpus-level statistics and Word Error Rates (WER) for the public ASR datasets, after normalization, for all data splits.

To replicate and utilize the Public ASR dataset in a low-resource setting, similar to the Industry Dataset (English - $4,060$ examples, French - $3,825$ examples), I sample a subset of *4,500 examples* from the Public ASR datasets for bitext training. The remaining *source* text examples were used as the unitext for Synthetic Data Augmentation. The size and WER of the sampled low-resource training dataset are presented in Table 3.4.

3.3.3 Edit Type Distribution

The three atomic ASR edit errors can differ in terms of how easily they can be solved in a text-only setting, which makes studying their error distribution across the dataset insightful. Figure 3.5 breaks down the training set WER of each dataset into its substitution, deletion, and insertion edit components. The complete breakdown for all ASR data splits is presented in Appendix C.1.

The count for each ASR edit type is considered in the regular ASR direction, relative to the target transcript (Section 2.3.3): deletions are reference words the ASR model

Dataset	Training Data	# Sentences	Mean Words/Sent.	WER (%)
RED-ACE	LR Bitext	4,500	34.03	13.89
(English)	Source Unitext	430,811	34.04	—
MLS	LR Bitext	4,500	36.99	6.14
(French)	Source Unitext	253,713	36.95	—

Table 3.4: Corpus-level statistics for the sampled Low-Resource (LR) training data (bitext) and the remaining unannotated source text used for synthetic data augmentation (unitext). $4,500$ examples were sampled for the bitext to replicate the low-resource setting of the industry ASR datasets.

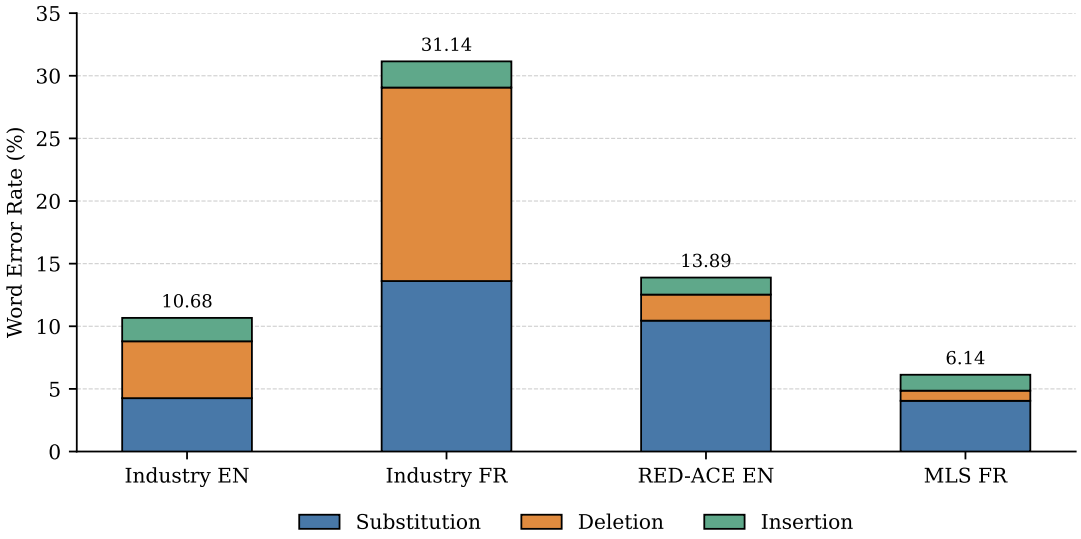

Figure 3.5: The breakdown of the low-resource training data WER into substitution, deletion, and insertion edits for each dataset. The edits are counted in the regular ASR direction: deletions are reference words the ASR model ignored, insertions are extra words it added. Each edit segment in the bar is that edit type's contribution to the WER, the full bar height denoting that dataset's WER. The industry ASR dataset is deletion-dominant, while the public ASR dataset is primarily substitution edits. The public dataset corresponds to the sampled low-resource bitext (Table 3.4) used in this work.

ignored, insertions are extra words it added. The correction model edits operate in the opposite direction. ASR deletions are words the model has to *insert* in the source, and

ASR insertions are words it has to *delete*. I use this ASR convention when describing the dataset edits and the correction model's edit explanation.

The two ASR datasets (Industry and Public) have different error distributions as shown in Figure 3.5. The industry customer-support call transcripts dataset is dominated by deletion errors (words omitted by the ASR model in the transcription) which have a greater WER than substitution errors for both languages. This can be explained by the noisy, overlapping multi-speaker call recordings, where fragments of the speech are ignored by the ASR model. The public ASR dataset error distribution shows that substitutions outnumber deletion errors significantly (~ 5 times) for both RED-ACE and MLS, reflecting the clean single-speaker nature of the audiobook recordings, where the ASR model is more likely to incorrectly transcribe words rather than ignoring them.

This difference is important for a text-only ASR Transcription correction system. Substitutions and incorrectly inserted words may contain linguistic features (example - similar pronunciation, word-context inconsistency, etc.) in the source that a correction model can learn and train on. However, deleted words have to be corrected using only the surrounding context. Removing deletion edits from the training pair (bitext) lowers the bitext WER from 10.68% $\rightarrow$ 6.40% for Industry English and from 31.14% $\rightarrow$ 18.53% for Industry French ($\sim 40\%$ reduction for both). For public ASR datasets, I see a reduction from 13.89% $\rightarrow$ 12.05% for RED-ACE and 6.14% $\rightarrow$ 5.36% for MLS ($\sim 13\%$ reduction). These results indicate that a significant amount of the industry dataset error contains missing content from the source, which is the low-resource ASR Transcription correction setting I focus on in this chapter.

3.4 Human Evaluation and Error Analysis

3.4.1 Error Categorization

To understand the nature and distribution of ASR transcription errors present in the industry dataset, I conduct a human evaluation on a development set of 200 English sentences and 100 French sentences, manually transcribed by the industry linguists. The categories of correction edits identified, example description, along with their count and relative error percentages in the development set, are presented in Tables 3.5 (English) and 3.6 (French).

Category	Type and Examples	Count	Relative %
Similar sounding	Up in /upping, this/these	101	18.23
Missing short words	Okay, yeah	82	14.80
Symbols/numbers	6/six, %/percent	53	9.57
Missing long sequences	Possibly when two speakers talk at same time	49	8.84
Other	Hallucinations, incorrect reference	49	8.84
Tag Error	[unk] ->word, [laugh] ->word	49	8.84
Word Variations	Traveling/travelling, all right/alright	38	6.86
Extra Word	Extra word inserted in Source that is not in Target	32	5.78
Slang	Gonna/going to, cause/because	32	5.78
Repeated words	I I I, okay okay okay	17	3.07
Hallucination	Erroneous sentences or phrases inserted	14	2.53
Domain words	Industry Specific Words and Product Terms	14	2.53
Questionable Reference	Source seems right and Target is questionable	12	2.17
Anonymization	-	5	0.9
Contraction	Will not/won't, you will/you'll	4	0.72
Spelling out	B beer U unicorn C car K key L low E exit Y you	3	0.54
Total		554	**100**

Table 3.5: The different categories of edits between the ASR transcription source and the annotated target correction for the *Industry English dataset (200 sentences)* obtained during the human evaluation stage. The table presents the error category, examples, and relative percentage distribution.

3.4.2 Human Annotation Baseline

As a Human upper bound on my task of ASR transcript correction, I asked the Industry linguists to manually correct the development set source sentences. Table 3.7 reports the resulting WER, Error Corrected%, and Error Introduced% for both languages. The relatively high Error Introduced% even for human annotators reflects the inherent difficulty and subjectivity of post-editing noisy call transcripts: correction of one error can inadvertently introduce mismatches elsewhere, particularly when source disfluencies are present.

Category	**Type and Examples**	**Count**	**Relative %**
Similar sounding	savoir / lavoir	136	28.87
Missing short words	ouais, okay, là	118	25.05
Other	completely different transcription	48	10.19
Filler word	eh, euh	34	7.22
Missing long sequences	overlapping speakers	32	6.79
Grammar	vous habitez / vous habiter	24	5.10
Spacing	y a / ya	13	2.76
Extra word	word improves readability	13	2.76
Contraction	tsais / tu sais	11	2.34
Symbols/numbers	7 / sept, % / percent	10	2.12
Repeated words	oui, oui, oui	8	1.70
Slang	tsais / tu sais	8	1.70
Hallucination	Extra phrases added	5	1.06
Language variation	alternate spellings	4	0.85
Domain word	Industry Specific Terms	3	0.64
English words	English expressions in French	2	0.42
Tag error	[unk] $\rightarrow$ word	1	0.21
Questionable reference	hypothesis seems correct	1	0.21
Total		471	**100.00**

Table 3.6: The different categories of edits between the ASR transcription source and the annotated target correction for the *Industry French dataset (100 sentences)* obtained during the human evaluation stage. The table presents the error category, examples, and relative percentage distribution.

Dataset	**WER**	**Error Corrected%**
English Dev (ASR)	16.80	—
English Dev (Manual)	24.44	8.63%
French Dev (ASR)	31.33	—
French Dev (Manual)	37.02	5.84%

Table 3.7: Human annotation correction results on the development sets for English and French. The manual annotation WER is computed against the reference transcripts and reflects the difficulty of the task.

3.5 Evaluation Metrics

Word Error Rate (WER) is the standard ASR evaluation metric, computed as the normalized edit distance between a predicted transcript and the reference:

$$\text{WER} = \frac{S + D + I}{N}, \tag{3.1}$$

where S, D, and I are the number of substitution, deletion, and insertion edits, respectively, and N is the total number of words in the reference.

Word Error Rate (WER) weighs all edit operations equally. For instance, a substitution error for a domain-specific term has the same impact as a filler word. As a result, a model that makes many edits of which half are wrong results in the same net WER change as one that makes few to no edits. Peyghan et al. [216] present in their survey that the ASR Transcript correction task is susceptible to over-editing, where the model introduces new errors while attempting to correct existing ones. This makes the net WER an insufficient evaluation signal on its own. To address this issue, I additionally present the **WER Corrected** and **WER Introduced** metrics, which decompose the net WER change into how many ASR errors the model fixes, and how much new error it introduces. The transcript alignment and the WER calculation for the additional two metrics are explained below.

3.5.1 Word-Level Alignment via KaldiAlign

Some inconsistencies occur while computing the error-corrected and error-introduced metrics using the JiWER - standard alignment library. The JiWER Alignment[6] edit tagging operates at *phrase level spans*, producing a sequence of edit operations to compare the source and target. As a result, the difference $\text{error}_{\text{corrected}} - \text{error}_{\text{introduced}}$ computed from phrase-level alignments does not match the $\text{WER}_{\text{source}} - \text{WER}_{\text{prediction}}$, since a single Phrase-level edit can contain multiple words and does not match the count of word-level edits.

To resolve this, I use a word-level alignment approach using `kaldialign`[7]. For each transcript triple (source s, target t, prediction p), two word-level alignments are computed:

$$\mathcal{A}_{\text{base}} = \textsc{Align}(t,\, s), \tag{3.2}$$
$$\mathcal{A}_{\text{pred}} = \textsc{Align}(t,\, p), \tag{3.3}$$

[6] https://jitsi.github.io/jiwer/usage/#alignment
[7] https://github.com/pzelasko/kaldialign#alignment

where each alignment is a sequence of (target word, hypothesis word) pairs, with ε (the empty symbol $*$) used to denote word insertions and deletions in the hypothesis relative to the target. Because both alignments are fixed to the same target sequence t, a word-by-word comparison is straightforward: for every position i in the target, the base-aligned source word s_i and the pred-aligned prediction word p_i can be compared against the target word t_i directly.

3.5.2 Metric Definitions

Given the two word-level alignments, WER Corrected and WER Introduced are computed as follows. For each target position i:

- A word is **corrected** if the source was wrong ($s_i^* \neq t_i$) and the prediction is right ($p_i^* = t_i$).
- A word is **introduced** (as a new error) if the source was correct ($s_i^* = t_i$) and the prediction is wrong ($p_i^* \neq t_i$).

Insertion edits — cases where the source or prediction contains an extra word not aligned to any target word — are handled separately. If the source has more insertions than the prediction ($I_{\text{base}} > I_{\text{pred}}$), the difference counts toward words corrected; if the prediction has more insertions than the source ($I_{\text{pred}} > I_{\text{base}}$), the difference counts toward words introduced. The final metrics, normalized by the total number of reference words N, are:

$$\text{WER}_{\text{corrected}} = \frac{\text{words corrected}}{N}, \tag{3.4}$$

$$\text{WER}_{\text{introduced}} = \frac{\text{words introduced}}{N}, \tag{3.5}$$

By construction, this word-level decomposition satisfies the consistency constraint:

$$\text{WER}_{\text{source}} - \text{WER}_{\text{prediction}} = \text{WER}_{\text{corrected}} - \text{WER}_{\text{introduced}}, \tag{3.6}$$

which can't be obtained using phrase-level methods. The $\text{WER}_{\text{corrected}}$ and $\text{WER}_{\text{introduced}}$ metric can better clarify what changes an ASR Transcription Correction approach is making.

3.6 Synthetic Data Augmentation for ASR Transcript Correction

The primary challenge of this project is the small amount of training bitext data available. To address this, I utilize a confusion-matrix-based synthetic data augmentation approach, building on the text-level error simulation framework of Schatzmann, Thomson, and Young [257], which was extended and validated by Fazel-Zarandi et al. [68] and Wang et al. [300] for the task of learning Dialogue Models and Policies. I use the large set of unpaired ASR transcripts (unitext) to create synthetic training data. First, a mapping of likely substitution corrections is derived from the ground truth bitext by extracting edit pairs (e.g., `weekend` → {`weaken`, `we can`, `weekend`}). These substitution mappings are then used to apply probabilistic noise injection into unitext sentences, partitioning each call transcript into sub-spans using the correction patterns observed in the bitext. This procedure generates a large set of synthetic source-target sentence pairs that approximately replicate the noise characteristics of the original ASR errors, providing pre-training signal for the sequence-to-sequence correction model.

3.6.1 N-Gram Confusion Matrix Construction

The confusion matrix is the main data structure used for the synthetic data augmentation (SDA) method. Following Schatzmann, Thomson, and Young [257], ASR confusions are modelled as fragment-to-fragment "translations": a source utterance $w = w_1^S$ containing S words is segmented into N contiguous fragments f_1^N, and each fragment f_i is independently mapped to its target aligned fragment $\tilde{f}_i$, allowing all substitution, insertion, and deletion ASR errors to be included in the matrix (since aligned fragments can have different lengths, including zero). The confusion matrix C is a lookup table that contains, for every source fragment s present in the training data (bitext) - $\mathcal{D}$, the set of target fragments t transformations aligned to s in the bitext:

$$C(s,t) = \text{frequency of } (s \to t) \text{ alignment in } \mathcal{D}. \tag{3.7}$$

I refer to s and t as the input and output direction of the transformation being stored in the matrix rather than to a fixed bitext set (supervised source or target). As described below in this section, the bitext alignment can be observed in both directions (source→target and target→source), so the s, t values depend on the type of matrix being constructed. Additionally, the frequency of mapping $C(s,s)$ is also saved (when the source fragment

s is correctly transcribed as itself). The confusion matrix is used to provide a confusion distribution over all possible transformations for each input fragment, including the no-error case.

To populate C, each hypothesis–reference pair $(h_i, r_i) \in \mathcal{D}$ (bitext) is aligned at the word level using `kaldialign`[8], which calculates the minimum Levenshtein edit distance alignment between the two sequences. This generates a mapping of contiguous word spans in h_i to contiguous word spans in r_i, from which all fragment pairs (s, t) with $|s| \leq M$ are extracted and their counts incremented in C. I set the maximum n-gram order to $M = 6$, so the matrix contains confusions for fragments of one to six words. This upper bound (which can be varied) keeps the matrix tractable and allows capturing longer fragment possibilities (which is needed for the partitioning algorithm - Section 3.6.2 below). Table 3.8 illustrates the mapped alignment for a single sentence pair.

Hypothesis (source): `fill a new car insurance claim`
Reference (target): `file a car insurance claim`

Levenshtein alignment:

`fill` ↔ `file`	(substitution)
`a` ↔ `a`	(match)
`new` ↔ ε	(deletion in reference)
`car insurance claim` ↔ `car insurance claim`	(match)

Table 3.8: Example Levenshtein alignment between an ASR hypothesis and its reference transcript. Each aligned fragment pair increments the corresponding entry in the confusion matrix C.

Since the bitext alignment is symmetric, two confusion matrices can be constructed from the same data, depending on which sequence is considered as source/hypothesis. The **source** → **target** matrix C_{correc} captures the correction transformations, while the **target** → **source** matrix C_{corrup} contains corruption transformations. For example, the substitution edit `fill` → `file` in Table 3.8 is stored in C_{correc} since it is a valid correction the annotator made, while C_{corrup} stores `file` → `fill` as an error the ASR system made. Both matrices are built and saved, by swapping the bitext s, t input, and the choice of which to sample from during the generation step depends on the SDA *corruption* or *correction* setting (Section 3.6.5).

[8] https://github.com/pzelasko/kaldialign#alignment

3.6.2 Probabilistic Sentence Partitioning

The Synthetic Data Augmentation (SDA) method generates a synthetic sentence pair, which consists of the starting sentence (source) and its transformation, created using the confusion matrix C. The first step of the SDA method is partitioning the source text into fragments. The selected partitioning determines the groups of word which are substituted together into a valid transformation (obtained from C lookup), and is constructed in a single left-to-right iteration over the source sentence using the Schatzmann, Thomson, and Young [257] alignment model.

Let γ_i denote the fragment index assigned to word w_i, and $w_{\text{start}} \dots w_{i-1}$ are the words already assigned to be part of the current fragment. The first word of the source sentence - w_1 is assigned to the first fragment. For every subsequent word, the model decides whether to append it to the current fragment ($\gamma_i = \gamma_{i-1}$) or start a new fragment ($\gamma_i = \gamma_{i-1} + 1$):

$$P(\gamma_i \mid w_i, w_{\text{start}}^{i-1}, \gamma_{i-1}) = \begin{cases} \varphi & \text{if } \gamma_i = \gamma_{i-1} \\ 1 - \varphi & \text{if } \gamma_i = \gamma_{i-1} + 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

where φ denotes the likelihood of word w_i being appended to the current fragment, extending it from $[w_{\text{start}} \dots w_{i-1}]$ to $[w_{\text{start}} \dots w_{i-1} w_i]$:

$$\varphi = \frac{\text{freq}(w_{\text{start}} \dots w_{i-1}\, w_i)}{\text{freq}(w_{\text{start}} \dots w_{i-1})}, \tag{3.9}$$

and freq$(\cdot)$ denotes the frequency of occurrence for a fragment across all the bitext-source sentences, stored in C. Equation 3.9 shows the calculation of φ as the ratio of freq$(\cdot)$ of the longer string $[w_{\text{start}} \dots w_i]$ to its prefix $[w_{\text{start}} \dots w_{i-1}]$. If the longer string is more common in the supervised bitext data, φ obtains a higher value and the string is likely to be kept and transformed as one fragment. If the longer string is rare, then φ is lower, the current fragment is closed, and w_i becomes the start of a new fragment. The maximum n-gram value $M = 6$ (selected in Section 3.6.1) is the upper-bound for the fragment word length, since freq$(\cdot)$ would return zero for bigger spans after C lookup.

In my implementation, the fragment choice in Equation 3.8 is made deterministically (using a threshold θ) rather than by sampling - the fragment is extended when $\varphi \geq \theta$ and closed otherwise. This makes the partitioning reproducible for a given source sentence, so the fragment length selection can be set and evaluated as a controllable hyperparameter (Section 3.6.5). It also allows using different θ values to construct distinct partitions for

the same source sentence, which I use to generate n-best synthetic correction candidates (Section 3.7.3) for the LLM input.

3.6.3 Confusion Sampling and Synthetic Pair Generation

After the source sentence is partitioned into N fragments f_1^N, each fragment is replaced by sampling possible transformations from its confusion distribution in C. This is the second step of the synthetic data augmentation (SDA) method. Schatzmann, Thomson, and Young [257] model the source transformed sentence as a fragment-level product, considering each fragment transformation as conditionally independent of its context (previous/next fragment):

$$P(\tilde{f}_1^N \mid f_1^N) = \prod_{i=1}^{N} P(\tilde{f}_i \mid f_i,\, \tilde{f}_1^{i-1}) \;\approx\; \prod_{i=1}^{N} P(\tilde{f}_i \mid f_i). \tag{3.10}$$

The conditional independence makes the SDA model tractable - replacements for each fragment can be sampled independently from the confusion matrix C,

$$\hat{f}_j \sim P(\cdot \mid f_j) = \frac{C(f_j,\, \cdot)}{\sum_{t'} C(f_j,\, t')}, \tag{3.11}$$

where $\hat{f}_j$ is the transformation selected for fragment f_j, sampled from the distribution obtained by normalizing the row $C(f_j, \cdot)$ by the total occurrence count of f_j across all observed transformations t' in the bitext-source.

the final synthetic sentence is constructed by the concatenation of all sampled fragments, $\hat{h} = \hat{f}_1 \cdot \hat{f}_2 \cdots \hat{f}_N$. Each sampled replacement $\hat{f}_j$ may be the original fragment itself (no error), a different fragment of the same or different length, or an empty string (deletion), depending on valid bitext transformations. Since fragments can transform to spans of different length, including zero, all three ASR edit types are represented in the SDA method (Section 3.6.1).

Schatzmann et al. partially incorporate the fragment dependence ignored in Equation 3.10 by sampling the confusion model multiple times to generate an n-best list of candidates, scoring their quality using the dialogue language model, and sampling one transformation using those scores. I sample a single candidate directly instead, since no domain language model is available for the low-resource bitext transcripts. Multiple

candidates are also generated in this work, but for a different solution and using a different approach - varying the partitioning threshold θ (Section 3.6.2) to produce synthetic n-best pseudo-correction candidates for LLM-based ASR Transcript correction (Section 3.7.3).

Out-of-Vocabulary (OOV) Fragments The confusion matrix only contains n-gram phrases observed in the small bitext, so many unitext words may have no entry in C (OOV words) and no transformations can be obtained for them. Following Fazel-Zarandi et al. [68], I substitute the OOV words with their closest matching fragment (`difflib` sequence similarity) in the confusion matrix vocabulary and sample from its distribution. OOV substitution follows the same WER probability of the bitext.

Since the confusion matrix C stores all error transformations observed from the ASR system used to obtain the noisy transcripts, the edit distribution of the generated synthetic data follows that of the bitext [257, 300].

3.6.4 Application to the ASR Datasets

For the industry dataset, I use the bitext (50 English training calls / 68 French training calls) as the ASR corpus to construct the confusion matrix. The ∼50,000 unpaired call transcripts per language are used as source sentences for the SDA confusion matrix to generate a larger set of synthetic source–target pre-training data. The resulting synthetic dataset is segmented into sentences using the alignment-based extraction procedure (Algorithm 1). This results in ∼ 3–3.5 million synthetic sentence pairs per language, from which a subset of ∼500,000 pairs is sampled for pre-training, to ensure a comparable dataset scale to the public ASR datasets. I also pre-trained the industry correction models on all ∼3 million synthetic pairs, which did not result in a consistent or statistically significant improvement over the ∼500,000 pair subset.

The same setup is applied to both public datasets, apart from the sentence alignment and segmentation. The RED-ACE and MLS datasets contain chunked audio-text examples, so the text transcript is already present at the sentence level. The confusion matrix is built using the sampled low-resource bitext (4,500 pairs), and the remaining source sentences - 430,811 for RED-ACE and 253,713 for MLS (Table 3.4), is used as the unitext pool. Algorithm 2 summarizes the complete synthetic data generation procedure.

A key property of this approach is that it does not require any modification to the correction model architecture or the inference pipeline. The synthetic data is generated offline as a pre-processing step, and the correction model is trained on $\mathcal{D} \cup \mathcal{S}$ exactly as it

Algorithm 2 Confusion-Matrix-Based Synthetic Data Augmentation

Input: Bitext $\mathcal{D} = \{(h_i, r_i)\}_{i=1}^{N}$; unitext $\mathcal{U} = \{u_j\}_{j=1}^{M}$; max n-gram order M_n; target error rate ρ
Output: Synthetic training corpus $\mathcal{S}$
{Phase 1: Build confusion matrix from bitext}
Initialise $C \leftarrow \emptyset$
for all $(h_i, r_i) \in \mathcal{D}$ **do**
  $\mathcal{A}_i \leftarrow \textsc{LevenshteinAlign}(h_i, r_i)$
  for all aligned fragment pair $(s, t) \in \mathcal{A}_i$ **such that** $|s| \leq M_n$ **do**
    $C(s, t) \leftarrow C(s, t) + 1$
  end for
end for
for all fragment $s \in \mathcal{D}$ **do**
  Compute $\text{freq}(s) \leftarrow \sum_t C(s, t)$
end for
{Phase 2: Generate synthetic pairs from unitext}
for all sentence $u_j \in \mathcal{U}$ **do**
  {Probabilistic partitioning (Eq. 3.9)}
  $f_1, \ldots, f_K \leftarrow \textsc{Partition}(u_j, \text{freq})$
  {Sample replacement for each fragment (Eq. 3.11)}
  for $k = 1$ **to** K **do**
    if $\textsc{random.uniform}(0, 1) < \rho$ **then**
      $\hat{f}_k \sim P(\cdot \mid f_k; C) \;\mid\; \hat{f}_k \neq f_k$
    else
      $\hat{f}_k \leftarrow f_k$
    end if
  end for
  $\hat{h}_j \leftarrow \hat{f}_1 \cdot \hat{f}_2 \cdots \hat{f}_K$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\hat{h}_j, u_j)\}$
end for

return $\mathcal{S}$

would be on any other parallel corpus. This makes the method modular and applicable to any sequence-to-sequence correction model.

3.6.5 Synthetic Data Generation Configuration

I consider several configuration choices to modify the characteristics of the data produced by the synthetic data augmentation (SDA) method from the unitext described above. I treat these as hyperparameter choices and select the best setting based on experimental evaluation (Section 3.8.2). The four settings are defined below.

Generation Direction: Corruption or Correction Two confusion matrices are constructed from the aligned bitext (Section 3.6.1), and the choice of which to sample from determines the type of synthetic data generated. In the **corruption** setting, the target $\rightarrow$ source matrix is used to introduce error transformations into the unitext sentence, treating unitext as target and the constructed sentence as the synthetic source. For the **correction** setting, the source $\rightarrow$ target matrix is used to sample correction edits to construct a synthetic target from the unitext sentence (source).

Synthetic Word Error Rate The original Confusion Matrix SDA approach from Schatzmann, Thomson, and Young [257] samples a replacement for each partitioned fragment using its confusion distribution (frequency count + possible transformations), so the word error rate (WER) of the synthetic source-target pair is similar and within range of the supervised bitext data. Fazel-Zarandi et al. [68] extend this by adding out-of-vocabulary substitutions and varying the amount of error rate at the fragment level, allowing the synthetic error distribution to be different than the training data. I consider a simpler variation of this idea to control the amount of overall synthetic WER.

Before sampling a possible transformation for any fragment, a probability p is sampled from a uniform distribution and compared against the desired WER ρ. If $p \leq \rho$, a replacement t is sampled from $C(s, \cdot)$ with the constraint $t \neq s$, ensuring an edit for that fragment; otherwise the fragment is left unchanged. The same WER is applied for the out-of-vocabulary substitution step. This constraint is applied at line 21 of Algorithm 2. I generate synthetic datasets with ρ threshold set at $\{1\times, 2\times, 3\times, 4\times\}$ the WER value of the training bitext. A higher threshold introduces the correction model to more edit transformations per sentence during pre-training, but adds to the over-correction issue (Section 2.4.3) and makes the synthetic data edit distribution different than the inference data.

Partition Threshold The partitioning step of the SDA (Section 3.6.2) calculates the ratio φ (Equation 3.9) and compares it against the threshold θ to determine whether to

close the current fragment or extend it with the new word. A greater value of θ corresponds to extending a fragment only when that is more likely of the two outcomes in Equation 3.8, and ends up creating shorter fragments. For instance, if $\sim \theta = 0.50$, a bigger word span would have to occur more than half the time as its prefix in the bitext for φ to be greater than 0.50. Lowering the θ threshold allows the partitioning step to consider keeping multiple words together, which can have an impact on ASR errors that span word boundaries (liaison errors, e.g. "*I scream*" to "*ice cream*").

Target (unitext): `my regular army issue watch`
Source (ASR): `my regular i miss you watch`

Target span	Source span	Edit type
`my`	`my`	match
`regular`	`regular`	match
`army`	`i`	substitution
`issue`	`miss`	substitution
∅	`you`	insertion
`watch`	`watch`	match

Fragment-level confusion entry:

`army issue` → `i miss you` (C_{corrup})

`i miss you` → `army issue` (C_{correc})

Table 3.9: Word-level alignment for a liaison error. The ASR system incorrectly transcribes *army issue* as *i miss you*. The word-level alignment maps `army`→`i` and `issue`→`miss`, which is not a meaningful edit transformation on its own. Only the multiple-word fragment captures the error correctly, and it can only be sampled if both target words are assigned to the same partition.

Table 3.9 shows an example of this case, where the ASR system incorrectly transcribes *army issue* as *i miss you*. The word-level alignment to generate the confusion matrix (Section 3.6.1) splits this into `army`→`i` and `issue`→`miss`, with `you` as an additional insertion. The two substitutions do not represent a meaningful correction on their own, and sampling them separately would not recreate the actual edit.

The confusion matrix contains the complete transformation as a multi-word edit, `army issue` → `i miss you`, but this transformation can only be sampled if both words are assigned the same fragment during partitioning. If *army* occurs 20 times in the bitext-source and *army issue* occurs 4 times, $\varphi = 0.20$. If $\theta \geq 0.50$, the word *issue* initiates a new fragment and the multiple-word edit is not formed; for $\theta = 0.05$, the fragment can be extended, and the complete transformation is formed. Rare word sequences are most likely

to be incorrectly transcribed by an ASR system and will likely have a lower φ, so a lower threshold is needed for better partitioning. The optimal value of θ depends on the type of confusions present in the bitext dataset. I sweep the θ threshold value from 0.05 to 0.95 and select it experimentally (Section 3.8.2).

Insertion Edits ASR deletion errors (words present in the target but not in the source) require the correction model to make insertion edits (words that are not present in its input). Insertion edits can be difficult to learn, primarily when the missing content cannot be inferred from the surrounding transcription context, and they lead to overcorrections made by the model (Section 2.4.3). Insertion Edits account for a large portion of the errors present in the industry ASR data (Section 3.3.3). I explored generating synthetic data after removing these edits from the synthetic pairs (**NI**), and removing the insertion edits from the correction model output as a post-processing step (Section 3.7.4). Neither is used for the final results (Section 3.8.2).

3.7 ASR Correction Methods

3.7.1 Sequence-to-Sequence Correction Model

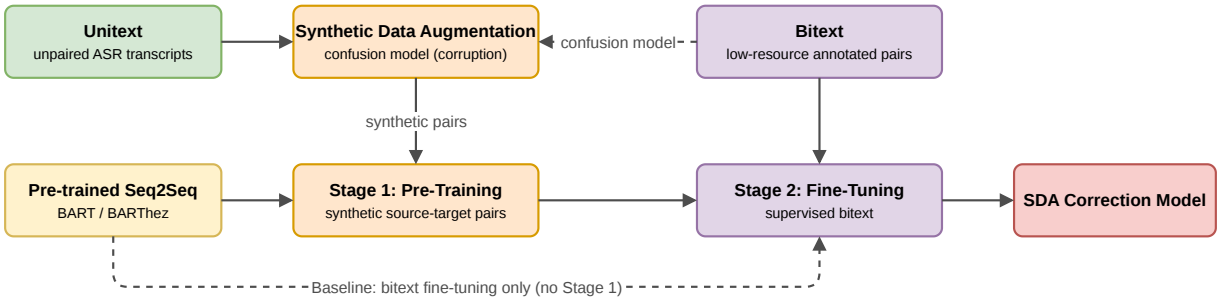

Figure 3.6: Overview of the Synthetic Data Augmented (SDA) Seq2Seq ASR transcription correction approach. The pre-trained Seq2Seq model is trained in two stages - pre-training on the synthetic data (Stage 1), then fine-tuning on the supervised bitext (Stage 2). The Seq2Seq Baseline model is only fine-tuned on the bitext.

The primary correction model is a Transformer-based sequence-to-sequence (Seq2Seq) encoder-decoder model (Section 2.2.2), which takes the ASR generate transcript as input and generates the corrected transcript. I use the BART-Base and BART-Large models

(Section 2.2.3) for English, and the BARThez and mBARThez models (Section 2.2.3) for French. The denoising pre-training objective of BART aligns with the ASR correction task (Section 2.4.2) making it a suitable model choice.

The SDA correction model is trained in two stages (Figure 3.6), similar to the staged training used in GEC curriculum learning (Section 2.2.5). In Stage 1, the model is pre-trained on the synthetic data generated from the unitext using the final SDA configuration (Section 3.8.2). In Stage 2, the model is further fine-tuned on the supervised bitext. Both stages use the Seq2Seq cross-entropy loss over the target tokens.

Apart from the SDA model, I also train a **baseline** (Seq2Seq Baseline) correction model for each dataset which is fine-tuned only on the supervised bitext with no synthetic pre-training (Stage 2 only). This allows me to evaluate the impact of SDA for the low-resource ASR Transcription Correction task.

3.7.2 LLM-Based Few-Shot Correction

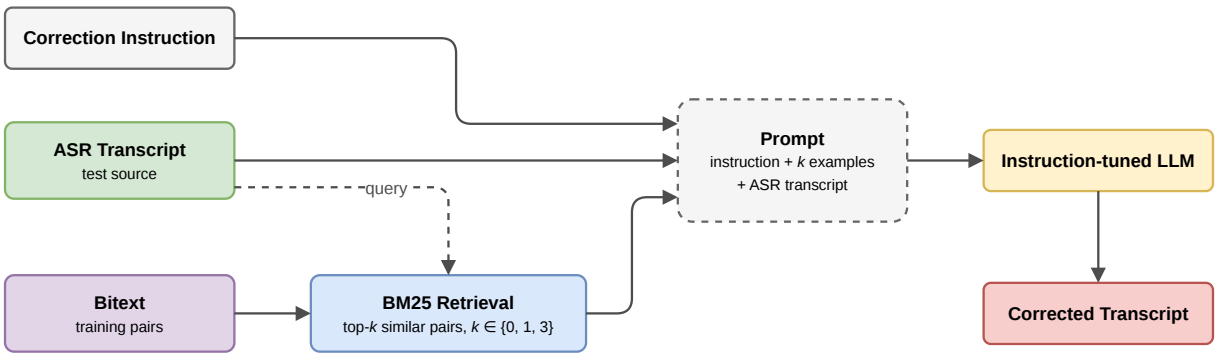

Figure 3.7: Overview of the LLM prompting-based ASR transcription correction method. The k in-context examples are selected from the bitext, based on BM25 similarity. For $k = 0$ (zero-shot), the prompt only contains the prompt instruction and the ASR-generated transcription.

As a complementary approach, I explore correction using instruction-tuned LLMs (Section 2.2.4). Given the low-resource constraint, no fine-tuning is performed and the LLM is used only with prompting (Section 2.2.6). The prompt contains the instruction to correct the ASR transcript, followed by k in-context correction examples and the ASR source transcript to be corrected (Figure 3.7, Algorithm 3). The in-context examples are selected from the training bitext using BM25 similarity [244], annotated pairs with the

most similar source to the test source transcript. I evaluate $k \in \{0, 1, 3\}$. The $k = 0$ setting is the zero-shot LLM baseline (Section 2.2.6), where the LLM relies only on the correction instruction, and $k \in \{1, 3\}$ are the few-shot (in-context learning) settings. The evaluated LLMs and the naming used in the results (e.g., Qwen-K3) are described in Section 3.8.1.

Algorithm 3 LLM Few-Shot Correction

Require: Test source sentence s_{test}; bitext $\mathcal{D} = \{(h_i, r_i)\}_{i=1}^{N}$; LLM $\mathcal{L}$; number of in-context examples k

Ensure: Corrected hypothesis $\hat{r}_{\text{test}}$

Retrieve the k most similar bitext examples (BM25)
$\mathcal{E} \leftarrow \text{BM25TopK}(s_{\text{test}}, \mathcal{D}, k)$
Construct few-shot prompt
prompt $\leftarrow$ "Correct the following ASR transcript:"
for all $(h_i, r_i) \in \mathcal{E}$ **do**
    prompt $\leftarrow$ *prompt* $\|$ "Input: " $\| h_i \|$ "Output: " $\| r_i$
end for
prompt $\leftarrow$ *prompt* $\|$ "Input: " $\| s_{\text{test}} \|$ "Output: "
Generate correction
$\hat{r}_{\text{test}} \leftarrow \mathcal{L}(\textit{prompt})$
return $\hat{r}_{\text{test}}$

Complete Bitext In-Context Example Correction During the few-shot approach, the LLM was only shown $k \leq 3$ retrieved few-shot examples. To consider LLM performance when it has access to the complete supervised data, I also use all 4,500 bitext pairs as in-context examples in a single prompt - (**Qwen1M-K4500**). Since this setting is not feasible for the industry dataset due to hardware constraints (Section 3.2), I only evaluate it on the public ASR datasets to compare the LLM correction performance without resource constraints. I use the same prompt instruction and format as Algorithm 3, but keep all the bitext examples in a fixed order during correction inference instead of ranking-selection by BM25. This is done deliberately so the instruction and bitext example input are computed and cached once and reused across all ASR transcription sentences I want to correct. The complete prompt contains ∼424,000 tokens for RED-ACE and ∼607,000 tokens for MLS-French, which is greater than the context length for Qwen2.5-7B (131,072 tokens). Thus, I switched to the Qwen2.5-7B-Instruct-1M [316] model (long-context variation of Qwen2.5-7B) to properly handle the long context. The Qwen1M-K4500 experiment with the full prompt required two NVIDIA L40S GPUs (48 GB each) for a single run, compared to a

single P100 and V100 GPU used for the Seq2Seq correction models (on both industry and public datasets). I also run the same model with $k = 3$ BM25 examples (**Qwen1M-K3**), to ensure that the change in performance comes from the number of examples and not from the change in model.

3.7.3 LLM-Based N-Best Correction with Synthetic Data

Building on the two approaches above, I explore a combined method (**N-Best**), where the SDA confusion model is used to create $N = 4$ candidate corrections for the ASR source transcript. Similar to the Generative Speech Error Correction (GenSEC) approach (Section 2.4.2), the LLM uses these candidates for additional context to generate the final correction, rather than performing selection or re-ranking among the candidates. Since I don't have access to the audio signal and the ASR decoder, the candidates are generated synthetically. Each candidate is generated by partitioning the ASR source transcript using different thresholds $\theta \in \{0.05, 0.15, 0.25, 0.35\}$ (Section 3.6.2) and sampling transformations from the correction (source $\rightarrow$ target) confusion matrix C_{correc} (Section 3.6.1). The four synthetic candidates along with the original ASR transcript form the n-best candidate list for the LLM. The prompt also includes the same BM25-selected $k \in \{0, 1, 3\}$ in-context examples as the few-shot approach (Figure 3.8, Algorithm 4).

3.7.4 Post-Processing

A recurring observation across all correction models is a tendency to over-correct: the models introduce extra edits that do not correspond to true ASR errors, increasing the WER Introduced (Section 2.4.3). To reduce this, I explored two post-processing approaches applied to the correction model output. (1) **Insertion edit removal** - removing all insertion edits made by the correction model, since ASR deletion errors (missing content) are difficult to correct from text alone (Section 3.3.3). (2) **LLM-based edit selection** - prompting an LLM to classify whether each model correction edit improves the source transcription, and filtering the edits that are not approved.

Neither approach improved the correction performance consistently. Insertion edit removal reduced both the WER Corrected and WER Introduced, which in some datasets removed too many valid corrections, causing the WER to increase after correction. The LLM-based edit selection did not perform as well as the insertion edit removal approach. As a result, neither post-processing approach is used for the final results reported in Section 3.8.3.

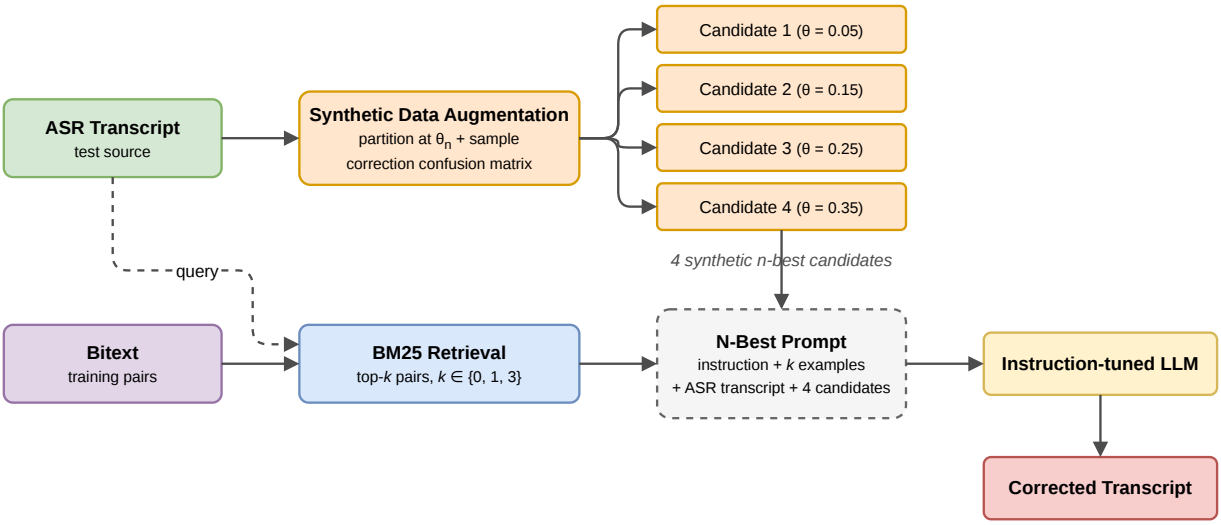

Figure 3.8: Overview of the LLM-based ASR transcription correction using synthetic n-best candidates. The SDA approach is used to generate four candidate corrections for the ASR source transcript at different thresholds θ. The four candidates and the ASR transcript serve as the n-best list for the LLM.

3.8 Experiments

3.8.1 Experimental Setup

Datasets All methods are evaluated on the industry customer support call transcripts (English and French) dataset and the two public ASR datasets, RED-ACE (English) and MLS (French), described in Section 3.3. For the public dataset, I sampled the bitext of 4,500 pairs (Table 3.4) to replicate the industry setting, using the remaining source transcripts as the unitext data for synthetic data augmentation. The industry development sets are also used for the human annotation study (Section 3.4) and for the Error Category Analysis in Section 3.8.4. The synthetic data configuration is evaluated using the bitext-training and development data, with the final results being reported for the dev and test sets. The RED-ACE (English) results are reported separately for the *clean* and *other* data splits.

Correction Model Architecture The sequence-to-sequence correction models use the BART-Base and BART-Large models [158] for English transcripts, and BARThez and mBARThez (Section 2.2.3) for French transcripts. Each model is pre-trained on the synthetic

Algorithm 4 LLM N-Best Correction

Require: Test source sentence s_{test}; correction confusion matrix C_{correc}; LLM $\mathcal{L}$; bitext $\mathcal{D}$; number of in-context examples k; threshold set $\Theta = \{0.05, 0.15, 0.25, 0.35\}$ $(N = 4)$

Ensure: Corrected hypothesis $\hat{r}_{\text{test}}$

Stage 1: Generate N synthetic candidate corrections
$\mathcal{C} \leftarrow \emptyset$
for all $\theta_n \in \Theta$ **do**
  $f_1, \ldots, f_K \leftarrow \textsc{Partition}(s_{\text{test}}, C_{\text{correc}}, \theta_n)$
  $\hat{h}_n \leftarrow \hat{f}_1 \cdots \hat{f}_K, \;\; \hat{f}_j \sim P(\cdot \mid f_j; C_{\text{correc}})$ // Eq. 3.11
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\hat{h}_n\}$
end for
Stage 2: Construct N-Best Prompt
$\mathcal{E} \leftarrow \text{BM25}\textsc{TopK}(s_{\text{test}}, \mathcal{D}, k)$
prompt $\leftarrow$ "Correct the following ASR transcript. Candidate corrections are provided as additional context:"
for all $(h_i, r_i) \in \mathcal{E}$ **do**
  prompt $\leftarrow$ *prompt* $\|$ "Input: " $\| \, h_i \, \|$ "Output: " $\| \, r_i$
end for
prompt $\leftarrow$ *prompt* $\|$ "Input: " $\| \, s_{\text{test}}$
for all $\hat{h}_n \in \mathcal{C}$ **do**
  prompt $\leftarrow$ *prompt* $\|$ "Candidate " $\| \, n \, \|$ ": " $\| \, \hat{h}_n$
end for
prompt $\leftarrow$ *prompt* $\|$ "Output: "
Stage 3: Generate Final Correction
$\hat{r}_{\text{test}} \leftarrow \mathcal{L}(\textit{prompt})$
return $\hat{r}_{\text{test}}$

data and then fine-tuned on the supervised bitext (Section 3.7.1). All configurations are trained from the same pre-trained checkpoint with identical hyperparameter settings.

LLM Models I evaluate four open-weight instruction-tuned LLMs on the public datasets for the LLM-based correction methods (Sections 3.7.2 and 3.7.3): Qwen2.5-7B [233], LLaMA3.1-8B [173], DeepSeek-R1-Distill-Qwen-7B [57], and Mistral-7B [117]. The LLMs are used in a prompting-only approach with $k \in \{0, 1, 3\}$ BM25-retrieved in-context examples from the bitext. Qwen2.5-7B obtained the best correction performance on both public datasets, so I only report the Qwen2.5-7B results in this chapter; the results for all

four LLMs are presented in Appendix C.2. Qwen2.5-7B was also selected as the only LLM for the industry dataset, due to hardware resource constraints. The full bitext few-shot approach (Section 3.7.2) is only evaluated using the Qwen model (Qwen2.5-7B-Instruct-1M).

Correction Models Naming Convention The ASR transcription correction models are named by their base model architecture and the synthetic data configuration used for pre-training. I describe the model naming used throughout this section in Table 3.10.

Name	**Meaning**
Seq2Seq base architecture	
BB / BL	BART-Base / BART-Large (English)
BH / MBH	BARThez / mBARThez (French)
Seq2Seq correction models (Section 3.7.1)	
Seq2Seq Baseline	Fine-tuned on the supervised bitext only
SDA	Pre-trained on the synthetic data, then fine-tuned on the bitext
LLM correction methods	
Qwen-Kk	Few-shot prompting with $k \in \{0, 1, 3\}$ in-context examples (Section 3.7.2)
Qwen-NB-Kk	Synthetic n-best correction with k in-context examples (Section 3.7.3)
Qwen1M-K3	Qwen2.5-7B-Instruct-1M with $k = 3$ BM25 in-context examples (Section 3.7.2)
Qwen1M-K4500	Qwen2.5-7B-Instruct-1M with all 4,500 bitext pairs as in-context examples (Section 3.7.2)

Table 3.10: Naming convention for the correction models reported in the results. The other evaluated LLMs (LLaMA, Mistral and DeepSeek) follow the same naming in Appendix C.2.

Evaluation Metrics The correction model's performance is evaluated using the Word Error Rate (WER) metric, along with the WER Corrected and WER Introduced breakdown explained in Section 3.5. The industry test set WER is computed at the call transcription level, and the development set and public dataset WER at the utterance level (Section 3.3.1).

Multiple-Run Evaluation and Statistical Significance To account for the randomness during model training, each correction method was trained and evaluated using six seed values, and the scores in the results tables are the average over these six runs (Appendix B).

Statistical significance was calculated using a one-tailed Wilcoxon signed-rank test on the six per-seed results, with the significance level set to $\alpha = 0.05$. Since a lower WER is better, the alternative hypothesis was that the correction approach reduces the ASR-normalized source WER. A statistically significant WER reduction over the ASR-normalized source WER ($p < 0.05$) is denoted with a downward arrow ($\downarrow$) next to the WER value. Due to compute constraint, the Qwen1M-K3 and Qwen1M-K4500 settings was run once (seed 42) instead of six times, to compare the correction performance trend with the Qwen2.5-7B base model.

3.8.2 Synthetic Data Configuration Selection

Generation Direction and Partition Threshold These two settings are selected by generating the synthetic data and comparing it with the real data reference, without training any correction model. I split the supervised bitext 90 : 10, using the 90% split to build the confusion model, and generate synthetic data from the 10% split. The data is generated in both directions - correction (adding edits to bitext-source compared with the bitext-target) and corruption (adding edits to bitext-target compared with the bitext-source). The WER between the generated text and the corresponding reference is used to evaluate the generation process. The generation is performed for varying θ threshold values in the SDA partitioning step (Section 3.6.5). Figure 3.9 presents the WER values for the threshold sweep for both directions.

The corruption direction, where the bitext-target is considered as the input to the synthetic data augmentation method, and the generated output is compared with the bitext-source, gets a lower WER than correction on both industry datasets and the public RED-ACE, and is similar for MLS. The gap between the corruption and correction direction plot depends on the edit type distribution of each dataset (Section 3.3.3). It is greater for the industry data, with mainly deletion errors, and smaller for the public ASR datasets, with greater substitution errors. This can also be explained since an ASR deletion edit is an insertion transformation in the correction direction, so the generated sentence contains content not present in its input. The threshold sweep shows that $\theta = 0.05$ gives the lowest WER in every corruption curve, and I select this partitioning threshold for generating synthetic data for model pretraining. I do not consider $\theta < 0.05$, as that makes the partitioning step unnecessarily combine adjacent words, producing long fragments that are left unedited.

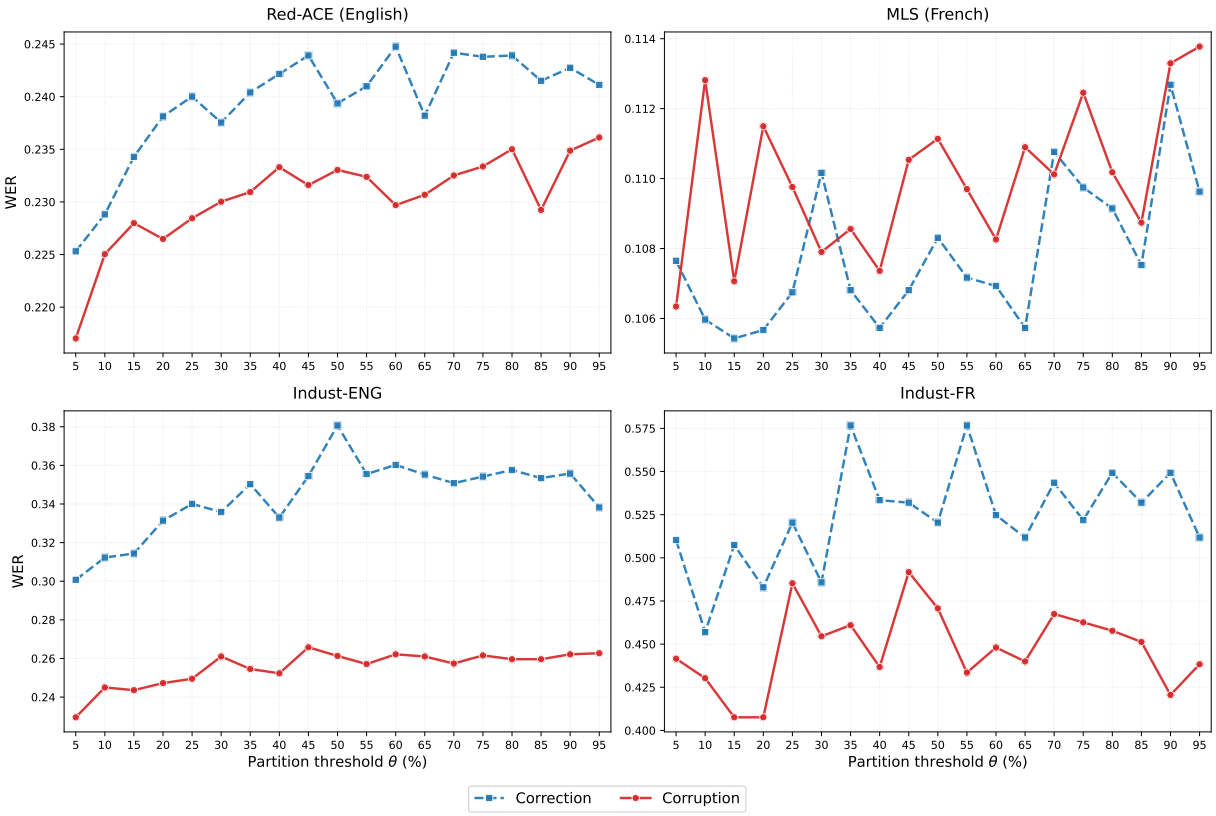

Figure 3.9: Word Error Rate (WER) between the Synthetic Data Augmentation (SDA) output for the starting sentence at different partitioning thresholds θ. A lower WER indicates that the generated text is closer to the desired reference. The plot lines indicate the two directions of SDA - corruption (bitext-target + edits -vs- bitext-source) and correction (bitext-source + edits -vs- bitext-target). The curve values are an average over six runs.

Synthetic Word Error Rate Increasing the error rate (ρ) in the synthetic data augmentation method increases the distance of the generated output from the reference by design. So I evaluate the impact of WER introduction by pre-training the correction model on each ρ - synthetic dataset and evaluating its performance on the ASR dataset development split (Table 3.11).

Increasing the ρ improves the amount of WER the model corrects, but raises the overall WER significantly, which denotes a high value for WER-introduced. This is more prominent for the correction direction, where the $4 \times \rho$ ends up with a WER of 51.08 on Indust-ENG

ρ	Direction	Indust-ENG		Indust-FR		RED-ACE		MLS	
		WER	WER_{Corr}	WER	WER_{Corr}	WER	WER_{Corr}	WER	WER_{Corr}
ASR Source		16.80	—	31.33	—	13.74	—	7.76	—
1×	Corrup	**20.84**	0.36	**39.17**	0.92	**15.28**	0.75	**8.78**	0.21
	Correc	22.82	0.32	43.87	0.64	15.44	0.01	8.91	0.02
2×	Corrup	20.42	0.47	40.05	1.12	15.48	1.15	8.99	0.27
	Correc	23.72	0.23	42.99	0.74	16.69	0.04	9.72	0.03
3×	Corrup	21.95	0.82	44.61	1.82	15.98	1.50	9.21	0.48
	Correc	30.83	0.34	52.41	0.68	23.28	0.28	14.62	0.18
4×	Corrup	25.05	1.02	46.32	1.99	17.17	1.77	12.16	0.94
	Correc	51.08	0.60	57.16	0.93	38.35	0.74	25.47	0.44

Table 3.11: Development set WER (%) and WER Corrected (%) for correction models pre-trained on synthetic data generated at ρ word error rate and in both generation directions, with $\theta = 0.05$. All values are obtained before fine-tuning on the supervised bitext using BART-Large and mBARThez.

compared to a starting dataset source WER of 16.80. The corruption direction at $\rho = 1\times$ has the lowest WER on three of the four datasets. Table 3.11 and Figure 3.9 together indicate that corruption is better than correction, with $\rho = 1\times$. The final configuration setting used for all the experiments is **corruption, $\rho = 1\times$, $\theta = 0.05$.**

Insertion Edits There are two options available for handling the extra insertion edits the correction model makes, which leads to high WER. The first is to remove them from all the training data pairs, ignoring the ASR deletion edits from the bitext and the transformations stored in the confusion matrix. This allows the correction model not to be trained on synthetic data with content missing from the source. This approach has two limitations. It modifies the bitext error distribution that the confusion matrix sampled transformations produce, and also, the word-level alignment does not always assign the edit-type label perfectly. For instance, Table 3.9 shows that the multi-word substitution *army issue* → *i miss you* is aligned as two substitutions and an insertion edit, so removing the insertion edit at the alignment level would destroy the substitution edit transformation. As a result, I keep all edit types in the training data for the final configuration. However, training with and without the insertion edits showed that the Seq2Seq correction model can be trained to selectively perform specific edit types, which motivated the Edit Type Ensembling

framework in Chapter 4.

Dataset	Model	Model output			+ Insertion edit removal		
		WER	WER_{Corr}	$\text{WER}_{\text{Introd}}$	WER	WER_{Corr}	$\text{WER}_{\text{Introd}}$
Indust-ENG	Baseline	17.18	0.42	0.80	17.14	0.34	0.68
	SDA	20.84	0.36	4.38	19.70	0.23	3.12
Indust-FR	Baseline	31.49	1.17	1.33	31.33	0.81	0.81
	SDA	39.17	0.92	8.77	36.97	0.71	6.34
RED-ACE	Baseline	13.91	0.05	0.21	13.87	0.05	0.17
	SDA	15.28	0.75	2.27	15.12	0.67	2.05
MLS	Baseline	7.87	0.82	0.94	7.74	0.83	0.81
	SDA	8.78	0.21	1.23	8.65	0.18	1.07

Table 3.12: Impact of removing all insertion edits from the correction model output as a post-processing step. The Baseline model is fine-tuned using only the supervised bitext; the SDA (synthetic data augmentation) model is pre-trained on synthetic data generated with the final configuration (corruption,$\rho = 1\times$, $\theta = 0.05$). Results are reported on the development set using BART-Large (English) and mBARThez (French) for all datasets.

The second option is removing the insertion edits from the model output as a post-processing step (Section 3.7.4). Table 3.12 shows the development set results for the SDA model and the bitext-trained baseline, using BART-Large (English) and mBARThez (French). Removing the insertion edits reduces both the WER Corrected and WER Introduced, and slightly reduces the overall development set WER. However, this improvement was not consistent, and in some cases the step removed too many valid corrections and increased the WER, so it is not used for the final results.

3.8.3 Results

I report the correction performance on the two public ASR datasets: RED-ACE (English) and MLS (French) in Tables 3.15 and 3.16. For the RED-ACE dataset, scores are pooled over the *clean* and *other* data splits (weighted by the number of reference words). For the LLM-based methods, I report the results of Qwen2.5-7B, which obtained the best results among the four evaluated LLMs; the results of all four LLMs are presented in Appendix C.2. Correction results on the Industry dataset are presented in Tables 3.13 and 3.14, where only Qwen2.5-7B was evaluated due to hardware constraints. In the results table, WER_{Corr} and

$\text{WER}_{\text{Introd}}$ are the word-level error corrected and error introduced rates (Section 3.5), and the value in parentheses is the % error corrected (WER_{Corr} / ASR Normalized $\times 100$). Each value is averaged over six runs (Appendix B). $\downarrow$ denotes a statistically significant WER reduction over the source WER (ASR Normalized) in a one-tailed Wilcoxon signed-rank test across the six paired runs ($p < 0.05$). All values are percentages.

	Dev Set			Test Set		
Model	**WER**	**WER_{Corr}**	**$\text{WER}_{\text{Introd}}$**	**WER**	**WER_{Corr}**	**$\text{WER}_{\text{Introd}}$**
Industry - English Results						
ASR Normalized	16.80	—	—	10.63	—	—
Manual Annotation	24.44	1.45 (8.63%)	9.10	—	—	—
Baseline - BB	17.11	0.42 (2.53%)	0.74	11.68	0.25 (2.37%)	1.30
Baseline - BL	17.18	0.53 (3.16%)	0.91	11.59	0.30 (2.86%)	1.27
SDA - BB	17.30	1.81 (10.75%)	2.31	$10.34^{\downarrow}$	0.92 (8.65%)	0.63
SDA - BL	18.90	1.88 (11.17%)	3.98	$10.41^{\downarrow}$	0.80 (7.54%)	0.59
Qwen-K0	22.78	0.74 (4.43%)	6.73	14.29	0.56 (5.30%)	4.22
Qwen-K1	21.19	0.81 (4.85%)	5.20	12.70	0.64 (6.00%)	2.71
Qwen-K3	21.04	1.17 (6.95%)	5.41	12.62	0.70 (6.63%)	2.70
Qwen-NB-K0	20.20	1.24 (7.38%)	4.64	12.36	0.59 (5.51%)	2.32
Qwen-NB-K1	19.38	1.24 (7.38%)	3.82	11.85	0.58 (5.44%)	1.80
Qwen-NB-K3	19.45	1.45 (8.64%)	4.10	11.70	0.62 (5.79%)	1.68

Table 3.13: Correction results on the industry English dataset. $\downarrow$: statistically significant WER reduction over ASR Normalized ($p < 0.05$).

Synthetic data augmentation-based correction model reduces the net ASR Transcription error. On both public datasets, the SDA correction model is the only approach that results in a lower net Word Error Rate (WER) than the ASR source transcription. On RED-ACE, SDA-BL reduces the test WER from 14.15 to 12.75, and on MLS (SDA - BH) from 7.05 to 6.87. The bitext-only baseline model and all LLM-based approaches lead to an increase in the WER on both datasets; they introduce more error (over-correction) than they correct. A similar trend is observed on the industry test data (Tables 3.13 and 3.14), where the SDA approach obtains a lower WER than the ASR transcription source. On the English industry dev data, no approach reduces the overall WER compared to the source. This may be because the development set utterances contain more errors than the test set (source WER 16.80 vs 10.63) and have more subjective

Model	Dev Set			Test Set		
	WER	**WER_{Corr}**	**$\text{WER}_{\text{Introd}}$**	**WER**	**WER_{Corr}**	**$\text{WER}_{\text{Introd}}$**
Industry - French Results						
ASR Normalized	31.33	—	—	28.71	—	—
Manual Annotation	37.02	1.83 (5.84%)	7.52	—	—	—
Baseline - BH	31.21	0.74 (2.35%)	0.62	30.19	1.03 (3.58%)	2.50
Baseline - MBH	31.41	1.33 (4.23%)	1.41	29.84	2.51 (8.75%)	3.64
SDA - BH	$30.61^{\downarrow}$	2.14 (6.82%)	1.42	$28.61^{\downarrow}$	2.98 (10.39%)	2.88
SDA - MBH	$30.09^{\downarrow}$	3.24 (10.34%)	2.00	$28.18^{\downarrow}$	3.55 (12.37%)	3.02
Qwen-K0	41.41	0.52 (1.65%)	10.60	32.61	0.73 (2.55%)	4.64
Qwen-K1	37.86	1.18 (3.76%)	7.71	31.23	0.81 (2.83%)	3.34
Qwen-K3	36.99	1.55 (4.94%)	7.21	31.14	1.01 (3.51%)	3.44
Qwen-NB-K0	39.66	1.55 (4.94%)	9.88	32.24	1.39 (4.84%)	4.91
Qwen-NB-K1	37.07	1.84 (5.88%)	7.58	30.54	1.57 (5.46%)	3.39
Qwen-NB-K3	36.75	1.84 (5.88%)	7.26	30.81	1.60 (5.59%)	3.70

Table 3.14: Correction results on the industry French dataset. ↓: statistically significant WER reduction over ASR Normalized ($p < 0.05$).

references - even the manual annotation increases the development set WER to 24.44. This trend on the test sets is consistent with recent work that shows that task-specific models can outperform significantly larger general-purpose LLMs in a low-resource language ASR correction task [17, 145].

Without synthetic pre-training, the RED-ACE correction model makes minimal corrections. The importance of synthetic data can be seen by the performance of the bitext-only baseline on RED-ACE dataset. The model corrects only $0.23 - 0.84\%$ of existing issues in the ASR transcription, introducing extra errors. With only 4,500 training examples, the model ends up learning to mostly copy the source tokens. Synthetic data training improves the WER_{Corr} from 0.12 to 2.55 while the $\text{WER}_{\text{Introd}}$ raises from 0.65 to 1.15, which results in a net WER reduction. The MLS baseline does not perform that poor ($8 - 16.40\%$ corrections), which I attribute to the lower initial bitext WER of 6.14%, but the SDA model still improves the correction rate. A similar trend is observed on both industry datasets, where the WER_{Corr} value increases 1.4 to 4.3 times after SDA training.

Model	Dev Set			Test Set		
	WER	**WER_{Corr}**	**$\text{WER}_{\text{Introd}}$**	**WER**	**WER_{Corr}**	**$\text{WER}_{\text{Introd}}$**
Public — RED-ACE (English) Results						
ASR Normalized	13.74	—	—	14.15	—	—
Baseline - BB	14.00	0.04 (0.29%)	0.29	14.47	0.03 (0.23%)	0.35
Baseline - BL	14.21	0.11 (0.84%)	0.58	14.68	0.12 (0.83%)	0.65
SDA - BB	12.51$^{\downarrow}$	1.99 (14.48%)	0.76	12.99$^{\downarrow}$	1.97 (13.94%)	0.81
SDA - BL	12.19$^{\downarrow}$	2.66 (19.38%)	1.11	12.75$^{\downarrow}$	2.55 (18.04%)	1.15
Qwen-K0	15.09	1.04 (7.60%)	2.39	15.50	1.04 (7.37%)	2.39
Qwen-K1	14.62	1.11 (8.10%)	1.99	15.06	1.11 (7.87%)	2.02
Qwen-K3	14.98	1.35 (9.85%)	2.59	15.43	1.37 (9.70%)	2.65
Qwen-NB-K0	13.93	0.92 (6.69%)	1.11	14.38	0.98 (6.90%)	1.20
Qwen-NB-K1	13.93	0.87 (6.30%)	1.05	14.41	0.90 (6.34%)	1.15
Qwen-NB-K3	13.93	0.92 (6.70%)	1.11	14.37	0.96 (6.79%)	1.18
Qwen1M-K3	15.15	1.26 (9.19%)	2.67	15.56	1.30 (9.16%)	2.70
Qwen1M-K4500	20.32	1.56 (11.38%)	8.14	20.92	1.58 (11.18%)	8.35

Table 3.15: Correction results on the public RED-ACE (English) dataset [79], pooled over the *clean* and *other* splits. Results for the other evaluated LLMs are in Appendix C.2. $\downarrow$: statistically significant WER reduction over ASR Normalized ($p < 0.05$). Qwen1M-K3 and Qwen1M-K4500 are single runs (Section 3.8.1).

In-context examples helps reduce the LLM over-editing. The WER breakdown shows the impact of using the k similar in-context examples for LLM based ASR transcription correction. On MLS, the Qwen few-shot test WER reduces from 9.02 ($k = 0$) to 7.59 ($k = 3$), and both the corrections (0.55 $\rightarrow$ 0.99) and the introduced error (2.52 $\rightarrow$ 1.53) improve. The same trend is observed on both industry test sets, where the introduced error drops from 4.22 to 2.70 (English) and from 4.64 to 3.44 (French). For RED-ACE, the improvement is less, with the best score obtained for $k = 1$ (15.06). This indicates that the similar task-examples teach the LLM how to perform minimal edits, reducing the over-corrections (Section 2.4.3). For the other three LLMs, the in-context examples also helped reduce the amount of over-editing (Appendix C.2).

Synthetic n-best candidates improve LLM correction. On RED-ACE, the best n-best result (Qwen-NB-K3 = 14.37) is better than the best LLM few-shot result (Qwen-K1 = 15.06) by 0.69 WER. For RED-ACE, all n-best settings perform better than the

Model	Dev Set			Test Set		
	WER	**WER$_{\text{Corr}}$**	**WER$_{\text{Introd}}$**	**WER**	**WER$_{\text{Corr}}$**	**WER$_{\text{Introd}}$**
Public — MLS (French) Results						
ASR Normalized	7.76	—	—	7.05	—	—
Baseline - BH	7.88	0.62 (8.03%)	0.74	7.10	0.86 (12.18%)	0.91
Baseline - MBH	7.87	0.82 (10.59%)	0.94	7.26	1.16 (16.40%)	1.37
SDA - BH	7.57$^{\downarrow}$	0.97 (12.51%)	0.78	6.87$^{\downarrow}$	1.15 (16.36%)	0.97
SDA - MBH	7.61$^{\downarrow}$	1.28 (16.44%)	1.12	6.92$^{\downarrow}$	1.51 (21.42%)	1.38
Qwen-K0	10.04	0.43 (5.52%)	2.71	9.02	0.55 (7.75%)	2.52
Qwen-K1	9.03	0.79 (10.24%)	2.06	8.06	0.87 (12.30%)	1.87
Qwen-K3	8.71	0.84 (10.86%)	1.80	7.59	0.99 (14.02%)	1.53
Qwen-NB-K0	8.88	1.02 (13.11%)	2.14	8.14	1.04 (14.75%)	2.13
Qwen-NB-K1	8.64	0.96 (12.37%)	1.84	7.77	1.01 (14.39%)	1.73
Qwen-NB-K3	8.64	0.95 (12.25%)	1.83	7.97	1.00 (14.22%)	1.92
Qwen1M-K3	8.95	0.96 (12.37%)	2.15	7.99	1.04 (14.72%)	1.98
Qwen1M-K4500	9.04	0.71 (9.22%)	2.00	8.38	0.81 (11.43%)	2.13

Table 3.16: Correction results on the public MLS (French) dataset [225]. Results for the other evaluated LLMs are in Appendix C.2. ↓: statistically significant WER reduction over ASR Normalized ($p < 0.05$). Qwen1M-K3 and Qwen1M-K4500 are single runs (Section 3.8.1).

few-shot setting. On both industry test sets, the n-best approach improves the few-shot approach with the same k examples, English: Qwen-NB-K3 = 11.70 vs Qwen-K3 = 12.62 and French: Qwen-NB-K1 = 30.54 vs Qwen-K1 = 31.23. The candidate list provides the LLM model with additional correction information sampled from the confusion matrix. The greatest improvement is observed when the prompt has $k = 0$ examples (except for the French-industry data, where it is $k = 1$) - on MLS, the zero-shot WER improves from 9.02 (Qwen-K0) to 8.14 (Qwen-NB-K0). The exception is MLS with $k = 3$ examples, where the few-shot setting gets a lower WER than the n-best method (7.59 vs 7.97).

All bitext examples do not improve LLM corrections. The Qwen1M-K3 transcript correction performance is similar to the Qwen-K3 on both public test sets (RED-ACE: 15.56 vs 15.43, MLS: 7.99 vs 7.59). Adding the complete 4,500 bitext pairs as few-shot examples increases the overall WER on both datasets, substantially on RED-ACE (15.56 → 20.92) and slightly on MLS (7.99 → 8.38). For the RED-ACE dataset, there is a large increase in the amount of over-corrections (introduced error) from 2.70 to 8.35. For the MLS dataset,

Category	n	Manual	SDA - BB	Qwen-K3	Qwen-NB-K3
Similar sounding	101	12.87	12.87	13.86	8.91
Missing short words	82	4.88	4.88	0.00	1.22
Symbols/numbers	53	13.21	20.75	28.30	28.30
Missing long sequences	49	2.04	0.00	0.00	2.04
Other	49	10.20	12.24	2.04	8.16
Tag error	49	2.04	0.00	0.00	0.00
Word variations	38	5.26	21.05	10.53	15.79
Extra word	32	15.63	21.88	15.63	18.75
Slang	32	3.13	21.88	3.13	6.25
Repeated words	17	11.76	11.76	17.65	17.65
Domain words	14	35.71	28.57	14.29	42.86
Questionable reference	12	0.00	0.00	0.00	8.33
Spelling out	3	0.00	33.33	0.00	0.00
Total	554	8.30	**11.37**	8.12	9.75

Table 3.17: Percentage of errors corrected for each category on the industry English development set. n is the number of instances per category (Table 3.5). Categories with no corrections by any approach (Hallucination, Anonymization and Contraction) are removed.

there is a reduction in the amount of corrections and an increase in the introduced error. The SDA model corrects more errors and introduces fewer overcorrections than the LLM with all bitext examples.

Qwen2.5-7B is the best performing LLM on both public datasets. Out of the four evaluated LLMs, Qwen2.5-7B obtains the lowest WER on both RED-ACE and MLS, for both few-shot and the n-best settings (Appendix C.2). This motivated the decision for selecting Qwen2.5-7B as the only LLM evaluated on the industry dataset (Section 3.8.1), where hardware constraints did not allow running all LLMs.

3.8.4 Error Category Analysis

To understand what types of errors the models are actually correcting, I analyze the corrections using the human-annotated error categories (Section 3.4). Tables 3.17 and 3.18 show the correction percentages of the Manual Annotation, the SDA model, and the Qwen few-shot (Qwen-K3) and n-best (Qwen-NB-K3) approaches for each error category, on the industry development data.

Category	n	Manual	SDA - MBH	Qwen-K3	Qwen-NB-K3
Similar sounding	136	8.82	9.56	7.35	8.09
Missing short words	118	2.54	5.93	2.54	4.24
Other	48	0.00	2.08	0.00	0.00
Filler word	34	0.00	11.76	0.00	5.88
Grammar	24	12.50	29.17	20.83	12.50
Spacing	13	0.00	30.77	0.00	0.00
Extra word	13	30.77	15.38	15.38	0.00
Symbols/numbers	10	10.00	10.00	10.00	10.00
Slang	8	0.00	25.00	0.00	12.50
Language variation	4	0.00	75.00	0.00	50.00
Domain word	3	66.67	0.00	0.00	0.00
Total	471	5.31	**9.34**	4.46	5.31

Table 3.18: Percentage of errors corrected for each category on the industry French development set. n is the number of instances per category (Table 3.6). Categories with no corrections by any approach (Missing long sequences, Contraction, Repeated words, Hallucination, English words, Tag error and Questionable reference) are removed.

The SDA model corrects the highest amount of annotated errors for both languages (11.37% English and 9.34% French), more than the manual annotation (8.30% and 5.31%) and both Qwen approaches, which is consistent with the higher WER_{Corr} on the development data (Section 3.8.3). The annotators also introduced a large amount of over-corrections ($\text{WER}_{\text{Introd}}$ of 9.10 for English, Table 3.13), which indicates the subjectivity of the ASR Transcription correction task (Section 3.4). The synthetic n-best candidates increase the corrections made by Qwen for both languages (English: 8.12% $\rightarrow$ 9.75%, French: 4.46% $\rightarrow$ 5.31%). However, several error types contain only a few instances, so I report the number for comparison below.

The error category scores show that the missing content stays missing after post-correction. For *Missing long sequences*, only one of the 49 English occurrences (2.04%) is corrected by any approach, and none of the 32 French occurrences. These are the transcription phrases lost by the ASR model due to noise and overlapping speakers, and the surrounding context can't recover them. *Missing short words* are also rarely recovered (at most 5.93%), and Qwen-K3 does not recover any in English. This is one of the reasons that motivated exploring the removal of insertion edits from the model output (Section 3.7.4); however, that did not result in consistent improvements and was ultimately not added.

The SDA model makes more corrections than the annotators on local and pattern-based edits that are repeated across transcripts: word variations and slang in English, and

grammar, filler words and language variation for French. These edits reoccur in the small bitext and can be substituted in the synthetic pre-training data. The SDA model also corrects 4 of the 13 French spacing errors, which no other approach corrects. Spacing errors span across word boundaries (e.g. *y a* → *ya*), so they are represented as multiple-word fragments, which the low partitioning threshold ($\theta = 0.05$) allows (Section 3.6.5).

For the domain-specific terms in English, Qwen-NB-K3 corrects 6 of the 14 instances (42.86%), compared with 5 for the annotators, 4 for the SDA model and 2 for Qwen-K3. The difference from the annotators is a single edit, so it does not indicate that the LLM corrects domain words better than the annotators. However, the difference from Qwen-K3 (6 vs 2) suggests that the synthetic candidates provide useful domain edits from the bitext confusion matrix that the LLM can use, although the number of instances is small. In French, the annotators correct two of the three domain word instances, while the correction models do not correct any.

The synthetic n-best LLM method does not improve in every setting. Qwen-K3 corrects more English similar sounding errors (14 vs 9 of 101) and more French grammar errors (5 vs 3 of 24) than Qwen-NB-K3. This might be due to the correction candidates restricting the LLM to edits present in the confusion matrix, impacting the LLM's own error correction knowledge.

The symbol and number normalization is the most corrected English category for Qwen-K3 and the second most corrected for Qwen-NB-K3 (15 of 53, 28.30% for both), compared with 11 for the SDA model and 7 for the annotators. These edits are stylistic conversions (e.g. *6* → *six*) that can be inferred from the few-shot examples by the correction model.

3.9 Conclusions and Future Directions

This chapter shows that the confusion matrix-based synthetic data augmentation (SDA) method for Seq2Seq pre-training is a viable strategy for ASR transcript correction in a low-resource, text-only setting. Across the test sets of the industry customer support transcripts and both public ASR datasets, SDA is the only method that achieves a lower WER than the ASR transcription source. On these test sets, the bitext-only baselines and every LLM configuration introduce more error than they remove. The augmentation approach increases the correction compared to the baseline, while the introduced error increases much less (e.g., RED-ACE SDA-BB: WER_{Corr} 0.03 → 1.97, $\text{WER}_{\text{Introd}}$ 0.35 → 0.81). Providing the LLM with all 4,500 bitext pairs as in-context examples does not close the gap. It increases the WER on both public datasets compared to the $k = 3$ few-shot setting, mainly through more introduced error on RED-ACE and fewer corrections on MLS.

Beyond the WER results, the method fits the constraints that motivated the task (Section 3.2). It requires no audio, no access to the ASR decoder, and no confidence scores - only the ASR system's own text output and a few thousand annotated pairs, which is what a deployment behind a proprietary speech-to-text service and customer privacy regulations leaves available. The synthetic data is generated offline as a pre-processing step, so the correction model architecture and inference pipeline are unchanged: a BART-sized model that trains and runs locally on older GPU hardware and is suitable for the low-latency inference a production correction service needs. The LLM approaches, which cost considerably more per correction (compute and time), do not perform better on either public dataset, including when the complete bitext is provided in the few-shot setting. LLM-based correction remains complementary rather than competitive - the synthetic n-best candidates improve it in most settings, the most when no in-context examples are provided, and help it on some error categories such as domain words (Section 3.8.4).

Several directions follow from this work. The most immediate is that the confusion matrix is a far more extensible object than the way I use it here. It is a plain lookup table of observed fragment transformations, so its contents can be edited, weighted, or extended without retraining anything: substitutions involving domain terminology, product names, or numerical entities can be up-weighted so that the synthetic data over-represents the errors that matter most for the downstream application, and entries can be seeded manually from a domain lexicon for terms that never appear in the small bitext. The same property makes the approach cheap to maintain — when the deployed ASR system is updated and its error distribution shifts, only the matrix has to be rebuilt from a fresh set of annotated pairs. A related direction is to relax the conditional independence assumption of Equation 3.10 and condition fragment sampling on the surrounding context, which would produce synthetic errors that cluster the way real ASR errors do.

Ensemble methods combining the Seq2Seq model and the LLM are a natural next step, using the precision of the first and the vocabulary coverage of the second to compensate for each other, in the spirit of the edit-level combination explored in Chapter 4. Working with shorter correction spans, focusing on 5–10 word windows rather than full sentences, may reduce error propagation and better match the local nature of most ASR errors. Incorporating the audio signal alongside the transcript text, rather than restricting to text-only post-editing, could make the missing-word errors recoverable that no method in this chapter (including manual annotation) was able to correct. Better-aligned transcripts with consistent speaker segmentation and improved anonymization would also reduce the noise in the training signal. Finally, for downstream applications where transcript fluency matters more than WER fidelity, treating the correction task as a constrained paraphrase problem rather than a strict error correction problem may be a more appropriate framing.

Chapter 4

Edit Type Ensemble Models

4.1 Introduction

Automatic text generation systems for text quality improvement tasks such as Grammatical Error Correction (GEC) are generally designed using a sequence-to-sequence (Seq2Seq) model architecture (Section 2.5.2), where a model is trained end-to-end to generate the correction output conditioned on the erroneous source text input (Figure 4.1).

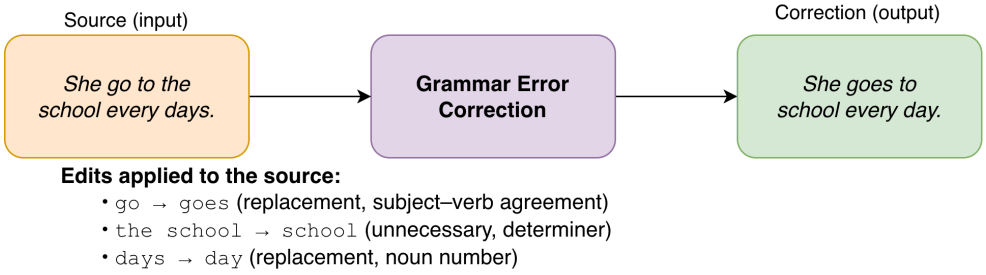

Figure 4.1: The Grammatical Error Correction (GEC) task. Given a source sentence, the system produces a correction (improved version of the source). Each correction can be represented as atomic edits - substitution, insertion, and deletion - which are used by the Edit-Type Ensemble framework.

The corrections made by a GEC Model correspond to three atomic edit operations - **substitution** of incorrect tokens, **insertion** of missing tokens, and **deletion** of unnecessary tokens, similar to the edit operations defined by the Levenshtein distance (Section 2.3.3).

In this chapter, I label these operations using the ERRANT toolkit's GEC notation [27] - **R** (Replacement, substitution), **M** (Missing, insertion), and **U** (Unnecessary, deletion). The four edit-specific correction models, in this work (Figure 4.2), are denoted as M_R, M_M, M_U, and M_{All}, where M_{All} is the baseline trained on all edit types together.

Since a text span can have multiple issues of different kinds, a Seq2Seq text correction model learns to capture patterns between these different edits. This allows the model to perform various corrections containing various complex and connected edits like fluency and stylistic consistency, rephrasing, word reordering, etc. However, this joint-learning approach, which improves the model's complex syntactic rephrasing knowledge, may cause the model to reduce focus on simple independent atomic edits like punctuation (insertion), spelling errors, missing articles, etc.

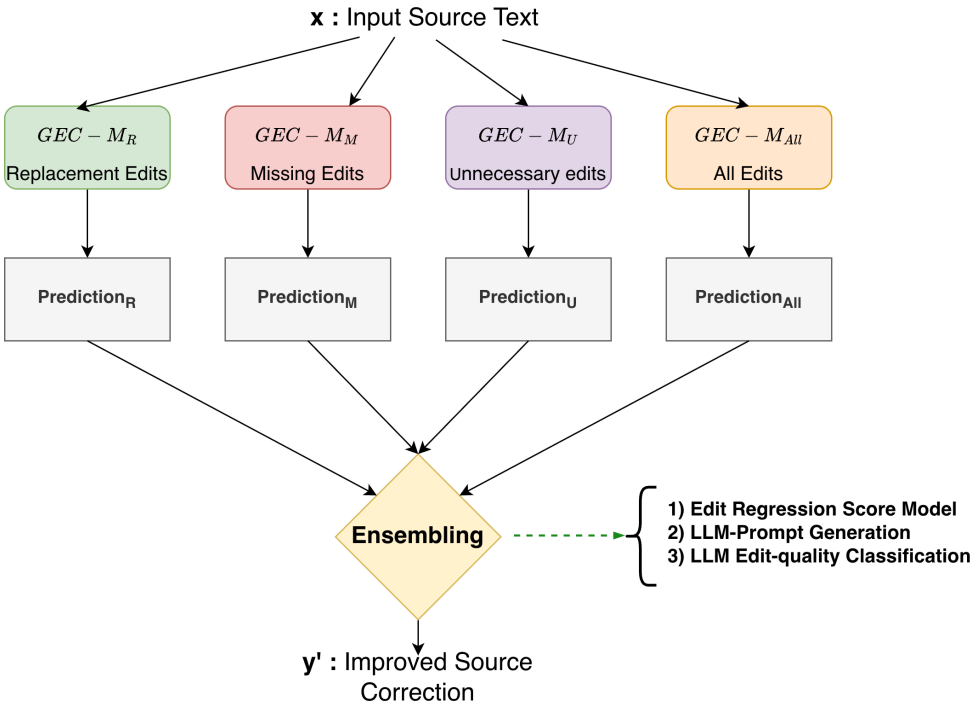

Figure 4.2: Summary of the complete project. Four edit-specific GEC models - M_R (substitution), M_M (insertion), M_U (deletion), and M_{All} (all edits) generate the correction predictions, which are combined by one of three ensembling approaches to produce the improved correction.

In the previous chapter, I explored and observed the impact of training ASR Transcript correction models with and without insertion edits. This allowed me to realize that Seq2Seq

models can be trained to perform (or not perform) a specific kind of edit operation. Building on this fact, this chapter proposes an Edit Type Ensembling (ETE) framework, which breaks down the overall GEC task into specific correction sub-tasks based on the three primary edit operations - Insertion, Deletion, and Substitution. Apart from the regular baseline correction model trained on all connected edits, three independent models are trained on the individual edit types. The predictions from these four models are combined using an edit-type-based ensembling approach to obtain the final correction. Figure 4.2 gives an overview of the complete framework.

As shown in Figure 4.2, the source text is passed as input to the four sequence-to-sequence (Seq2Seq) models that share the same architecture but are trained on different edit-specific data: a substitution model M_R, an insertion model M_M, a deletion model M_U, and the baseline M_{All} trained on all edit types. Each model produces its own correction, and the four predictions are combined in the ensembling stage. I use three ensembling approaches - (1) ESC edit-level regression combiner (Section 4.3.2), (2) an LLM prompt-based generator and (3) an LLM edit-quality classifier (Section 4.3.3). The final output $\hat{y}$ is the source sentence with the selected edits applied.

The driving hypothesis of this chapter is that a Seq2Seq model trained on all edit operations captures complex syntactic connected edits like rephrasing and rewrites effectively, but may not optimally learn the structurally simpler, independent operations. Training edit-specific models on these individual operations and ensembling their predictions can make more efficient use of the same limited training data, producing a stronger GEC system with the same baseline architecture.

The background for the GEC task, the edit-type-aware correction work, and the different ensembling approaches this chapter builds upon are presented in the GEC section of the background chapter (Section 2.5).

4.2 Motivation and Contributions

Most of the improvements proposed by recent GEC systems stem from adding something: a very large synthetic pre-training corpus, larger or specialized neural architectures, or instruction-tuned LLMs (Section 2.5.3). Most of these approaches introduce extra cost/-complexity somewhere in the pipeline (for LLM-based solutions, the cost falls on inference). This is important for a deployment solution when the correction model runs continuously over a large amount of input text, and where a precise model that can be run cheaply at scale is worth more than a fluent one that cannot (most of the industry production settings).

GEC is suited to a smaller task-specific correction model since it is a well-defined task with public training data, and is scored using the $F_{0.5}$ metric, which penalizes introduced errors more than a missed correction - the main issue of using LLMs for GEC (Section 2.5.5).

In this chapter, I focus on how to use the available training data effectively. I keep the training data, the baseline model architecture (BART), and the hyperparameters fixed, splitting the correction task on the different GEC edit types. The goal is to obtain a precise and fast-inference GEC solution that can be used in a production setting. To achieve this, I train three edit-specific models - M_R, M_M, and M_U, apart from the standard M_{All} baseline, and combine their predictions at the edit level. The hypothesis is that the joint M_{All} model learns complex syntactical edits like rephrasing well but may underperform on simpler independent operations, and that training a model for each edit operation and ensembling the predictions can result in a higher-confidence and higher-quality set of correction edits. Since all four models share the same BART architecture, this keeps the inference simple and inexpensive to run at scale - the setting that motivates the work.

4.2.1 Contributions

I present the following contributions in this chapter:

- The Edit Type Ensemble (ETE) framework, which decomposes GEC into substitution, insertion, and deletion sub-tasks using ERRANT labels, training edit-specific models for each, and combining their predictions.
- A two-stage curriculum training applied for all four base models, which improves on the published BART baseline and provides a better reference point for calculating the impact of ensembling.
- Two LLM-based ensembling methods - prompt generation and edit quality classification. Evaluated using four open-weight LLMs, which have consistently higher recall but lower precision than the ESC ensemble.
- A sentence and edit-level result evaluation, showing improvement of the regression ensembling approach on all edit operations on BEA-2019, and on substitution and insertion edits on CoNLL-2014.
- A parameter-efficient version of the ensemble framework that replaces the four separate fully fine-tuned models with a single frozen base model and four edit-specific adapters. On BART-Base the adapters match full fine-tuning for all models, training only 10%

of the parameters, and all four corrections are generated in a single inference pass, 1.76× faster for single-sentence correction with 2.8× less storage.

4.2.2 Chapter Structure

I present the edit-type dataset construction and the two-stage curriculum training for all models in Section 4.3.1. Section 4.3.2 describes the regression-based ESC system combination framework that I use in my work, and Section 4.3.3 introduces the two LLM-based alternative approaches. Section 4.4 reports the experiments and results on the BEA-2019 and CoNLL-2014 test sets, with the edit-level breakdown. Section 4.5 replaces the four edit models with a single frozen base model and four edit-specific adapters, and compares the two for correction quality, inference time and storage, followed by the discussion and summary.

4.3 Methodology

4.3.1 Individual Edit-Based Models for GEC

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ denote the full parallel GEC training dataset, where x_i is the input source text and y_i is its target correction. For each training pair, I use the ERRANT toolkit (Section 2.5.7) to extract the edit set $\mathcal{E}(x_i, y_i)$ and annotate each edit e with its operation type - $\text{op}(e) \in \{R, M, U\}$ - Replacement, Missing, and Unnecessary, as introduced above.

I use $\mathcal{D}$ to create multiple edit-specialized training datasets that contain target text with only the desired edit-type correction:

$$\begin{aligned}
\mathcal{D}_R &= \{(x_i, y_i^R) \mid \exists\, e \in \mathcal{E}(x_i, y_i) : \text{op}(e) = R\} \\
\mathcal{D}_M &= \{(x_i, y_i^M) \mid \exists\, e \in \mathcal{E}(x_i, y_i) : \text{op}(e) = M\} \\
\mathcal{D}_U &= \{(x_i, y_i^U) \mid \exists\, e \in \mathcal{E}(x_i, y_i) : \text{op}(e) = U\} \\
\mathcal{D}_{All} &= \mathcal{D}
\end{aligned}$$

These datasets are used to train the three edit-specialized GEC Baseline Models. Apart from the edit-specific model, I also use $\mathcal{D}$ with all edits to train the baseline GEC Model. This gives us the four GEC Models ($\mathcal{M}$) and their predictions, which are used for further analysis, evaluation and ensembling to generate the final target prediction (Figure 4.3) - $\mathcal{M} = \{M_{\text{All}}, M_{\text{R}}, M_{\text{M}}, M_{\text{U}}\}$.

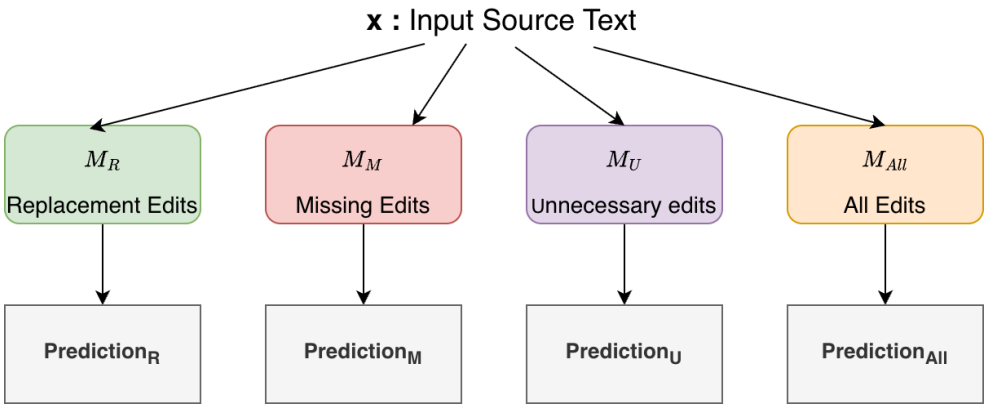

Figure 4.3: Overview of the four edit-specific baseline GEC models (M_i) which generate the correction predictions. The final correction is proposed by analyzing, evaluating and ensembling the four individual predictions.

We train four sequence-to-sequence models on the edit-specific dataset variations of $\mathcal{D}$ described above:

- M_{All}: trained on $\mathcal{D}_{All}$, the original dataset. This acts as the general GEC baseline model, learning complex multi-edit correction patterns including stylistic paraphrasing, word reordering, and connected edits across different edit types.
- M_R (Substitution/Replacement): trained on $\mathcal{D}_R$. Focuses on replacing incorrect words/phrases with better alternatives.
- M_M (Insertion/Missing): trained on $\mathcal{D}_M$. Predicts insertion edits that may be missing in the source text.
- M_U (Deletion/Unnecessary): trained on $\mathcal{D}_U$. Specializes in removing erroneous or redundant parts of the source text.

All four GEC models share the same base architecture and pre-trained checkpoint - BART-Base, BART-Large [158]. Training hyperparameters — learning rate, batch size, number of steps, and beam decoding settings — were identical across all the model variations, so I can observe and compare the impact of edit-type ensembling. I also follow the standard GEC Model training practice of filtering the training dataset to only contain source-target pairs where the target has some corrections made.

$$\mathcal{D} = \{(x_i, y_i) \mid x_i \neq y_i\}_{i=1}^{N}$$

Unlike Machine Translation or other Conditional Text Generation tasks, most of the tokens between x_i and y_i remain the same. Hence, training the model only on source-target text pairs where $source \neq target$ enforces the model to learn correction patterns and edits better.

Curriculum Learning Training I follow the staged training approach proposed by GECToR [203]. I perform a curriculum learning two-stage training strategy for all four edit-specialized base models - $\mathcal{M} = \{M_{\text{All}}, M_{\text{R}}, M_{\text{M}}, M_{\text{U}}\}$. This has been shown to improve GEC performance by teaching the model broad grammatical corrections and then refining it on a cleaner, stronger GEC dataset.

Stage 1 - GEC Training I use the pre-trained BART checkpoint [158] to train each model on the combined dataset $\mathcal{D}_{\text{S1}}$, which consists of four publicly available GEC corpus:

$$\mathcal{D}_{\text{S1}} = \text{FCE} \cup \text{W\&I+LOCNESS} \cup \text{NUCLE} \cup \text{Lang-8}$$

This stage teaches the models how to perform a broad and diverse set of possible GEC corrections across different proficiency levels and domains.

Stage 2 - GEC Fine-tuning The Stage 1 trained model checkpoint is then fine-tuned on just the W&I+LOCNESS alone:

$$\mathcal{D}_{\text{S2}} = \text{W\&I+LOCNESS}$$

The Stage 1 training is dominated by the Lang-8 dataset (88.97% of $\mathcal{D}_{\text{S1}}$), which consists of text written by English-language learners and includes simple sentences and some inconsistencies in the target corrections. The W&I+LOCNESS corpus was presented for the BEA19 GEC Task [26] and consists of high-quality, manually annotated corrections from the Write & Improve platform and the LOCNESS corpus of native and near-native English essays. Fine-tuning on this dataset in a second stage allows the GEC model to learn stronger and better English GEC corrective edits.

This two-stage training approach is applied to all four models in $\mathcal{M}$. For the edit-specific models (M_R, M_M, M_U), the edit-type dataset filtering described in Section 4.3.1 (above) is performed for both $\mathcal{D}_{\text{S1}}$ and $\mathcal{D}_{\text{S2}}$. The All-edits model M_{All} is trained on the whole dataset in both stages, containing all edit types.

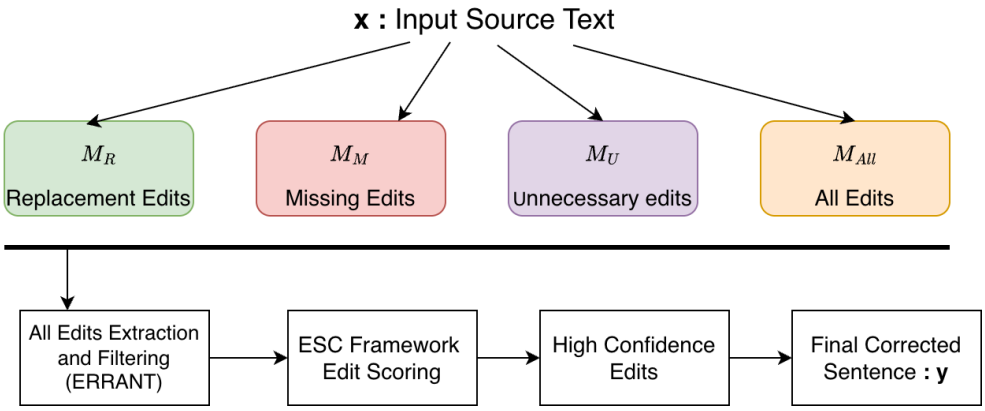

Figure 4.4: Overview of the Edit-based Ensemble approach using the ESC framework. Multiple edit-type specific models (M_i) generate correction predictions, which are decomposed into ERRANT-based edits. These edits are evaluated using the ESC regression model. The chosen edits are applied to the source text to get the final correction prediction.

Paper Baseline Model The GEC Model - M_{All} obtained after Stage 1 training corresponds to the GEC Baseline model reported by Katsumata and Komachi [128]. I refer to this as the - Paper Baseline Model and include it in the result tables (Table 4.1) for comparison.

4.3.2 Ensembling via Edit-based System Combination (ESC)

I use the ESC framework for Grammatical Error Correction (GEC) to score and combine the four edit-type-focused predictions from the component models into a final prediction. The ESC ensembler by Qorib, Na, and Ng [231] uses a logistic regression model trained on individual GEC edit features extracted and labelled using the ERRANT toolkit. Compared with previous GEC ensembling approaches, the ESC model can easily assess, score and combine effective edits made by multiple base systems into the final GEC prediction in a single pass. I describe how the ESC model ensembling approach was used in this work in Figure 4.4. This gives us a straightforward and fast approach to evaluate and ensemble the edits proposed by the four base - Substitution/Replacement (M_R), Insertion/Missing (M_M), Deletion/Unnecessary (M_U) and Connected/All (M_{All}) Correction Models (Section 4.3.1). I explain the ESC System Combination method in detail below.

Ensemble Task Formulation Let (x, y) denote the source and target text, respectively. The GEC task can be considered as a set of corrective edit operations $\mathcal{E}$ that transform text from x to y. Each individual edit $e \in \mathcal{E}$ is represented as a tuple:

$$e = (\text{start},\ \text{end},\ \text{correction},\ \text{type})$$

where start and end are token index of the edit in x, correction is the replacement string (empty for deletion (U) type edits), and type is the ERRANT error category label assigned to the edit. The edit type comprises the syntactic type prefixed by its operation - `R:` for substitution, `M:` for insertion, and `U:` for deletion. Considering all possible edit-type labels present in the ERRANT results yields an edit-vocabulary feature set of $|T| = 54$ error categories.

Each base component model $\mathcal{M} = \{M_{\text{All}}, M_R, M_M, M_U\}$ produces a prediction $\hat{y}^{(k)} = M_k(x)$. The ERRANT tagging is applied to each source-prediction pair $(x, \hat{y}^{(k)})$ to extract the corresponding edit set $\mathcal{E}^{(k)}$. The candidate set of possible edits to evaluate - $\mathcal{E}$ is the unique edits in the union of edit sets across all model predictions:

$$\mathcal{E} = \left\{ \bigcup_{k=1}^{|\mathcal{M}|} \mathcal{E}^{(k)} \right\}$$

Edits in $\mathcal{E}$ proposed by multiple models are represented in the feature vector described below.

Edits Feature Representation The ESC model generates a feature representation for each individual edit $e \in \mathcal{E}$ that represents *which* base component model predicted it. The feature vector for each edit $\boldsymbol{x}^e$ is the combination of the one-hot edit representations of that edit for each prediction - k per base model.

$$\boldsymbol{x}^e = [\mathbf{x}_1\ ;\ \mathbf{x}_2\ ;\ ...\ ;\ \mathbf{x}_k] \in \mathbb{R}^{k|T|} \tag{4.1}$$

Since $\mathcal{M} = \{M_{\text{All}}, M_{\text{R}}, M_{\text{M}}, M_{\text{U}}\}$, I have $k = 4$ component models and predictions. The one-hot representation denotes which ERRANT edit e is out of the possible $|T| = 54$ error categories, a 54-dimension vector.

$$\boldsymbol{x}^e = [\mathbf{x}_R\ ;\ \mathbf{x}_M\ ;\ \mathbf{x}_U\ ;\ \mathbf{x}_{All}] \in \mathbb{R}^{216} \tag{4.2}$$

If model M_i proposed a particular edit e with ERRANT type t, then $\mathbf{x}_i$ has a 1 in the dimension corresponding to t and 0 elsewhere (one-hot vector). If M_i does not propose edit e at all, $\mathbf{x}_i$ is 0 for all dimensions. Since $|T| = 54$ error categories and $k = 4$ models, x^e has dimensionality $k|T| = 216$. This representation shows which base model components agrees on an edit.

Logistic Regression Estimation Model A linear neural model is trained on the feature vectors x^e extracted from a GEC development dataset. For each candidate edit $e \in \mathcal{E}$ with feature vector $\boldsymbol{x}^e$, the ESC model estimates the probability that e is an edit that should be applied to the source text:

$$\text{score}(e) = \sigma(\mathbf{w} \cdot \boldsymbol{x}_e + b) \tag{4.3}$$

where σ denotes the sigmoid scoring function, $\mathbf{w} \in \mathbb{R}^{k|T|}$ is the logistic regression weight vector, and b is the bias term. The sigmoid is used to get a confidence score for all candidate edits, which is used to threshold low-quality edits and to resolve overlapping edits by greedy selection, as described below. The model is trained using the binary cross-entropy loss using stochastic gradient descent (SGD) optimization.

Ensembling Edit Selection The trained ESC model is used during inference to evaluate the edits made by all edit-specific base models. The inference procedure is explained in Algorithm 5.

The ESC regression model estimates the confidence score $\text{score}(e) \in [0, 1]$ for each candidate edit in E^*. A threshold value $\tau = 0.5$ is set for the regression estimated score. Edits with $\text{score}(e) < \tau$ are removed. Since the four base model components can predict edits that overlap, a greedy selection step is performed after threshold-filtering to select the final edits to apply to the source text.

The filtered edits are sorted in descending order of their score, and edits are chosen from the highest to lowest score, skipping edits with index spans that have already been selected (overlapping edits).

4.3.3 LLM-Based Edit-Specific Ensembling

The ESC GEC framework ensembles correction edits based on the regression model scores. The model is trained on edit type, syntax structure, and system agreement features. However, it does not consider the actual *semantic validity* or contextual significance of the

Algorithm 5 Edit-Type Ensemble using ESC Inference

Require: Source sentence x, model set $\mathcal{M}$, regression model f_θ, threshold $\tau = 0.5$
Ensure: Final corrected sentence $\hat{y}^*$

$E \leftarrow \emptyset$
for each model $M_k \in \mathcal{M}$ **do**
  $\hat{y}^{(k)} \leftarrow M_k(x)$
  $E^{(k)} \leftarrow \text{ExtractEdits}(x, \hat{y}^{(k)})$
  $E \leftarrow E \cup E^{(k)}$
end for
$E^* \leftarrow \text{Unique}(E)$
for each edit $e \in E^*$ **do**
  $\boldsymbol{\phi}_e \leftarrow \text{ExtractFeatures}(e, \mathcal{M})$ {one-hot encoding over $k|T|$ dimensions (Eq. 4.2)}
  $\text{score}(e) \leftarrow f_\theta(\boldsymbol{\phi}_e)$ {$\text{score}(e) \in [0, 1]$ (Eq. 4.3)}
end for
Sort E^* in descending order of $\text{score}(e)$
$E_{\text{sel}} \leftarrow \emptyset$
for each edit $e \in E^*$ **do**
  if $\text{score}(e) > \tau$ **then**
    if e does not overlap with any edit in E_{sel} **then**
      $E_{\text{sel}} \leftarrow E_{\text{sel}} \cup \{e\}$
    end if
  end if
end for
$\hat{y}^* \leftarrow \text{Apply}(x, E_{\text{sel}})$
return $\hat{y}^*$

candidate edits. The syntactic features of the ESC model (ERRANT tags) may not fully capture whether an edit improves meaning, fluency, or domain-specific correctness.

To address this limitation, I propose an alternative ensembling method using large language models (LLMs) as semantic evaluators and generators. These methods focus on the *type* (Insertion, Deletion, and Substitution) and *content* of the edit to estimate and generate the final correction.

I introduce two LLM-based ensembling approaches for the text corrections made by the individual edit-specific GEC Base Models (Section 4.3.1):

- **Prompt-based Edit Type Ensemble**: Utilizing an LLM to generate the final text correction, considering the individual edit-type model predictions through prompting.

- **Edit Quality Classification Ensemble**: Fine-grained scoring and selection of edits using LLM-based quality classification.

Both methods utilize the predictions from the edit-specialized models:

$$\mathcal{M} = \{M_{\text{All}}, M_{\text{R}}, M_{\text{M}}, M_{\text{U}}\}$$

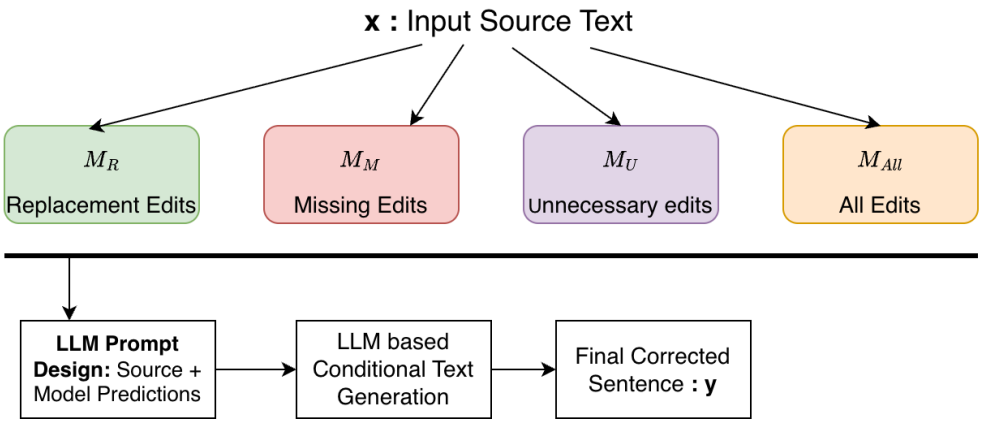

Figure 4.5: Overview of the LLM-based GEC Ensemble approach. The edit-type model (M_i) predictions are used to design an LLM instruction prompt. The LLM generates the final correction conditioned on the prompt.

Prompt-based Edit Type Ensemble

In this approach, GEC ensembling is formulated as a conditional text generation task through LLM prompting. Given a source text and multiple corrected hypotheses from edit-specialized models, an LLM is prompted to generate the final text correction. Figure 4.5 describes the LLM generation-based GEC Ensembling approach.

Formally, let:

$$\hat{y}^{(k)} = M_k(x), \quad \forall M_k \in \mathcal{M}$$

The LLM takes as input:

$$\mathcal{I} = \{x, \hat{y}^{(all)}, \hat{y}^{(ins)}, \hat{y}^{(del)}, \hat{y}^{(sub)}\}$$

and produces:

$$\hat{y}^* = \text{LLM}(\mathcal{I})$$

Two prompting strategies are considered:

- **Standard Prompt**: The LLM is instructed to generate the final correction text by considering all four model predictions, allowing it to freely modify or introduce extra edits.
- **No-Extra (NE) Prompt**: The LLM is instructed to generate the final correction by only considering the edits present in the provided model predictions. The model is explicitly instructed not to introduce any new corrections beyond those suggested by the GEC edit-specialized models ($\mathcal{M}$).

This allows the LLM to implicitly perform edit evaluation and selection for GEC in a single step, utilizing edit content and context information. The inference process for the LLM Prompt-based Edit Type Ensemble approach is explained in Algorithm 6.

Algorithm 6 Inference process for Prompt-based Edit Type Ensemble

Require: Source sentence x, model set $\mathcal{M}$, LLM
Ensure: Final corrected sentence $\hat{y}^*$
1: **for** each model $M_k \in \mathcal{M}$ **do**
2: $\quad \hat{y}^{(k)} \leftarrow M_k(x)$
3: **end for**
4: Construct LLM input prompt $\mathcal{I} = \{x, \hat{y}^{(All)}, \hat{y}^{(R)}, \hat{y}^{(M)}, \hat{y}^{(U)}\}$
5: $\hat{y}^* \leftarrow \text{LLM}(\mathcal{I})$
6: **return** $\hat{y}^*$

Edit Quality Classification Ensemble

Even though the prompt-based approach instructs the LLM to generate the target prediction conditioned on the four edit-type predictions, there is no guarantee that the LLM only uses the edits proposed by the four GEC Models ($\mathcal{M}$) to generate the target correction text. To provide a more structured alternative to direct generation, I introduce an LLM-based edit evaluation framework that somewhat matches the ESC-ensembling pipeline (Section 4.3.2) while adding semantic evaluation at the word level. Figure 4.6 describes the LLM edit-quality classification GEC Ensembling approach.

Given model predictions, I first extract candidate edits:

$$E^{(k)} = \text{ExtractEdits}(x, \hat{y}^{(k)})$$

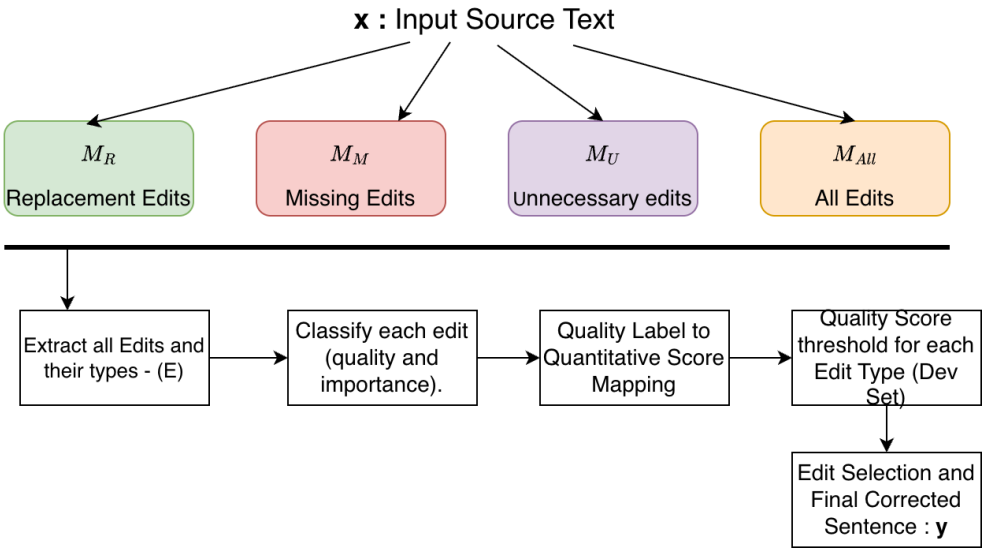

Figure 4.6: Overview of the LLM-edit quality classification GEC Ensemble approach. All candidate edits from all models (M_i) predictions are extracted and evaluated using LLMs on their quality and importance.

We define the global candidate set:

$$E = \bigcup_{k \in \mathcal{M}} E^{(k)}, \quad E^* = \text{Unique}(E)$$

Each edit $e \in E^*$ is then evaluated independently by the LLM using a classification prompt. The model assigns a label:

$$\ell_e \in \mathcal{L}$$

where:

$$\mathcal{L} = \{\text{harmful}, \text{incorrect}, \text{unnecessary}, \text{minor_improvement}, \text{useful}, \text{important}, \text{essential}\}$$

Each label is mapped to a mapped discrete score, $\text{score}(e) \in [0, 1]$:

Label	Score
harmful	0.00
incorrect	0.15
unnecessary	0.35
minor_improvement	0.55
useful	0.70
important	0.85
essential	0.97

Edits whose LLM output contains no valid label are assigned a score of 0.

Selected edits are determined based on their score threshold specific to the edit type $t(e) \in \{R, M, U\}$ (replacement, missing/insertion, unnecessary/deletion):

$$E_{\text{sel}} = \{e \in E^* \mid \text{score}(e) \geq \tau_{t(e)}\}$$

The thresholds (τ_R, τ_M, τ_U) are tuned on the BEA-19 development set (B19D). Each threshold value is chosen independently from $\{0.0, 0.1, 0.3, 0.5, 0.6, 0.8, 0.9\}$, giving $7^3 = 343$ combinations. For each combination, the selected edits are compared against the gold edits (B19D-Reference) by exact edit match, and the combination with the highest corpus-level $F_{0.5}$ is kept. The sweep is performed for each ensemble configuration and LLM, and the best thresholds are then used unchanged on the test sets (B19T, C14T). Because the label scores are discrete, each threshold acts as a cutoff value; for example, $\tau = 0.5$ retains edits labelled *minor_improvement* or higher. The selected thresholds are reported in Appendix D.4. The threshold sweep results showed that the replacement edits required at least the *minor_improvement* ($\tau_R = 0.5$) label for selection, and the insertion and deletion-based edits were minimally filtered. DeepSeek's labels didn't improve F0.5 at any cutoff, so all its edits were kept.

The final sentence is obtained by applying the selected edits:

$$\hat{y}^* = \text{Apply}(x, E_{\text{sel}})$$

The inference process for the LLM based edit quality estimation ensembling approach is described in Algorithm 7.

Algorithm 7 Edit Quality Classification Ensemble

Require: Source sentence x, model set $\mathcal{M}$, LLM, thresholds $\{\tau_R, \tau_M, \tau_U\}$
Ensure: Final corrected sentence $\hat{y}^*$

$E \leftarrow \emptyset$
for each model $M_k \in \mathcal{M}$ **do**
  $\hat{y}^{(k)} \leftarrow M_k(x)$
  $E^{(k)} \leftarrow \text{ExtractEdits}(x, \hat{y}^{(k)})$
  $E \leftarrow E \cup E^{(k)}$
end for
$E^* \leftarrow \text{Unique}(E)$
for each edit $e \in E^*$ **do**
  Construct prompt with (x, e)
  $\ell_e \leftarrow \text{LLM}(x, e)$
  $\text{score}(e) \leftarrow \text{LabelMap}(\ell_e)$
end for
$E_{\text{sel}} \leftarrow \emptyset$
for each edit $e \in E^*$ in descending order of $\text{score}(e)$ **do**
  $\tau \leftarrow \tau_{t(e)}$ {threshold based on edit type}
  if $\text{score}(e) \geq \tau$ AND e does not overlap with E_{sel} **then**
    $E_{\text{sel}} \leftarrow E_{\text{sel}} \cup \{e\}$
  end if
end for
$\hat{y}^* \leftarrow \text{Apply}(x, E_{\text{sel}})$
return $\hat{y}^*$

4.4 Experiments

4.4.1 Experimental Setup

Datasets For GEC, I train all component models on the standard BEA-2019 shared task training corpus [26], comprising of the FCE, Lang-8, NUCLE, and W&I+LOCNESS dataset. The BEA-2019 development set is used for ESC combiner training and LLM-Edit Evaluation threshold hyperparameters selection. Evaluation is performed on the BEA-2019 test set using the ERRANT scorer and on the CoNLL-2014 test set using both the M^2 [51] and ERRANT scorer (ERRANT provides edit-type breakdown performance as well).

Base Model Architecture We use the two BART [158] model variants — BART-Base and BART-Large — for all four edit-specific models. These models were also used as a strong and simple GEC baseline by Katsumata and Komachi [128]. This allows me to evaluate the edit-specific ensembling approach since I use the same training dataset and model as the baseline. Each variant is fine-tuned from the same pre-trained checkpoint [1]. The edit-specific datasets (D_R, D_M, D_U, D_{All}) described in Section 4.3.1 are used to train the M_R, M_M, M_U, and M_{All} models separately with identical hyperparameter settings.

LLM Models For the LLM ensembling experiments, I evaluate the approach using four open-weight instruction-tuned LLMs: Qwen2.5-7B [233], LLaMA3.1-8B [173], DeepSeek-R1-8B [57], and Mistral-7B [117] (Appendix D.1). Each model is prompted with the LLM ensembling task instruction and GEC source text (Appendix D.2). Relevant information from the four edit-specialized model predictions was also added to the prompt instructions. For the Edit Quality Estimation task, I create a seven-class label scheme (*harmful, incorrect, unnecessary, minor_improvement, useful, important, essential*) assigning each label to a continuous score in $[0, 1]$. Apart from using the LLM for edit ensembling, I also present the LLM-based GEC task baseline using zero-shot and few-shot setting in Appendix D.3.

Correction Evaluation GEC text performance is measured using the precision (P), recall (R), and $F_{0.5}$ scores following standard practice [200]. Fine-grained edit-specific evaluation is performed using ERRANT's edit-type breakdown into R, M, and U correction categories.

Multiple-Run Evaluation and Statistical Significance To account for the random nature of model training, each experiment was trained and evaluated using six seed values (Appendix B). The model scores presented in the results table (precision, recall and $F_{0.5}$) represent the average performance over these six runs. This applies to the Curriculum Learning (All) baseline, the three edit-specific base models (M_R, M_M, M_U), the ESC ensemble, and the LLM-based ensembling approaches, where the six scores were obtained by running inference on the six correction models obtained for each category using the six seed values.

The Paper Baseline [128] results are averaged over the six inference results after Stage 1 training of the M_{All} model (Section 4.3.1). Statistical significance was calculated using a one-tailed Wilcoxon signed-rank test on the paired results from the six seed runs, with the

[1] https://huggingface.co/facebook/models?search=bart

significance level set to $\alpha = 0.05$. Since a higher $F_{0.5}$ is better, the alternative hypothesis was that the ensembling approach improves the Curriculum Learning (All) baseline $F_{0.5}$. A statistically significant $F_{0.5}$ improvement over the Curriculum Learning (All) baseline ($p < 0.05$) is denoted with an upward arrow (↑) next to the $F_{0.5}$ value. Further details on the six runs are provided in Appendix B.

4.4.2 Results

I evaluate the three Edit Type Ensemble (ETE) approaches on two standard GEC Test Datasets the BEA-2019 test set scored using ERRANT [27] toolkit and the CoNLL-2014 test set scored using the M^2 scorer [51]. The first approach is the ESC ensemble (Section 4.3.2), which combines the four base edit-specific BART models using the regression-based edit combiner. The second and third are the LLM-based ensembling methods described in Section 4.3.3: the Instruction prompt generation method (Standard and No-Extra(NE)) and the Edit Quality Classification method. All approaches are evaluated for the BART-Base and BART-Large model architectures. Results are reported in Table 4.1.

I include the results of the BART Baseline Model [128] - Paper Baseline, without curriculum learning or edit-type ensembling for comparison with all my approaches. The Curriculum Learning (All) Baseline model, which uses the two-stage training procedure on the full edit set (Section 4.3.1), serves as the new stronger baseline for evaluating the improvement from Edit ensembling methods. I also include the performance of the other three Edit-specific Base Models - R, M, and U.

ESC ensemble outperforms all individual models. As shown in Table 4.1, the R-M-U-All ensemble achieves the highest $F_{0.5}$ for both BART-Base and BART-Large on BEA-2019, and performs better than every individual model on both benchmarks. On CoNLL-2014 (M^2), some LLM ensembles (Section 4.3.3) score higher: LLaMA, Qwen-NE and LLaMA-NE with BART-Base, and LLaMA-NE with BART-Large. For BART-Large on BEA-2019, the ensemble reaches 74.70 $F_{0.5}$, an improvement of **+6.62** points over the Curriculum Learning (All) baseline (68.08). For CoNLL-2014, the BART-Large ensemble gets $F_{0.5} = 64.03$ using M^2 evaluation, outperforming the Curriculum Learning baseline by **+1.46** points. Primarily, the ESC ensemble result improvement is because of a large improvement in precision scores. For BART-Large on BEA-2019, the ensemble precision reaches 82.75 compared to 69.46 for the Curriculum Learning baseline. This supports the hypothesis of this chapter: the ESC edit ensembling helps select a better, higher-confidence and higher-quality set of correction edits, which is rewarded by the GEC $F_{0.5}$ metric. I

Model	Strategy	BEA19 Test (ERRANT)			CoNLL14 Test (M2)		
		P	R	$F_{0.5}$	P	R	$F_{0.5}$
BART-Base	Paper Baseline	66.66	50.40	62.62	68.73	38.42	59.36
	Curriculum Learning (All)	70.26	56.62	67.03	68.84	41.43	60.80
	Substitution Edits (R)	68.50	36.51	58.28	69.72	27.29	53.18
	Insertion Edits (M)	71.09	22.80	49.94	54.14	09.81	28.44
	Deletion Edits (U)	51.40	10.09	28.25	52.14	08.46	25.64
	R-M-U-All Edits Ensemble	79.71	50.90	**71.60**$^{\uparrow}$	74.75	36.86	62.00$^{\uparrow}$
	LLM-Qwen	52.33	66.94	54.72	59.32	54.61	58.32
	LLM-LLaMa	64.27	63.23	64.05	68.77	46.12	**62.62**$^{\uparrow}$
	LLM-Deepseek	63.13	55.07	61.34	68.22	39.48	59.55
	LLM-Mistral	42.38	65.43	45.59	53.62	53.60	53.62
	LLM-Qwen-NE	63.27	62.98	63.21	67.94	47.47	62.55$^{\uparrow}$
	LLM-LLaMa-NE	65.94	60.61	64.81	69.07	44.30	62.13
	LLM-Deepseek-NE	64.00	55.82	62.18	68.74	39.53	59.89
	LLM-Mistral-NE	57.63	62.43	58.53	66.27	46.32	61.02
	LLM-Qwen-Edit	68.74	52.16	64.63	67.17	37.16	57.83
	LLM-LLaMa-Edit	67.11	56.65	64.72	65.21	40.98	58.32
	LLM-Deepseek-Edit	65.02	58.50	63.61	65.66	44.50	59.96
	LLM-Mistral-Edit	66.49	57.22	64.40	65.74	42.72	59.35
BART-Large	Paper Baseline	71.45	52.60	66.67	71.48	39.76	61.64
	Curriculum Learning (All)	69.46	63.07	68.08	67.42	48.60	62.57
	Substitution Edits (R)	70.82	43.04	62.72	71.41	32.43	57.57
	Insertion Edits (M)	74.61	26.14	54.43	50.47	11.75	30.42
	Deletion Edits (U)	55.03	8.64	26.53	55.43	08.77	26.86
	R-M-U-All Edits Ensemble	82.75	53.78	**74.70**$^{\uparrow}$	75.12	40.27	64.03$^{\uparrow}$
	LLM-Qwen	53.41	69.37	55.98	60.24	56.01	59.35
	LLM-LLaMa	65.08	66.46	65.35	67.92	50.50	63.54$^{\uparrow}$
	LLM-Deepseek	63.79	60.99	63.21	68.33	45.13	61.96
	LLM-Mistral	43.64	67.28	46.94	54.99	56.36	55.26
	LLM-Qwen-NE	63.93	67.04	64.53	67.35	52.13	63.63$^{\uparrow}$
	LLM-LLaMa-NE	66.72	65.29	66.44	69.40	49.82	**64.34**$^{\uparrow}$
	LLM-Deepseek-NE	64.31	61.30	63.68	68.31	45.88	62.23
	LLM-Mistral-NE	59.45	65.73	60.61	66.67	51.33	62.91
	LLM-Qwen-Edit	68.08	60.86	66.50	65.98	46.61	60.92
	LLM-LLaMa-Edit	68.17	61.95	66.83	65.96	46.25	60.78
	LLM-Deepseek-Edit	65.49	64.84	65.36	64.88	51.53	61.69
	LLM-Mistral-Edit	67.76	62.77	66.70	65.30	48.65	61.12

Table 4.1: Performance of the BART-Base and BART-Large GEC models under different training strategies on the BEA19 (ERRANT) and CoNLL14 (M^2) test sets. Each value is averaged over six runs. ↑: statistically significant $F_{0.5}$ improvement over the Curriculum Learning (All) baseline ($p < 0.05$).

also present the CoNLL-2014 scores calculated using the ERRANT scorer in Table 4.2 for completeness. I make use of the ERRANT evaluation to get the edit-specific model performance (explained next), which the M2 scorer does not calculate.

Edit-Type Performance. Table 4.3 compares the performance of the ESC ensemble and the Curriculum Learning (All) baseline for each edit type (R, M and U). On BEA-2019,

Model	Strategy	CoNLL14 Test (ERRANT)		
		P	R	$F_{0.5}$
BART-Base	Paper Baseline	65.29	37.55	56.88
	Curriculum Learning (All)	65.00	40.14	57.83
	Substitution Edits (R)	66.65	26.36	51.04
	Insertion Edits (M)	52.28	09.64	27.75
	Deletion Edits (U)	47.90	07.68	23.38
	R-M-U-All Edits Ensemble	73.20	35.49	**60.37**$^{\uparrow}$
BART-Large	Paper Baseline	67.67	38.77	58.89
	Curriculum Learning (All)	62.03	46.95	58.29
	Substitution Edits (R)	66.58	31.51	54.45
	Insertion Edits (M)	46.26	11.89	29.31
	Deletion Edits (U)	48.00	08.17	24.30
	R-M-U-All Edits Ensemble	73.44	38.70	**62.26**$^{\uparrow}$

Table 4.2: Performance of the BART-based GEC models on the CoNLL14 test set evaluated using ERRANT, reported in addition to the M^2 scores (Table 4.1) for the edit-type (R, M, U) analysis. Each value is averaged over six runs. ↑: statistically significant $F_{0.5}$ improvement over the Curriculum Learning (All) baseline ($p < 0.05$).

the ensemble gives a statistically significant $F_{0.5}$ improvement for all three edit types with both BART models. On CoNLL-2014, the improvement is significant for substitution (R) and insertion (M) edits, but the ensemble scores lower than the baseline on deletion (U) edits (62.40 vs. 62.84 for BART-Base and 61.21 vs. 63.28 for BART-Large). This is likely because the deletion model (M_U) is the weakest edit-specific model, so the ensemble has less inter-model agreement for deletion edits and mainly filters the (M_{All}) deletion edits, which reduces recall more than it improves precision.

LLM ensembling underperforms ESC. Despite the semantic reasoning capabilities of LLMs, the prompt-based approaches achieve lower $F_{0.5}$ than the ESC ensemble in almost all settings. The best prompt-based result on BEA-2019 (LLaMA-NE with BART-Large: 66.44) falls 8.26 points below the ESC ensemble (74.70). I attribute this to two factors: (1) LLMs without constrained decoding tend to introduce new corrections beyond those proposed by the component models, harming precision; and (2) the ESC ensemble approach makes an explicit selection decision for each edit, learned from system-agreement and edit-type features. However, the prompt-based LLM method regenerates the whole sentence, implicitly choosing what changes to make. The No-Extra (NE) constraint improves $F_{0.5}$ score compared to the standard prompt in most settings (except LLaMA with BART-Base

Model	Strategy	R Correction Edits			M Correction Edits			U Correction Edits		
		P	R	$F_{0.5}$	P	R	$F_{0.5}$	P	R	$F_{0.5}$
BEA19-Test Dataset										
BART-Base	Baseline (CL-All)	67.70	48.65	62.78	75.75	69.42	74.39	69.41	66.79	68.87
	ESC-Ensemble	77.07	45.11	67.51$^{\uparrow}$	87.43	59.60	79.96$^{\uparrow}$	74.42	62.94	71.80$^{\uparrow}$
BART-Large	Baseline (CL-All)	67.27	56.49	64.79	74.57	74.17	74.49	69.19	69.07	69.16
	ESC-Ensemble	81.84	48.32	71.86$^{\uparrow}$	86.63	64.36	81.02$^{\uparrow}$	76.97	57.63	72.13$^{\uparrow}$
CONLL14-Test Dataset										
BART-Base	Baseline (CL-All)	66.10	37.19	57.21	58.15	44.89	54.91	69.09	46.14	62.84
	ESC-Ensemble	75.01	34.19	60.55$^{\uparrow}$	69.19	35.80	58.31$^{\uparrow}$	72.13	40.52	62.40
BART-Large	Baseline (CL-All)	62.26	43.56	57.34	57.15	55.50	56.81	68.22	49.08	63.28
	ESC-Ensemble	76.87	37.62	63.60$^{\uparrow}$	67.90	41.46	60.22$^{\uparrow}$	70.54	40.02	61.21

Table 4.3: Edit-type (R, M, U) Precision, Recall and $F_{0.5}$ on the BEA19 and CoNLL14 test sets, evaluated using ERRANT. Each value is averaged over six runs. ↑: statistically significant $F_{0.5}$ improvement over the Curriculum Learning (All) baseline ($p < 0.05$).

on CoNLL-2014, 62.62 to 62.13). On CoNLL-2014, some prompt-based ensembles give a statistically significant $F_{0.5}$ improvement over the Curriculum Learning (All) baseline: LLaMA and Qwen-NE with both BART models, and LLaMA-NE with BART-Large (Table 4.1). Several of these also score above the ESC ensemble. On BEA-2019, none of the LLM-based approaches improve over the baseline.

LLM-based Edit Quality Classification improves over prompting method on BEA-2019 but not on CoNLL-2014. The LLM-Edit approach, which scores individual edits with the seven-class label scheme, gives the best LLM-based result with BART-Large on BEA-2019 (LLaMA-Edit, 66.83 $F_{0.5}$, 7.87 points below ESC), and with BART-Large, all four LLM-Edit scores perform better than the NE prompt. On CoNLL-2014 (M^2), LLM-Edit is below the NE prompt for every LLM and both BART sizes except DeepSeek with BART-Base (59.96 vs. 59.89). No LLM-Edit configuration improves over the baseline on either test set. Structured edit-level ensembling using LLMs is therefore worth studying further, to get it closer to ESC in performance.

4.5 Parameter-Efficient Edit-Type Models

The Edit Type Ensemble framework uses four base BART models - M_{All}, M_R, M_M and M_U. All four models are separately trained, stored, loaded and used for the inference pass. Since the four models are fine-tuned on the same starting checkpoint and use different edit-specific training data (Section 4.3.1), I substitute them with a single frozen base BART model and four lightweight edit-specific adapters. In this section, I describe the adapter models and compare them with the full fine-tuned models based on correction quality, inference time and storage.

4.5.1 Method

Adapters [107] are small trainable modules inserted into each layer of a pre-trained Transformer model. The adapter weights are updated during training, and the base model weights stay frozen. I use the Pfeiffer adapter configuration [217], where a single bottleneck module (down-projection, non-linearity, up-projection and a residual connection) is added after the feed-forward component of all encoder and decoder layers. The adapter type and size selection is described in Appendix D.5.

A separate adapter is trained for each edit type - A_{All}, A_R, A_M and A_U - on the same frozen BART base model using the two-stage curriculum learning setup as the fully fine-tuned models (Section 4.3.1). For a direct comparison, I also retrain the four fully fine-tuned models under the same setting, which differs from Section 4.3.1 in one aspect - I keep the source-target pairs that contain no edit of the model's type ($y_i^R = x_i$), so that the edit-specific model learns not to make over-corrections. The adapter and the fully fine-tuned models are trained three times (with different seeds). The best checkpoint of each model is selected on the FCE development set, and the results are averaged and compared.

Since the four adapters use the same frozen base model, they can be combined into a single model and run together using the *Parallel* composition block of the Adapters library [218]. The input source batch is replicated at the first adapter layer, and the copies are passed through the edit-specific adapter, so a single beam search call generates all four corrections.

4.5.2 Results

Table 4.4 compares the adapter-based models with the full fine-tuned models after the two-stage training, for all edit types and both BART-based models.

Model	Edit Type - Training	BEA19 Test (ERRANT)			CoNLL14 Test (M2)		
		P	R	$F_{0.5}$	P	R	$F_{0.5}$
BART-Base	All - Full Fine-Tuning	72.46	47.88	65.71	73.35	35.69	60.57
	All - Adapter	74.38	52.45	**68.64**	72.08	38.52	**61.38**
	Substitution (R) - Full Fine-Tuning	75.82	32.40	**59.79**	74.09	22.87	51.17
	Substitution (R) - Adapter	76.06	32.21	59.78	73.43	23.33	**51.37**
	Insertion (M) - Full Fine-Tuning	77.12	14.38	41.17	71.38	5.09	19.79
	Insertion (M) - Adapter	86.25	19.21	**50.79**	71.41	7.09	**25.37**
	Deletion (U) - Full Fine-Tuning	59.17	8.47	26.93	56.84	8.22	**26.03**
	Deletion (U) - Adapter	63.15	8.28	**27.15**	54.13	8.10	25.33
BART-Large	All - Full Fine-Tuning	75.52	60.74	**72.01**	72.43	44.95	**64.54**
	All - Adapter	74.97	58.73	71.03	73.27	43.67	64.52
	Substitution (R) - Full Fine-Tuning	76.48	39.09	64.20	74.19	29.11	56.64
	Substitution (R) - Adapter	76.78	41.91	**65.82**	75.29	31.10	**58.63**
	Insertion (M) - Full Fine-Tuning	86.76	21.88	54.45	68.13	9.43	30.35
	Insertion (M) - Adapter	86.03	23.33	**55.95**	68.78	9.85	**31.31**
	Deletion (U) - Full Fine-Tuning	73.73	8.29	**28.59**	71.98	7.98	27.64
	Deletion (U) - Adapter	75.36	7.96	27.96	67.40	8.90	**29.10**

Table 4.4: Results of the full fine-tuned and adapter-based edit-specific models after the two-stage training, on the BEA19 (ERRANT) and CoNLL14 (M^2) test sets. The values are the average of three runs. The higher $F_{0.5}$ of each pair is presented in bold.

Adapter models can match full fine-tuning performance. For both model sizes and test sets, the adapter models obtained a greater $F_{0.5}$ score compared to the fully fine-tuned models in 11/16 settings, while training only 10% of the fully fine-tuned model parameters. The largest improvements were found for the BART-Base insertion (M) adapter model (+9.62 on BEA-2019 and +5.58 on CoNLL-2014) improving both precision and recall scores. The BART-Base All adapter also performs better on both test sets (+2.93 and +0.81). For the BART-Large model, the edit-specific adapter models get better scores in a few settings. The substitution (R) adapter shows the best improvement (+1.62 and +1.99). There are many instances where the difference between full fine-tuning and adapter model scores is within one $F_{0.5}$ point. I consider this to be similar and, as a result, the adapter-based training for GEC is comparable with full fine-tuning.

4.5.3 Inference Cost

I compare the inference time for the four fully fine-tuned and the adapter models on a single NVIDIA Tesla P100 GPU (12 GB), using a beam size of 5. Two settings are evaluated: (1) correcting one sentence at a time (batch size 1, streaming approach), which corresponds to an interactive correction tool, and (2) bulk correction with the largest batch size that fits in GPU memory (BART-Base - 64, BART-Large - 16). The four fully fine-tuned models

Model	Setup	Storage (GB)	Inference time (s)	
			Per sentence	Full set
BART-Base	4 full models, sequential	2.23	1.21	173.5
	Base + 4 adapters, sequential	0.78	1.77	194.0
	Base + 4 adapters, parallel	**0.78** (2.8×)	**0.69** (1.76×)	162.0–179.5 (≈ 1×)
BART-Large	4 full models, sequential	6.50	2.13	**615.1**
	Base + 4 adapters, sequential	2.43	3.47	954.0
	Base + 4 adapters, parallel	**2.43** (2.7×)	**1.45** (1.47×)	918.4–1229.3 (1.5–2.0× slower)

Table 4.5: Storage and inference time for generating all four corrections (All, R, M, U) on a single NVIDIA Tesla P100 GPU with beam size 5. *Per sentence* is the time (batch size 1) for all four streaming corrections; *Full set* is the total time to correct all FCE-Dev 2,191 sentences. The factor value in parentheses is the size saving/time speed-up compared to the four fully fine-tuned models.

are run in succession on the same GPU, and the total time includes the time to load the model. Table 4.5 reports the overall results.

Parallel adapter model is faster for single-sentence correction. When correcting one sentence at a time in a streaming setup, the parallel adapter composition generates all four corrections 1.76× faster than the four fully fine-tuned models for BART-Base (0.69 vs. 1.21 seconds per sentence) and 1.47× faster for BART-Large (1.45 vs. 2.13 seconds). The speed-up is less than 4×, since the parallel inference calculates the beam search for all four adapters and runs until the longest of the four corrections is complete. Running the adapters one after another is slower than the full fine-tuned models (in succession), because the adapter module adds computation to every forward pass. The net speed-up is obtained from the parallel composition and not just the adapter module.

Bulk correction is expensive. Performing bulk correction with larger batches makes the GPU fully utilized by a single model, so the parallel adapters are not able to reduce the inference time. For BART-Base the inference time is similar to the four fully fine-tuned models (≈ 1×), and for BART-Large the parallel adapters take 1.5-2.0× longer, since the BART-Large adapter adds a 1024-dimension bottleneck component to every layer.

Lower storage and memory cost. The adapter-based models require 2.8× (BART-Base) and 2.7× (BART-Large) less storage compared to the four fully fine-tuned models, and only one model has to be kept in memory. The four fully fine-tuned BART-Large models require 6.5 GB, more than half of the 12 GB GPU memory, before any inference steps. Adding a new edit-specific model requires training and saving a new adapter, which is efficient.

4.6 Discussion

The results obtained show that utilizing the existing training data better using Curriculum Learning - Stage Training and better task knowledge utilization - Edit Type for GEC can help improve baseline GEC performance. The gains from edit-specific ensembling are clear and consistent: the ESC ensemble outperforms the M_{All} baseline by up to 6.6 $F_{0.5}$ points (BART-Large). The edit-type analysis indicates that the ESC system combination at the edit-level can help pick high confidence edits that help the GEC correction be more precise, with the exception of deletion edits on CoNLL-2014.

The LLM based ensembling results are interesting. Although it gives lower $F_{0.5}$ than the ESC combiner in most settings, every LLM setting has higher recall scores than the ESC ensemble (and 32 of the 48 configurations have higher recall than the Curriculum Learning (All) baseline). The LLMs consider and accept more of the candidate corrections than ESC, at the cost of precision.

The adapter-module experiments (Section 4.5) show that the four edit-specific models show similar performance when fully fine-tuned or trained with the PEFT adapter. The frozen BART-Base model with four edit-specific adapters matches (within one $F_{0.5}$ point) or exceeds the full fine-tuning for all edit types, and generates all four corrections in one pass. This makes the framework cheaper to deploy, which is the production setting that motivates this chapter: one model in memory, 2.8× less storage, and faster single-sentence correction. The ESC ensembling step is the same for both the fully fine-tuned and the adapter predictions, adding the same storage and inference time in both settings, so I compare only the generation of the four edit-specific models. Since the adapter models perform similarly to the fully fine-tuned models, I expect similar ensemble performance, although this is not evaluated in this work.

4.7 Summary and Future Work

In this chapter, I introduce the Edit Type Ensemble (ETE) framework, which decomposes the GEC task into three edit-type-specific correction sub-tasks and combines the edit-specific predictions using different combination approaches. Four sequence-to-sequence models were trained — M_{All}, M_R, M_M, and M_U — and their predictions were combined via a logistic regression classifier trained on ERRANT edit-type features as well as LLM-based edit-conditional correction generation and LLM edit-quality classification.

The ESC ensemble consistently outperforms all individual edit-specific and GEC Baseline models on both the BEA-2019 and CoNLL-2014 Test datasets. Two different LLM-based ensembling approaches - prompt-based generation and edit quality classification - were also introduced and evaluated. Even though neither matched the precision scores of the ESC approach, both have consistently higher recall than the ESC ensemble. This suggests that LLM-based ensembling for GEC is worth exploring further, since the prompt-based approach requires no annotated data for training or tuning, and no ERRANT-tagged edit feature labels, although no LLM-based ensemble improves over the baseline on BEA-2019.

I also showed that the four edit-specific models can be replaced by a single frozen BART model with four edit-specific adapters. On BART-Base, the adapters match (within one $F_{0.5}$ point) or improve on full fine-tuning for all edit types while training only 10% of the parameters, and the parallel adapters generate all four corrections 1.76× faster for single-sentence correction with 2.8× less storage. On BART-Large, the edit-specific adapters are similar to full fine-tuning, with a difference within two $F_{0.5}$ points.

There are a few directions that can be explored after this work. First, the ERRANT toolkit provides 54 error type categories, and the per-type results in Table 4.3 suggest that edit specialization beyond the three atomic operations may be beneficial for frequent error categories such as punctuation and determiners. Second, the ESC and LLM ensembling approaches show complementary results - ESC produces precise, high-confidence edits while LLM ensembling has consistently higher recall - so a joint approach might be able to leverage the strengths of both methods.

Chapter 5

Temporal Post-Editing Effort for GEC Evaluation

5.1 Introduction

Text editing can involve several iterations of revision. Incorporating an efficient Grammatical Error Correction (GEC) tool in the initial correction round can significantly impact the further human editing effort and the final text quality. There has been a lot of work on building automated GEC tools that can improve the structure and fluency of text while also correcting language errors [29] (Section 2.5). Since GEC tool-assisted text editing is an iterative process, an editor can make post-edits to the tool output to obtain the closest targeted correction. Estimating the post-editing (PE) effort required to reach this targeted correction can be used as a quality evaluation for the GEC tool.

In the previous chapter, I improved the baseline GEC model performance by using an ensemble of models that focus on different edit types: substitution, insertion, and deletion (Chapter 4). Since different edit types and categories can have an impact on the editing effort, this chapter focuses on evaluating GEC tools based on the temporal effort of the post-editing corrections. I introduce Post-Editing Effort in Time (PEET) as a human-centric direction for measuring GEC tool usability.

Human-in-the-loop PE effort was introduced and explored extensively for Machine Translation (MT) systems [138]. PE effort is studied across three levels [142]: *technical* effort, which is the number of edits; *cognitive* effort, which denotes the psychological assessment required to identify and correct the errors; and *temporal* effort, which is the

total time taken to evaluate and perform post-edits (it includes both technical and cognitive effort). Ye et al. [328] and Tezcan, Hoste, and Macken [287] have explored estimating the MT PE time based on edit features. Technical PE effort has also been studied in areas like Text Summarization [151], Natural Language Generation [273] and GEC [251, 207].

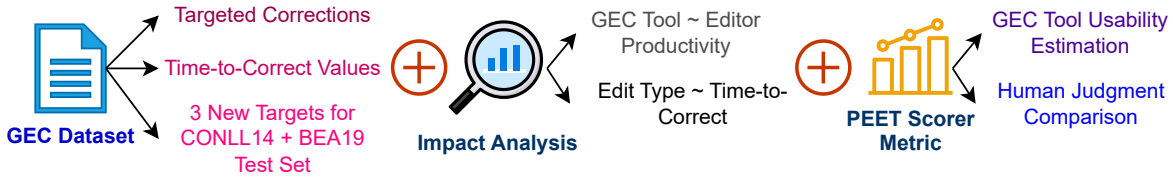

Figure 5.1: The source sentences and the first-pass outputs of two GEC tools are post-edited by professional editors while logging the time-to-correct. The resulting dataset is used to quantify the impact of GEC tools on text editing and to train the PEET Scorer.

The usability of a GEC tool depends inversely on the PE effort required to fix the tool output. To incorporate the human editor effort in text correction, this chapter presents the first work to consider PE effort in Time (PEET) for the quality estimation of a GEC tool (Figure 5.1). I release the first large-scale dataset capturing time-to-correct annotations for two English GEC test sets - BEA-2019 [26] and CoNLL-2014 [200], post-edited in two variations: the source sentence and the outputs using two strong GEC tools - GECToR [203] and GEC-PD [133]. This dataset is used to design PEET Scorer, a new human-centric GEC tool evaluation method to estimate the time-to-correct value for GEC tool predictions, which aligns with human judgment rankings that correlate strongly with the technical post-editing effort. As a result, I propose that the PEET Scorer can be incorporated along with post-editing to evaluate a GEC tool from a human editor's perspective.

The background and related work for the GEC task and the GEC evaluation metrics are presented in the GEC section of the background chapter (Section 2.5). The post-editing effort literature, which is specific to this chapter, is presented in Section 5.3.

5.2 Motivation and Contributions

GEC tool evaluation relies on reference-based metrics and human judgment rankings (Section 2.5.7). Reference-based metrics compare the tool outputs against a set of source-corrected targets (untargeted references). This can ignore the human subjectivity of text correction, and the human judgment rankings used to calibrate the GEC metrics reflect the perceived cognitive effort, which may not correlate with the actual PE effort. For GEC,

PE effort has only been explored at the technical and cognitive level. To the best of my knowledge, temporal PE effort has not been explored in the GEC literature. Temporal effort (time-to-correct value) denotes the overall PE effort, since it includes both technical and cognitive effort.

In this chapter, I focus on the time taken by a human editor to post-edit a GEC tool output. I partner with a professional text-editing company to create a dataset of post-edit corrections and their time-to-correct value. The dataset is utilized to quantify the impact of GEC tools on text editing and to train the PEET Scorer metric. The goal is to design a GEC tool metric that can reflect the usability of the tool from the human editor's perspective.

5.2.1 Contributions

I present the following contributions in this chapter:

- The first large-scale GEC dataset with post-editing time-to-correct annotations, along with three new high-quality human-preference targeted correction sets for the BEA-2019 and CoNLL-2014 test sets - the source sentence correction and the post-edits of two strong GEC tool (GECToR and GEC-PD) outputs.
- A quantification of the editing time saved and the improvement in final correction quality (estimated using GEC metrics) when using GEC tools for first-pass text editing, along with an analysis of edit types that have the greatest impact on the time-to-correct.
- The PEET Scorer, a new evaluation method that can rank GEC tools in terms of time-to-correct, compared against three human judgment rankings of 33 GEC tools, aligning with the rankings that correlate strongly with the technical post-editing correction effort required.

5.2.2 Chapter Structure

I present the post-editing effort background work for Machine Translation and GEC in Section 5.3. Section 5.4 mentions the dataset collection, filtering, correction quality evaluation, and the empirical impact of GEC tools on text editing. Section 5.5 describes the ERRANT-based edit features and the PEET Scorer metric. The scorer results, impact of edit type on post-editing time and comparison with human judgment rankings are reported in Section 5.6. The limitations are discussed in Section 5.7 and the conclusion and future work in Section 5.8.

5.3 Background Work

The background on GEC task, models, dataset, evaluation metric and the ensembling method used by tools are presented in the background chapter (Section 2.5). The background and the post-editing effort literature in Machine Translation and GEC is specific to this chapter only. As a result, I present the post-editing background in this section rather than the background Chapter 2.

In this chapter, I use two GEC tools for first-pass text editing: the GECToR edit tagging model [203] and the GEC-PseudoData (GEC-PD) sequence-to-sequence model [133]. The output from these tools is then further corrected by human editors (post-edited) and the time-to-correct (temporal effort) value tracked. Even though GECToR and GEC-PD are not the most recent GEC tools, they are on par with human-level performance, as observed in Section 5.4.4 (Table 5.3).

5.3.1 Post Editing Effort in Machine Translation

Post Editing Effort (PEE) for Quality Estimation is an actively researched task in Machine Translation (MT). It evaluates the output of an MT system for quality and correctness [259, 270]. Post-editing (PE) the output of an MT system can improve the final translation quality compared to translating the source from scratch, while improving overall editor productivity [221, 93, 87]. I briefly review previous work in MT that explores PEE across three levels (technical, cognitive and temporal effort) [142].

Technical effort has been defined by edit distance metrics like - Translation Edit Rate (TER) and Human TER [266] as well as keystroke and edit operation logging [9, 201, 31]. Cognitive effort has also been studied in terms of edit complexities [283, 139, 222, 49] and human-assessed quality judgment and ranking [272, 271, 137]. Keystroke logs to determine pause information [201, 31], eye gaze tracking and pause fixation [298, 114, 50] and Thinking Aloud Protocol (TAP) [142, 297, 201] have also been proposed as measures of cognitive effort. The work on Temporal Effort in MT estimates the relationship between the time-to-correct and different evaluation metrics [281], source/target translation characteristics [282], and quality estimation [270]. Zaretskaya et al. [334] and Popović et al. [222] study the average temporal effort required for each error type by considering the time-to-correct and frequency of error edits. Finally, Ye et al. [328] and Tezcan, Hoste, and Macken [287] train models to estimate the post-editing time based on PE features.

PE has also been explored in tasks like Text Summarization [151], and Cognitive and Technical PE Effort has been studied for Grammatical Error Correction (GEC) evaluation.

5.3.2 Post Editing Effort in Grammatical Error Correction

I review previous work in GEC that closely relates to post-editing (PE) effort across two levels (cognitive and technical effort). To the best of my knowledge, temporal effort for PE has not been explored for GEC tools.

Cognitive Post Editing Effort

Although cognitive PE effort has not been measured directly for GEC, Human judgment rankings of GEC Tools [90, 136, 197], which are an estimate of perceived cognitive effort, have been used extensively for GEC evaluation metric assessment. Reference-based GEC metrics like ERRANT [27], M^2 [51], GoToScorer [83], and GLEU [198] and reference-less metrics like PT-M^2 [82], Scribendi Score [115], SOME [331] and IMPARA [179] designed to estimate GEC Tool quality are trained and evaluated using the GEC human judgment rankings.

However, perceived cognitive effort does not always agree with the actual PE effort and can be subjective. Sentence correction experiments in GEC have shown poor cognitive agreement between editors. Tetreault, Chodorow, and Madnani [285] and Tetreault and Chodorow [284] asked two native English speakers to insert a preposition into 200 sentences, from which a single preposition was removed, obtaining an agreement score of just 0.7. Rozovskaya and Roth [250] asked three annotators to evaluate and mark 200 sentences for correctness, showing a poor pairwise agreement between them $(0.4, 0.23, 0.16)$. Finally, there has been some work considering the cognitive proficiency of the user interacting with a GEC Tool [195] and the annotators who create the evaluation references of GEC test sets [278, 199].

Surprisingly, none of the GEC metrics described above have considered using targeted references (target obtained after correcting the GEC Tool output) to estimate the tool usability dependent on human PE effort.

Technical Post Editing Effort

To the best of my knowledge, only two prior studies have explored the impact of PE technical effort on GEC evaluation. Rozovskaya and Roth [251] introduced targeted references for English and Russian datasets and Östling et al. [207] utilize PE references to assess Swedish GEC Tools. The studies show that GEC evaluation using untargeted references ignores the human subjectivity involved in text correction. For instance, the SEEDA - human

judgment rankings from Kobayashi, Mita, and Komachi [136] compared the correction outputs of GPT3.5, human editors and various Neural GEC Tools. The GPT-3.5 and human corrections were ranked significantly higher and contained nearly two and three times more edits than other corrections. As a result, these high-quality corrections obtain poor evaluation scores when compared against untargeted references. This inconsistency highlights the importance of PE for GEC Tool evaluation, to capture the true technical effort.

Apart from estimating the PE effort, targeted references can also be used for fine-tuning and aligning Large Language Models (LLMs) with human preferences to generate better outputs [161].

Temporal Post Editing Effort

I introduce the first work to study the Temporal Effort in PE for GEC. Temporal effort described in terms of time-to-correct can efficiently capture the overall PE effort. I present the first large-scale dataset of post-edited corrections along with their temporal effort annotations for two strong GEC tools, GECToR [203] and GEC-PD [133], outputs on two English GEC Test sets - CONLL14 [200] and BEA19 [26]. I also use this dataset to quantify the impact of GEC Tools in Text Editing and the contribution of different edit types to the human post-editing effort. I present PEET Scorer, a regression-based metric, to estimate the time-to-correct scores, which can be incorporated along with post-editing to evaluate the usability of GEC Tools in a human-centred manner.

5.4 Dataset Collection and Processing

An important component in this work is the high-quality dataset of post-edit corrections for GEC, along with their time-to-correct (temporal effort) annotations. I partnered with a professional text-editing company - Scribendi Inc.[1] to collect this data. This section explains my dataset collection, filtering, and quality estimation process (Figure 5.2).

5.4.1 Dataset Source

I use source sentences from two popular English GEC test sets - CONLL14 [200] and BEA19 [26] ($1312 + 4477 = 5789$ sentences). Each sentence was corrected in three variations: the

[1] https://www.scribendi.com/

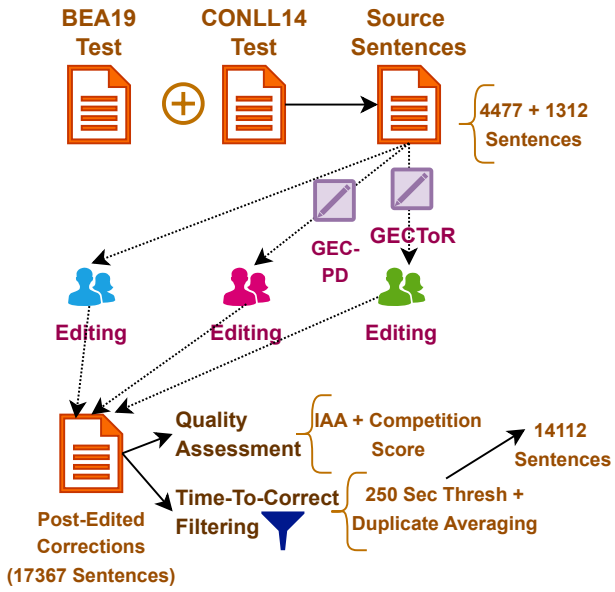

Figure 5.2: Data Collection Setup and Framework for this experiment.

source and post-editing outputs from Two GEC Tools - GECToR [203] and GEC-PD [133] (Section 5.3). Each sentence variation was corrected by one of eight professional text editors, employed by Scribendi Inc. This resulted in a dataset of $5789 * 3 = 17367$ target corrections along with their time-to-correct scores.

5.4.2 Editor Correction Framework

The source sentence and GEC Tool output serve as the basis for further editor correction. This follows the real framework for Text Editing, where a GEC Tool output is evaluated for further correction, compared with the original sentence (Figure 5.3). This similarity also indicates that Post-Editing for GEC Tool evaluation can be incorporated efficiently into the text editing framework.

The editors were given GEC post-editing (PE) instructions (Figure 5.4) and asked to perform minimal edits and avoid rewrites. I used the Qualtrics[2] survey tool to collect PE corrections and enabled the "Timing Question" to log time-to-correct for each source sentence. All other metadata logging was disabled.

[2] https://www.qualtrics.com/

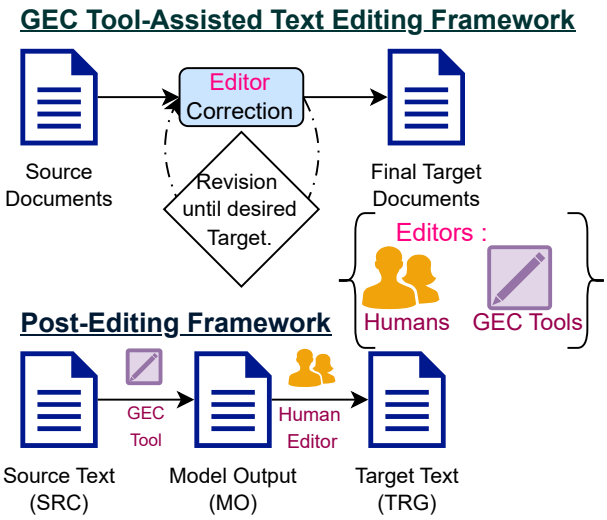

Figure 5.3: Comparison between the GEC-Tool assisted Text Editing Framework and the Post-Editing Framework.

The three variations for each sentence - source, GECToR and GEC-PD output- were given to a different professional editor (in a pool of eight editors) to eliminate any time-to-correct bias. The task of evaluating 17,367 sentences was performed in batches of 50. The editors were shown the source sentence and the first-pass GEC Tool output (Figure 5.5). The final target correction and time-to-correct were logged for each sentence. For source sentence correction, only the original sentence was presented.

5.4.3 Data Filtering

To improve the dataset quality, I performed two stages of data filtering on the three target correction sets for each source (17367 sentences initially). In the first stage, I eliminate outliers based on the logged time-to-correct. Snover et al. [266] showed that editors took between 3-6 minutes for each correction. Considering this and the distribution of the time-to-correct in my dataset, I filter corrections that took more than 250 seconds (17033 sentences remaining). Finally, I merge duplicate corrections from my dataset by averaging the time-to-correct values (14112 sentences dataset). This filtering allows me to retain 81.26% of the dataset that I use as train and test sets (80:20 split) for the Post-Editing Effort in Time (PEET) Scorer.

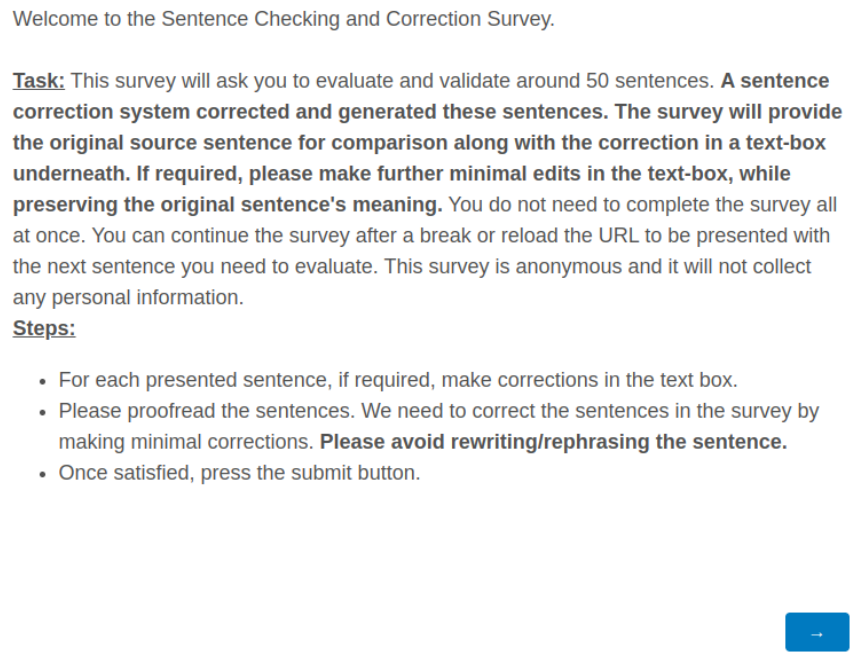

Figure 5.4: Survey instructions for the editor to perform post editing, and obtain target corrections for my dataset.

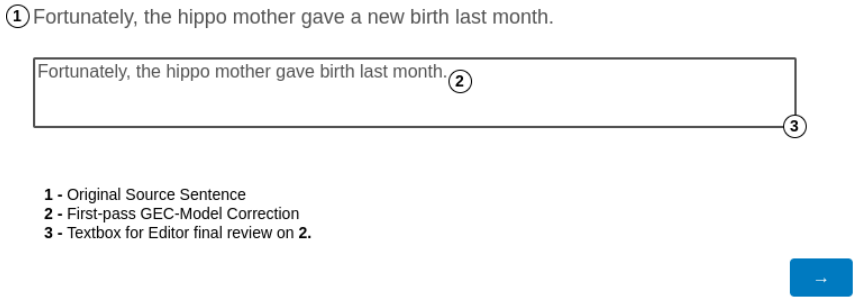

Figure 5.5: Example source sentence and its first-pass edit from the Survey. The editor can make further improvements in the text box. Submitting the final target correction.

5.4.4 Correction Quality

I collect and present three new target corrections for the CONLL14 [200] and BEA19 [26] test datasets. The correction for the source and two post-edited target corrections. I evaluate the quality of the target corrections using the official GEC competition metric and

the Inter Annotator Agreement (IAA) scores. Each target correction set can be divided into CONLL14 and BEA19 corrections. I evaluate the CONLL14 and BEA19 target corrections separately.

Correction	M2 Score	(Precision : Recall)
A1	46.9	44.6 : 59.1
A2	53.0	51.7 : 59.5
A3*	98.6	98.7 : 98.3
A4	55.3	54.9 : 57.0
A5	52.8	51.3 : 59.7
A6	56.4	55.8 : 58.8
A7*	98.6	98.7 : 98.5
A8	53.5	53.8 : 52.6
A9	55.7	55.6 : 56.0
A10	52.8	51.3 : 59.4
c1	50.9	49.0 : 60.4
c2	52.3	50.5 : 61.0
c3	53.7	52.1 : 60.8

Table 5.1: The M2 precision and recall quality score for all Bryant and Ng [28] target correction sets for the official CONLL14 competition task.

Bryant and Ng [28] released 10 additional target corrections for the CONLL14 test dataset. I compare the quality scores of my three corrections with theirs using the official CONLL14 competition - M2 Scorer [200] metric. Table 5.1 shows the M2 scores for all target correction sets - Bryant and Ng [28] corrections $A1 - A10$, and my corrections $c1 - c3$. Corrections A3 and A7 obtain near-perfect quality scores, since they were generated by the two editors who created the official CONLL14 competition target references [28]. Ignoring the two outliers, I observe similar quality scores for my corrections. This indicates that my three CONLL14 Target corrections are of similar quality to the existing high-quality CONLL14 target corrections. Unfortunately, there are no public correction references available for the BEA19 Test set (this work being the first to present three target references), making it hard to compare the quality scores directly.

To overcome this issue, I calculate the quality scores for the three target correction sets and the GEC-Tool first-pass outputs on the official BEA19 and CONLL14 competitions and compare trends between the correction sets. I use the BEA19 competition website scorer[3]

[3]BEA19 GEC competition website - https://codalab.lisn.upsaclay.fr/competitions/4057

Candidate Set	BEA19 Test (P : R : $F_{0.5}$)	CONLL14 Test (P : R : $F_{0.5}$)
Source Sentence	-	-
Source Sentence EC	45.30 : 66.08 : 48.34	49.05 : 60.45 : 50.97
GECToR Output	66.81 : 58.42 : 64.94	63.97 : 45.94 : 59.31
GECToR Output EC	48.24 : 71.38 : 51.59	50.50 : 61.09 : 52.31
GEC-PD Output	66.20 : 61.48 : 65.20	64.06 : 44.92 : 59.03
GEC-PD Output EC	47.33 : 70.54 : 50.66	52.17 : 60.86 : 53.71
GRECO Model Output	86.45 : 63.13 : 80.50	79.36 : 48.69 : 70.48

Table 5.2: Quality Scores of the two GEC Tools output predictions, target Editor Corrections (EC) and the State-of-the-Art GEC Tool - GRECO [232] on the official BEA19 and CONLL14 competition.

Candidate Set	BEA19 Test (P : R : $F_{0.5}$)	CONLL14 Test (P : R : $F_{0.5}$)
GECToR Output	52.59 : 28.59 : 45.03	57.74 : 25.10 : 45.82
GECToR Output EC	45.47 : 47.91 : 45.94	44.31 : 43.53 : 44.15
GEC-PD Output	49.88 : 26.37 : 42.33	56.49 : 23.13 : 43.85
GEC-PD Output EC	45.90 : 48.31 : 46.36	46.14 : 42.64 : 45.39

Table 5.3: Quality Scores of the two GEC Tools output predictions and their final target Editor Corrections (EC) using the BEA19 and CONLL14 - Source Sentence EC as target reference.

to evaluate the performance of BEA19 target corrections. Table 5.2 shows the quality scores for the GECToR and GEC-PD Tool output and the final editor target corrections (EC).

Similar trends are observed between the CONLL14 and BEA19 target correction sets. I observe a large increase in Recall scores for the EC compared to the first-pass GEC Tool output. This indicates the final EC target contains additional post-edit corrections missed by the GEC Tool. The reduction in the precision score for EC is consistent with the 10 CONLL14 target corrections released by Bryant and Ng [28] since post-editing often leads to subjective paraphrasing and rewrite edits, which may not be present in the official competition target reference. The final EC obtained better Recall scores compared to the State-of-the-Art (SOA) GEC Tool - GRECO (as of doing this work) [232] for both datasets. Observing similar quality score trends for the GEC Tool predictions and my target EC across both CONLL14 and BEA19 Test data, consistent with the existing CONLL14 target

corrections, I infer that the three target corrections collected in this work are of high quality. The higher Recall (also compared to the SOA GRECO tool) indicates that the editors make more corrections, including subjective rewrites. The post-edited GEC Tool output EC also obtains better $F_{0.5}$ and Recall scores than the Source Sentence EC for both datasets (Section 5.4.5).

I also use the GEC Inter Annotator Agreement (IAA) framework proposed by Bryant and Ng [28] to compare the target correction sets for both datasets with themselves to ensure better consistency and quality. The IAA framework states that the $F_{0.5}$ multi-reference score, used to evaluate a GEC Tool-vs-human corrections, can similarly evaluate human-vs-human corrections. When comparing multiple annotator corrections, a single correction set can be compared using the rest as references to get quality scores. The final IAA score is calculated as the average of all correction set scores. I use the ERRANT tool [27] to perform the IAA evaluation. I evaluate three target correction sets:

$A = \{A1 - A10\}$ The 10 target corrections for CONLL14 by Bryant and Ng [28].

$C = \{c1, c2, c3\}$ The three CONLL14 target corrections collected by us.

$B = \{b1, b2, b3\}$ The three BEA19 target corrections collected by us.

To compare IAA scores, I conduct a 1-vs-2 target correction set evaluation. For each correction in A, I randomly select two corrections from the remaining nine as the reference. Scores for each correction in B and C are calculated using the remaining two corrections as target references. Table 5.4 shows the average IAA scores for A, B, C correction sets. I observe better Avg-IAA scores for the C and B correction sets collected by us in this work, compared to A. This may be because the corrections collected in this work were made by professional editors from a single company, following the same task guidelines.

To ensure I choose strong GEC Tools (Section 5.3) to obtain first-pass output predictions, I compare the quality of the GEC Tool output and the subsequent human EC. I consider the Source Sentence EC (collected by us) as the target reference for the BEA19 and CONLL14 Test sets. The $F_{0.5}$ quality scores obtained in Table 5.3 show similar performance between the GECToR and GEC-PD Tool prediction output and the subsequent EC because of the variation in Precision and Recall scores. This indicates that GECToR and GEC-PD are strong first-pass GEC Tools.

Human Annotation Set	Reference Set and Size	IAA Score - $F_{0.5}$
$A1$	$\lvert\{RAND(2) \in \{A - A1\}\rvert = 2$	36.21
$A2$	$\lvert\{RAND(2) \in \{A - A2\}\rvert = 2$	45.48
$A3$	$\lvert\{RAND(2) \in \{A - A3\}\rvert = 2$	46.72
$A4$	$\lvert\{RAND(2) \in \{A - A4\}\rvert = 2$	40.54
$A5$	$\lvert\{RAND(2) \in \{A - A5\}\rvert = 2$	46.01
$A6$	$\lvert\{RAND(2) \in \{A - A6\}\rvert = 2$	50.85
$A7$	$\lvert\{RAND(2) \in \{A - A7\}\rvert = 2$	42.72
$A8$	$\lvert\{RAND(2) \in \{A - A8\}\rvert = 2$	49.46
$A9$	$\lvert\{RAND(2) \in \{A - A9\}\rvert = 2$	52.0
$A10$	$\lvert\{RAND(2) \in \{A - A10\}\rvert = 2$	48.57
Avg-IAA $\{A\}$	$\{A\}$, 2	**45.85**
$c1$	$\lvert\{C - c1\}\rvert = 2$	54.11
$c2$	$\lvert\{C - c2\}\rvert = 2$	57.36
$c3$	$\lvert\{C - c3\}\rvert = 2$	59.14
Avg-IAA $\{C\}$	$\{C\}$, 2	**56.87**
$b1$	$\lvert\{B - b1\}\rvert = 2$	57.94
$b2$	$\lvert\{B - b2\}\rvert = 2$	59.39
$b3$	$\lvert\{B - b3\}\rvert = 2$	59.81
Avg-IAA $\{B\}$	$\{B\}$, 2	**59.05**

Table 5.4: Inter Annotator Agreement (IAA) scores for the different A, B, C annotation sets using the ERRANT $F_{0.5}$ metric. RAND(n) represents a random selection of "n" items from the respective set.

5.4.5 Impact of GEC Tools

Comparing the time-to-correct for the source sentence versus the GEC Tool output post-editing, I can quantify the impact of using GEC Tools in Text Editing.

Quality scores presented in Table 5.2 show that the GEC Tool output EC has better values compared to the Source Sentence EC. In Table 5.5, I compare the time taken (in seconds) by a human editor to correct the source sentences with and without first-pass editing by a GEC tool. I observe that GEC Tools help in reducing the post-editing time by roughly 4 seconds per sentence. Combined insights from these results indicate that incorporating the two GEC Tools in the text-editing workflow reduces editing time and generates better final target corrections. Thus, strong GEC Tools can help improve editor efficiency.

Sentence Source	Average Time per Sentence	Average Time per Word
Source Sentence	31.16	1.91
GECToR Output	26.82	1.57
GEC-PD Output	27.46	1.67

Table 5.5: The average time to correct (**in seconds**) for a sentence and word; correcting the source and after first-pass GEC Tool editing.

5.5 Methodology

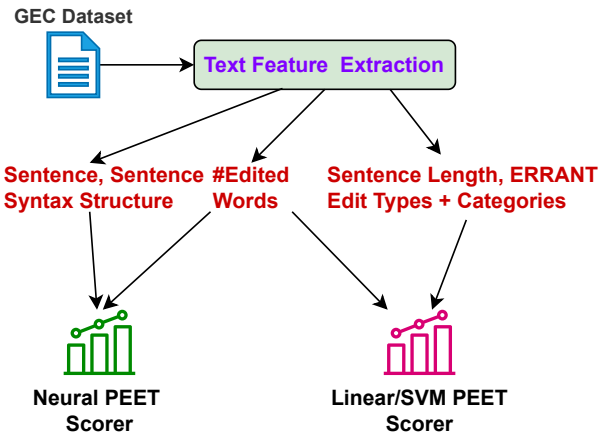

Figure 5.6: Different PEET Models and their corresponding training features.

I design statistical and neural network regression models for my post-editing effort in time (PEET) scorer. The scorer is trained to estimate the time-to-correct value for a source sentence given the target correction, using the number and type of edits and sentence property - Sentence Length, Correct/Incorrect (Figure 5.6).

5.5.1 Training Feature Sources

The dataset that I collected has three iterations for each source sentence: the original sentence - source (SRC), the first-pass correction by a GEC Tool - Model Output (MO),

and the final targeted editor correction - target (TRG). Figure 5.7 shows the three iterations for a source sentence. Each arc represents a sentence transition pairing, and the ERRANT edits extracted for each pairing (A, B and C) can be used as training features. Considering the source sentence as a feature can further separate the target edits into incorrect and ignored edits.

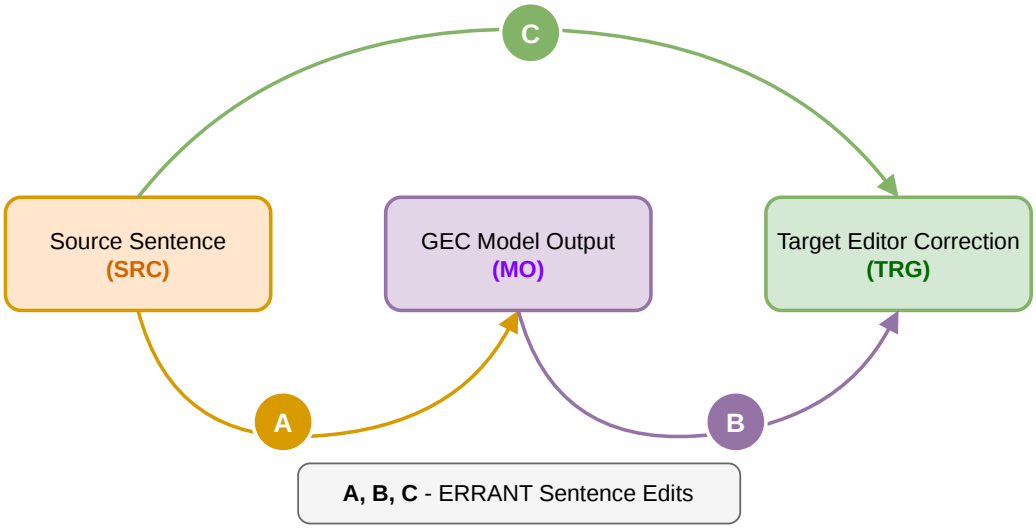

Figure 5.7: The three sentence iterations for each source sentence and the ERRANT sentence edits (A, B and C) extracted between each sentence pairing.

I consider two feature sources for training the PEET Scorer:

- **SRC - MO - TRG**: I extract the edit sets A and C (Figure 5.7) and use them to create two edit categories - Incorrect edits $|A - C|$, the GEC Tool edits that are not part of the target correction, and Ignored edits $|C - A|$, the target corrections that the GEC Tool did not make.
- **MO - TRG**: I only use the edit set B (Figure 5.7), the edits made by the editor during post-editing.

The PEET Scorer model trained on these two feature sources performs similarly, with the $MO - TRG$ features performing slightly better (Appendix E.4). This also indicates that the PEET Scorer can estimate the time-to-correct from the post-editing correction stage (B) alone. Statistical PEET models also performed as well as Neural models while allowing greater interpretability of training features (Appendix E.2). Hence, I only discuss

the features and results of the Statistical PEET Model trained using the $MO - TRG$ sentences here, referring to MO as the source.

5.5.2 ERRANT Edit Feature Extraction

I use ERRANT [27] to align and extract edit features between the source and target corrections (Section 2.5.7). Apart from the edit categories, Replacement(R), Missing(M) and Unnecessary(U), the feature also includes the edit type. The detailed ERRANT edit label breakdown and type is shown in Figure 2.5 (Section 2.5.7).

I use the number and type of edits as features for the statistical models. Similar to the edit type hierarchy used by Yuan et al. [333], considering category, type and their combination can provide 4, 25 or 55 edit features. For instance, if I only consider the three edit categories, then the four edit features are Replacement(R), Missing(M), Unnecessary(U) and Correct/Incorrect (binary feature). Using the 24 edit types 2.5 and Correct/Incorrect gives us 25 edit features. Similarly, combining edit categories with their possible types, I get 55 edit features (see Table E.6 in Appendix E.5). I train separate models for all three edit levels (4, 25, 55).

5.5.3 PEET Scorer Models

I design Linear Regression (LR) and Support Vector Regression (SVR) models, for the PEET Scorer, using the ERRANT Edit count and different edit type levels (4, 25, 55), number of edited words, source and target sentence length as features. I also experimented with Neural Regression models, but they didn't perform better than statistical models (Results in Appendix E.2). I only discuss the results of the statistical PEET models here. The details of each model and the hyperparameters are presented in Appendix E.3.

The PEET estimation task has a continuous range of prediction values - time (in seconds). I report the Mean Absolute Error (MAE) and the Pearson correlation coefficient (r) between the predicted time and the target time. However, earlier work on Quality Estimation in Machine Translation has indicated that MAE, Mean Square Error (MSE) and Root Mean Square Error (RMSE) can be a flawed metric when used to evaluate post-editing time estimation [84, 287]. MAE measures only the average magnitude of the errors, ignoring their direction and whether they are systematic or random. Since most of the time-to-correct values are close to each other, a model that predicts time estimates close to the most common time value (dataset mean) could achieve a significantly lower MAE score, giving an incorrect estimation of the model's performance.

Graham [84] recommends the Pearson correlation coefficient for evaluating such models, since it is a unit-free measure of how well the predictions co-vary with the target values, and is unaffected by changes in scale or a constant offset of either the predictions or the target values. I therefore use r to compare model performance, with a higher correlation denoting a stronger predictive model that captures the variation in effort across sentences. I report MAE alongside it to show the absolute error, since r cannot show the constant bias in the predictions. An illustrative example comparing the two metrics is presented in Appendix E.1.

5.6 Experiment Results

5.6.1 Performance of the PEET Scorer

The results for the Linear Regression (LR) and SVR PEET Scorer, with count of different edit feature levels (4,25,55), sentence word length and number of word edits as features (Section 5.5.2), are presented in Table 5.6.

As baselines, I also train the models using only the source and target sentence length (Length), only the number of edited words (Edited Words), which corresponds to the technical post-editing effort, and their combination (Length + Edited Words).

The statistical models relying on edit type information (25,55 labels) performed better than using minimal substitution, deletion and insertion edit category labels 2.5. This indicates that the type of edit has an impact on post-editing effort. The improvement over the sentence length and number of edited words baseline ($r = 0.554 \rightarrow 0.565$ for LR) reflects the subjectivity in time-to-correct scores of human editors across editors. I obtain a correlation of $r = 0.565$ from the best models (LR-25 edit features).

The sentence length baseline obtains the lowest correlation ($r = 0.429$). The number of edited words is a better predictor of post-editing time ($r = 0.527$), and combining both these features improves the correlation to $r = 0.554$ for the LR model. Adding the type of edit information further improves both r and MAE, with the 25 and 55 edit labels performing the best.

Figure 5.8 shows the predicted and actual time-to-correct values for a single run of the LR model, which is also used for the regression coefficient analysis (Section 5.6.2). One thing I observe is that the predictions for all models are compressed towards the mean time value. The edit feature models follow the $y = x$ line better than the Length baseline for

Statistical Model	Features	r	MAE
Linear Regression	Length	0.429	21.88
	Edited Words	0.527	19.90
	Length + Edited Words	0.554	19.39
	4	0.560	18.92
	25	**0.565**	**18.74**
	55	0.564	18.75
SVR Linear	Length	0.429	19.70
	Edited Words	0.527	17.29
	Length + Edited Words	0.551	17.43
	4	0.558	16.40
	25	0.564	16.19
	55	**0.565**	**16.16**

Table 5.6: Average PEET estimation performance for the Statistical Models over 50 runs (different train-test data seed). The results are presented as the Pearson Correlation (r), Mean Absolute Error (MAE) loss. The 4, 25 and 55 edit feature models also include the Length and Edited Words features.

sentences with shorter post-editing time, which corresponds to the increase in correlation in Table 5.6.

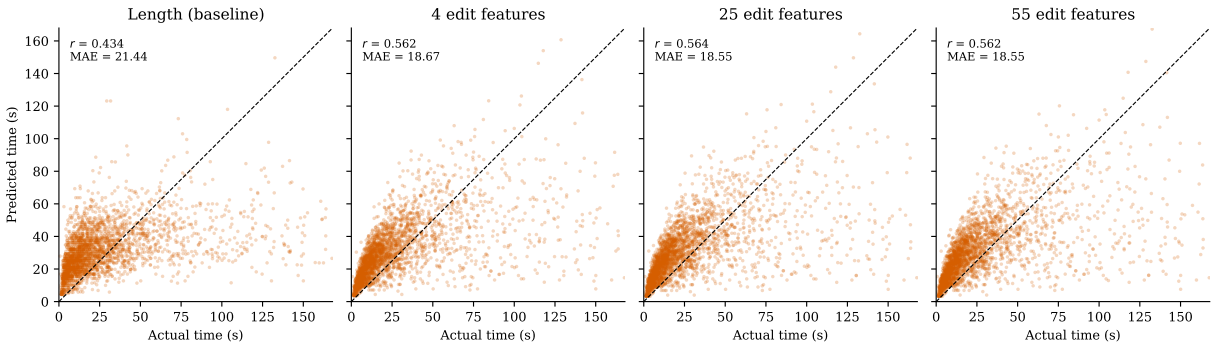

Figure 5.8: Predicted vs. actual time-to-correct for the Linear Regression PEET Scorer of a single run (seed 200), using the Length baseline and the 4, 25 and 55 edit features. The dashed line represents $y = x$. For better visualization, the axes range is limited to the 99th percentile of the time values (160.5 seconds). The r and MAE values are computed on all test sentences.

5.6.2 Impact of Error Types on Post-Edit Effort

I follow the work by Ye et al. [328], using regression coefficients of a Linear Regression (LR) model to estimate the PEET impact of different edit features. To make the coefficients interpretable, I center and standardize all edit-features by subtracting the mean and dividing by the standard deviation (except the binary/categorical edit feature - Correct/Incorrect) [258].

Model Features	Regression Coefficient	Model Features	Regression Coefficient	Model Features	Regression Coefficient
OTHER	10.15	**ORTH**	2.34	**ADJ**	0.97
PUNCT	4.55	**CONJ**	2.03	**CONTR**	0.78
PREP	4.03	**MORPH**	1.89	**VERB:INFL**	0.63
VERB	3.37	**SPELL**	1.87	**PART**	0.47
Sentence Correct	-3.31	**ADV**	1.79	**ADJ:FORM**	0.39
NOUN	3.23	**VERB:FORM**	1.66	**NOUN:INFL**	-0.30
DET	3.08	**WO**	1.63	**NOUN:POSS**	0.25
NOUN:NUM	2.52	**VERB:SVA**	1.16	-	-
VERB:TENSE	2.35	**PRON**	1.10	-	-

Table 5.7: The standardized regression coefficients of the LR model trained on the medium (25) edit features to measure the impact of each feature on PEET estimation.

The edit category *OTHER*, which corresponds to paraphrasing or rewriting text and modifying punctuation, has the highest impact on post-editing time. Deciding whether a sentence needs any correction also contributes considerably to the post-editing effort. Since the binary Correct/Incorrect feature is not standardized, its regression coefficient (-3.31, Table 5.7) indicates that a sentence that is already correct takes $\sim$3.3 seconds less to post-edit, if all other features are constant. This is likely because the editor requires no post-editing technical effort. The coefficients to study the impact of the 25 edit features are shown in Table 5.7. Coefficients for the other edit granularities (4 and 55 labels) and all PEET sentence features are provided in Appendix E.5.

5.6.3 PEET Scorer for GEC Quality Estimation

Since an efficient GEC Tool would reduce post-editing (PE) time, PE followed by PEET estimation can quantify the usability of a GEC Tool [270]. To study the correlation between cognitive, temporal and technical PE effort, I compare the PEET scorer rankings with

human judgment rankings (HJR) (Section 5.3.2) and Word Error Rate (Technical Effort) of GEC Tools. I evaluate the PEET-Linear Regression (25 Edit Features) Scorer (Section 5.5.2) estimated ranking for 33 GEC Tools in three GEC HJR (Appendix E.6).

- *Grundkiewicz-C14(EW)* - ranking of 12 GEC Tools that participated in the official CONLL-14 - GEC Task [200] by Grundkiewicz, Junczys-Dowmunt, and Gillian [90].
- *SEEDA-C14-All(TS)* - ranking of 15 newer and stronger GEC Tools on the CONLL-14 test dataset by Kobayashi, Mita, and Komachi [136]. *SEEDA-C14-NO(TS)* denotes the subset of 12 GEC tools without the three outliers.
- *Napoles-FCE* and *Napoles-Wiki* - ranking of six Seq2Seq GEC Tools on the FCE [323] and WikiEd [89] datasets by Napoles, Nădejde, and Tetreault [197].

Human Judgment Ranking	**PEET Metric**		**WER**	
	ρ	r	ρ	r
Grundkiewicz - C14 (EW)	0.48	0.26	0.28	0.18
SEEDA - C14 - All (TS)	0.18	0.63	0.18	0.65
SEEDA - C14 - NO (TS)	-0.1	-0.27	-0.1	-0.33
Napoles - FCE	**-0.96**	-0.94	**-0.96**	-0.88
Napoles - Wiki	**-0.71**	-0.63	**-0.93**	-0.88

Table 5.8: The correlation of the PEET model ranking with human-judgment rankings (HJR). I also provide the correlation of the HJR with the Word Error Rate (WER) metric. Spearman (ρ) and Pearson (r) correlation scores are used for comparison. A high negative correlation indicates lower time-to-correct and WER score corresponding to a higher human judgment ranking.

The *Grundkiewicz-C14* and *SEEDA-C14* human ranking calculation was conducted using the Expected Wins (EW) [21] and TrueSkill (TS) [100] method, which tracks relative ranking based on a set-wise comparison of a subset of all GEC Tool corrections. The EW and TS rankings were selected for the final *Grundkiewicz-C14* and *SEEDA-C14* rankings, respectively. The *Napoles - FCE* and *Napoles - Wiki* human ranking addressed the issue of partial comparison and relative ranking for GEC Tools by using the partial ranking with scalars (PRWS) method [255], collecting a quality score (0-100) for each sentence to infer the final rankings.

Table 5.8 shows the Pearson (r) and Spearman (ρ) correlation scores of the HJRs with the PEET model ranking and the Word Error Rate (WER) [266] (number of edits required

to correct a GEC Tool prediction). The WER and PEET are calculated using untargeted references, which may contribute to the lower alignment with perceived cognitive effort judgment.

I observe a good alignment (high negative correlation) between the PEET ranking and the *Napoles* HJR and a poor alignment (positive correlation) with the other HJRs. The positive correlation for SEEDA-C14-All may mainly be due to the three outlier systems, which add fluency-oriented corrections that require more edits compared to the minimal-edit references while being ranked higher by human judges [136]. The PEET ranking shows better alignment to HJRs that align with WER scores (Technical PE effort - Section 5.3.2). I also observe that human quality rankings collected using PRWS align better with the technical effort (WER) than those collected using TS or EW.

These results suggest that the PEET Scorer can estimate GEC Tool usability when output quality depends on further Post-Editing Effort (WER and type of edits) required to correct the tool output. Hence, performing PE to obtain the closest correction (lower WER) can improve GEC temporal effort estimation.

5.7 Limitations

One of the main limitations of Post-Editing (PE) Effort estimation is incorporating human annotation to evaluate GEC Tool performance, which can be expensive. However, PE allows us to quantify the true performance from a human-in-the-loop perspective. Currently, this work is limited to automated edit-type features generated by the ERRANT toolkit [27]. Evaluating the PEET Scorer as a GEC quality estimation tool shows that it is effective when the correction quality is dependent on the technical post-editing effort. However, similar to work in Machine Translation, it is inconsistent with quality estimation based on perceived PE efforts. Finally, I acknowledge that my work is limited to only the English language. Future work on post-editing GEC for other languages can show the impact of language type on PEET for GEC.

5.8 Conclusion and Future Work

Since I collect and present the first study and dataset of Post-Editing Effort (PEET) in Time for GEC, my goal is to provide a baseline for future work in this area. Using this dataset, I quantified the editing time saved and the final correction quality improvement

when using two GEC Tools for Text Editing. I also extract various automated sentence properties and edit type features from the sentence correction pairs to train the PEET Scorer. Recently, there has been some work in the area of Grammatical Error Explanation to define descriptive error types [69, 327] and use LLMs for error explanation [267, 163]. As future work, the descriptive edits can be used as possible features for the PEET model. Finally, I observe that the PEET model works well for GEC Tool evaluation when the output quality is dependent on the Technical PE Effort (amount of edits). Studying actual cognitive effort for GEC post-editing and how it compares with technical and temporal effort is another interesting direction for future work.

Chapter 6

Conclusion and Future Work

This thesis focuses on Text Quality Improvement - taking imperfect text produced by a system or a human, and improving it using a neural network model or in collaboration with a human editor through Post-Editing (PE). I focus on this task through three different projects - (1) correcting ASR transcription errors in a low-resource industry setting (Chapter 3), (2) designing an edit-specific Grammatical Error Correction (GEC) system through prediction ensembling (Chapter 4), and evaluating GEC tools using the temporal post-editing effort required for further correction by a human editor (Chapter 5).

A common theme connecting all three projects was designing a solution that makes use of the limited supervised data effectively. I also focused on the *type* of correction edits (as a feature) since it can have an impact on building correction models and evaluating them.

6.1 Low-Resource ASR Transcript Correction

In Chapter 3, I focus on ASR transcript correction in a text-only, low-resource industry setting, using a dataset of anonymized customer support call transcripts. This real-world setting is common across industries where only a small amount of source-target bitext is available for training, and the original audio is not available due to various privacy regulations. I utilize a semi-supervised synthetic data augmentation approach to generate the correction training data and use it to develop Seq2Seq and LLM instruction-based Post-Editing models for English and French transcripts. I also conducted a human annotation study to categorize different transcription error types and use it to better analyze the correction models.

The synthetic data-augmented Seq2Seq correction model is the only approach that reduced the Word Error Rate (WER) on the test sets of both industry and public datasets across both languages. The synthetic N-Best ASR correction candidates perform better than in-context few-shot LLM ASR correction in most settings. However, no LLM approach reduced the WER below the ASR source. These results indicate the effectiveness of using synthetic data for low-resource ASR transcript correction. The error type annotation labelling also allowed comparing the performance of different correction approaches across different error types. Finally, to handle over-corrections, different filtering methods were explored to limit the space of allowed edits; however, they did not consistently improve the trade-off between errors corrected and errors introduced.

One of the primary limitations of this work stems from its low-resource setting - the unavailability of the input audio signal. Correcting the transcript text without the audio information places a ceiling on errors that can be easily corrected. For instance, errors involving missing words/phrases cannot be perfectly reconstructed from context alone. This suggests several future directions worth pursuing. Working with shorter source transcript spans may better match the local nature of most transcription errors and reduce over-corrections. Since different correction approaches perform better/poorly on different error categories, designing solutions for specific error types and combining them can provide a more precise correction system. Finally, for downstream applications where the transcript content and fluency are important, framing the task as transcript paraphrasing rather than error correction may be more appropriate.

6.2 Edit Type Ensembling for Grammatical Error Correction

Filtering certain edit types from the training data in Chapter 3 showed that Seq2Seq models can be trained to selectively perform specific kinds of edits. Utilizing the different edit type information, along with the goal of using the limited training data more efficiently, motivated the Edit Type Ensembling (ETE) framework proposed in Chapter 4. The framework splits GEC into three sub-tasks based on the edit operations, insertion, deletion, and substitution, training an edit-specific model for each type along with a baseline GEC model trained on all edits jointly, ensembling all four predictions. Apart from a regression-based edit combiner, I also introduce two LLM-based ensembling methods, one based on instruction prompting and one on edit quality classification.

The results support the hypothesis behind the ETE framework. Using the same baseline

architecture and the same training data, the edit-specific ensemble produces a stronger and more precise GEC model than the baseline. Performing an edit-level performance analysis indicates that the ensembling approach is able to select a high-confidence set of corrections, reducing unnecessary over-corrections. The LLM-based edit combiners show different results, losing the high precision scores for improved recall, which can make them an alternative when detailed edit-level feature labels (from ERRANT) are not available to train the regression-based combiner, although they do not improve over the baseline on BEA-2019.

This work is deliberately limited to a single baseline BART-architecture model, since we want to use the baseline architecture and data more efficiently. However, the framework may be adaptable to other Seq2Seq GEC models. The edit ensemble has the recall ceiling of all its component models, since it can only select the final edits from the set of all proposed edits. A more detailed edit-type structure than the three main operations and edit category features generated using LLM-based error explanation rather than the ERRANT label classification could provide the regression ensemble with more descriptive features to train on.

6.3 Post-Editing Effort in Time for GEC Evaluation

In Chapter 5 I moved from text correction models to a human-centric GEC Tool evaluation metric. Since GEC tool-assisted editing is an iterative process, the post-editing effort required to reach the desired final version can serve as a quality estimate for the GEC tool. For this project, I collected the first large-scale post-edit correction dataset for GEC, with the time-to-correct annotations. I use this dataset to quantify the editing time saved and the quality improvements when using a GEC tool for the first edit pass. I also propose the PEET Scorer, a metric to estimate the time-to-correct value based on automatically extracted sentence and edit-type features.

The results indicate that the edit type information improves the post-editing time estimate over the sentence length and net edit count alone, with rewriting/paraphrasing and punctuation edits requiring the greatest editing time. Deciding whether a sentence needs correction at all also contributes considerably to the editing time. Comparing the PEET ranking of GEC tools against human judgment rankings shows that the scorer aligns well with the rankings that correlate strongly with the technical editing effort, and poorly when they are based on perceived correction quality.

The main limitation of this project is the reliance on and need for human correction annotations, which is expensive and time-consuming to collect, and the study is restricted

to English and the automated edit label features predicted by the ERRANT toolkit. Recent work on Grammatical Error Explanation using LLMs predicts descriptive error types that can be a source of better feature labels for the PEET model. Extending the time-to-correct data collection beyond English would show the impact of language type on the post-editing effort. Finally, measuring cognitive effort for GEC post-editing, along with the temporal and technical effort, would make it possible to study the impact and correlation between all three types of post-editing effort.

6.4 Summary

Across all three projects, the goal was to perform or evaluate Text Quality Improvement solutions. In Chapter 3, industry constraints such as limited training data and no access to the audio signal made me use synthetic data augmentation to design an ASR Transcript correction system. In Chapter 4, I focus on error-type features to use the limited GEC training data better, producing a stronger correction model without any additional data or a change in architecture through edit-type ensembling. Finally, in Chapter 5 I propose a human-centric metric that estimates the temporal post-editing effort of a GEC Tool. Overall, the results in this thesis indicate that correction models and their evaluation can be improved in settings with limited supervised data and resources, through synthetic data augmentation, edit-type-specific ensembling, and post-editing effort estimation.

References

[1] Sulaiman Aftan and Habib Shah. "A survey on bert and its applications". In: *2023 20th Learning and Technology Conference (L&T)*. IEEE. 2023, pp. 161–166. DOI: 10.1109/lt58159.2023.10092289.

[2] Sadeen Alharbi, Muna Alrazgan, Alanoud Alrashed, Turkiayh Alnomasi, Raghad Almojel, Rimah Alharbi, Saja Alharbi, Sahar Alturki, Fatimah Alshehri, and Maha Almojil. "Automatic speech recognition: Systematic literature review". In: *Ieee Access* 9 (2021), pp. 131858–131876. DOI: 10.1109/access.2021.3112535.

[3] Øistein E Andersen, Helen Yannakoudakis, Fiona Barker, and Tim Parish. "Developing and testing a self-assessment and tutoring system". In: *Proceedings of the eighth workshop on innovative use of NLP for building educational applications*. 2013, pp. 32–41.

[4] Abhijeet Awasthi, Sunita Sarawagi, Rasna Goyal, Sabyasachi Ghosh, and Vihari Piratla. "Parallel Iterative Edit Models for Local Sequence Transduction". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, 2019, pp. 4260–4270. DOI: 10.18653/v1/D19-1435.

[5] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. "wav2vec 2.0: A framework for self-supervised learning of speech representations". In: *Advances in neural information processing systems* 33 (2020), pp. 12449–12460.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *3rd International Conference on Learning Representations (ICLR)*. 2015. URL: https://arxiv.org/abs/1409.0473.

[7] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. "Qwen Technical Report". In: *arXiv preprint arXiv:2309.16609* (2023). DOI: 10.48550/arXiv.2309.16609.

[8] Srinivas Bangalore, Bergljot Behrens, Michael Carl, Maheshwar Gankhot, Arndt Heilmann, Jean Nitzke, Moritz Schaeffer, and Annegret Sturm. "The role of syntactic variation in translation and post-editing". In: *Translation Spaces* 4.1 (2015), pp. 119–144. DOI: 10.1075/ts.4.1.06sch.

[9] Sergio Barrachina, Oliver Bender, Francisco Casacuberta, Jorge Civera, Elsa Cubel, Shahram Khadivi, Antonio Lagarda, Hermann Ney, Jesús Tomás, Enrique Vidal, and Juan-Miguel Vilar. "Statistical Approaches to Computer-Assisted Translation". In: *Computational Linguistics* 35.1 (2009), pp. 3–28. DOI: 10.1162/coli.2008.07-055-R2-06-29.

[10] Youssef Bassil and Paul Semaan. "Asr context-sensitive error correction based on microsoft n-gram dataset". In: *arXiv preprint arXiv:1203.5262* (2012). DOI: 10.48550/arXiv.1203.5262.

[11] Jonas Becker, Jan Philip Wahle, Bela Gipp, and Terry Ruas. "Text Generation: A Systematic Literature Review of Tasks, Evaluation, and Challenges". In: *Journal of Artificial Intelligence Research* (2026). DOI: 10.1613/jair.1.22258.

[12] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. "Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks". In: *Advances in Neural Information Processing Systems*. Vol. 28. Curran Associates, Inc., 2015, pp. 1171–1179.

[13] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. "A neural probabilistic language model". In: *The journal of machine learning research* 3 (2003), pp. 1137–1155.

[14] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. "Curriculum learning". In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 41–48. DOI: 10.1145/1553374.1553380.

[15] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. "Dota 2 with Large Scale Deep Reinforcement Learning". In: *arXiv preprint arXiv:1912.06680* (2019). DOI: 10.48550/arXiv.1912.06680.

[16] Leonardo Berti, Flavio Giorgi, and Gjergji Kasneci. "Emergent abilities in large language models: A survey". In: *arXiv preprint arXiv:2503.05788* (2025). DOI: 10.48550/arXiv.2503.05788.

[17] Abhishek Bhandari and Gaurav Harit. "Post-ASR Correction for Low-Resource Rajasthani Language". In: *ACM Transactions on Asian and Low-Resource Language Information Processing* (2026). DOI: 10.1145/3793254.

[18] Giovanni Maria Biancofiore, Yashar Deldjoo, Tommaso Di Noia, Eugenio Di Sciascio, and Fedelucio Narducci. "Interactive question answering systems: Literature review". In: *ACM Computing Surveys* 56.9 (2024), pp. 1–38. DOI: 10.1145/3657631.

[19] Christopher M. Bishop. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[20] Daniel G Bobrow, Ronald M Kaplan, Martin Kay, Donald A Norman, Henry Thompson, and Terry Winograd. "GUS, a frame-driven dialog system". In: *Artificial intelligence* 8.2 (1977), pp. 155–173. DOI: 10.1016/0004-3702(77)90018-2.

[21] Ondrej Bojar, Christian Buck, Chris Callison-Burch, Christian Federmann, Barry Haddow, Philipp Koehn, Christof Monz, Matt Post, Radu Soricut, and Lucia Specia. "Findings of the 2013 Workshop on Statistical Machine Translation". In: *WMT@ACL*. 2013. URL: https://api.semanticscholar.org/CorpusID:1009868.

[22] Tiberiu Boroş, Stefan Daniel Dumitrescu, Adrian Zafiu, Verginica Barbu Mititelu, and Ionut Paul Văduva. "RACAI GEC–a hybrid approach to grammatical error correction". In: *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. 2014, pp. 43–48. DOI: 10.3115/v1/W14-1705.

[23] Andrey Bout, Alexander Podolskiy, Sergey Nikolenko, and Irina Piontkovskaya. "Efficient grammatical error correction via multi-task training and optimized training schedule". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 2023, pp. 5800–5816. DOI: 10.18653/v1/2023.emnlp-main.355.

[24] Chris Brockett, William B. Dolan, and Michael Gamon. "Correcting ESL Errors Using Phrasal SMT Techniques". In: *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*. Sydney, Australia: Association for Computational Linguistics, 2006, pp. 249–256. DOI: 10.3115/1220175.1220207.

[25] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[26] Christopher Bryant, Mariano Felice, Øistein E Andersen, and Ted Briscoe. "The BEA-2019 shared task on grammatical error correction". In: *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*. 2019, pp. 52–75. DOI: 10.18653/v1/W19-4406.

[27] Christopher Bryant, Mariano Felice, and Ted Briscoe. "Automatic Annotation and Evaluation of Error Types for Grammatical Error Correction". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, 2017, pp. 793–805. DOI: 10.18653/v1/P17-1074.

[28] Christopher Bryant and Hwee Tou Ng. "How far are we from fully automatic high quality grammatical error correction?" In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2015, pp. 697–707. DOI: 10.3115/v1/P15-1068.

[29] Christopher Bryant, Zheng Yuan, Muhammad Reza Qorib, Hannan Cao, Hwee Tou Ng, and Ted Briscoe. "Grammatical error correction: A survey of the state of the art". In: *Computational Linguistics* 49.3 (2023), pp. 643–701. DOI: 10.1162/coli_a_00478.

[30] Aoife Cahill, Martin Chodorow, Susanne Wolff, and Nitin Madnani. "Detecting missing hyphens in learner text". In: *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*. 2013, pp. 300–305.

[31] Michael Carl, Barbara Dragsted, Jakob Elming, Daniel Hardt, and Arnt Lykke Jakobsen. "The process of post-editing: A pilot study". In: *Copenhagen Studies in Language* 41.1 (2011), pp. 131–142.

[32] Asli Celikyilmaz, Elizabeth Clark, and Jianfeng Gao. "Evaluation of text generation: A survey". In: *arXiv preprint arXiv:2006.14799* (2020). DOI: 10.48550/arXiv.2006.14799.

[33] Ozgür Cetin and Elizabeth Shriberg. "Speaker overlaps and ASR errors in meetings: Effects before, during, and after the overlap". In: *2006 IEEE international conference on Acoustics Speech and Signal Processing Proceedings*. Vol. 1. IEEE. 2006, pp. I-357–I-360. DOI: 10.1109/icassp.2006.1660031.

[34] Hoyeon Chang, Jinho Park, Seonghyeon Ye, Sohee Yang, Youngkyung Seo, Du-Seong Chang, and Minjoon Seo. "How do large language models acquire factual knowledge during pretraining?" In: *Advances in neural information processing systems* 37 (2024), pp. 60626–60668. DOI: 10.52202/079017-1939.

[35] Shreyas Chaudhari, Pranjal Aggarwal, Vishvak Murahari, Tanmay Rajpurohit, Ashwin Kalyan, Karthik Narasimhan, Ameet Deshpande, and Bruno Castro da Silva. "Rlhf deciphered: A critical analysis of reinforcement learning from human feedback for llms". In: *ACM Computing Surveys* 58.2 (2025), pp. 1–37. DOI: 10.1145/3743127.

[36] Chen Chen, Yuchen Hu, Chao-Han Huck Yang, Sabato Marco Siniscalchi, Pin-Yu Chen, and Eng-Siong Chng. "Hyporadise: An open baseline for generative speech recognition with large language models". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 31665–31688. DOI: 10.52202/075280-1375.

[37] Shijie Chen, Yu Zhang, and Qiang Yang. "Multi-task learning in natural language processing: An overview". In: *ACM Computing Surveys* 56.12 (2024), pp. 1–32. DOI: 10.1145/3663363.

[38] Stanley F Chen and Joshua Goodman. "An empirical study of smoothing techniques for language modeling". In: *Computer Speech & Language* 13.4 (1999), pp. 359–393. DOI: 10.1006/csla.1999.0128.

[39] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179.

[40] Martin Chodorow and Claudia Leacock. "An unsupervised method for detecting grammatical errors". In: *1st Meeting of the North American Chapter of the Association for Computational Linguistics*. 2000.

[41] Yo Joong Choe, Jiyeon Ham, Kyubyong Park, and Yeoil Yoon. "A Neural Grammatical Error Correction System Built On Better Pre-training and Sequential Transfer Learning". In: *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*. Florence, Italy: Association for Computational Linguistics, 2019, pp. 213–227. DOI: 10.18653/v1/W19-4423.

[42] Shamil Chollampatt and Hwee Tou Ng. "A multilayer convolutional encoder-decoder neural network for grammatical error correction". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018. DOI: 10.1609/aaai.v32i1.12069.

[43] Jan Chorowski and Navdeep Jaitly. "Towards Better Decoding and Language Model Integration in Sequence to Sequence Models". In: *Proceedings of Interspeech 2017*. 2017, pp. 523–527. DOI: 10.21437/Interspeech.2017-343.

[44] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. "Deep reinforcement learning from human preferences". In: *Advances in neural information processing systems* 30 (2017).

[45] James Clarke and Mirella Lapata. "Discourse constraints for document compression". In: *Computational Linguistics* 36.3 (2010), pp. 411–441. DOI: 10.1162/coli_a_00004.

[46] Kenneth Mark Colby. *Artificial paranoia: A computer simulation of paranoid processes*. Vol. 49. Elsevier, 2013.

[47] Kenneth Mark Colby, Franklin Dennis Hilf, Sylvia Weber, and Helena C Kraemer. "Turing-like indistinguishability tests for the validation of a computer simulation of paranoid processes". In: *Artificial Intelligence* 3 (1972), pp. 199–221. DOI: 10.1016/0004-3702(72)90049-5.

[48] Steven Coyne, Keisuke Sakaguchi, Diana Galvan-Sosa, Michael Zock, and Kentaro Inui. "Analyzing the performance of gpt-3.5 and gpt-4 in grammatical error correction". In: *arXiv preprint arXiv:2303.14342* (2023). DOI: 10.48550/arXiv.2303.14342.

[49] Joke Daems, Sonia Vandepitte, Robert J Hartsuiker, and Lieve Macken. "Identifying the machine translation error types with the greatest impact on post-editing effort". In: *Frontiers in psychology* 8 (2017), p. 1282. DOI: 10.3389/fpsyg.2017.01282.

[50] Joke Daems, Sonia Vandepitte, Robert Hartsuker, and Lieve Macken. "The impact of machine translation error types on post-editing effort indicators". In: *Proceedings of the 4th Workshop on Post-editing Technology and Practice*. 2015.

[51] Daniel Dahlmeier and Hwee Tou Ng. "Better evaluation for grammatical error correction". In: *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2012, pp. 568–572. URL: https://aclanthology.org/N12-1067/.

[52] Daniel Dahlmeier, Hwee Tou Ng, and Siew Mei Wu. "Building a large annotated corpus of learner English: The NUS corpus of learner English". In: *Proceedings of the eighth workshop on innovative use of NLP for building educational applications*. 2013, pp. 22–31.

[53] Junqi Dai, Hang Yan, Tianxiang Sun, Pengfei Liu, and Xipeng Qiu. "Does syntax matter? A strong baseline for Aspect-based Sentiment Analysis with RoBERTa". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2021, pp. 1816–1829. DOI: 10.18653/v1/2021.naacl-main.146.

[54] Christopher Davis, Andrew Caines, Øistein E. Andersen, Shiva Taslimipoor, Helen Yannakoudakis, Zheng Yuan, Christopher Bryant, Marek Rei, and Paula Buttery. "Prompting open-source and commercial language models for grammatical error correction of English learner text". In: *Findings of the Association for Computational Linguistics: ACL 2024*. Bangkok, Thailand: Association for Computational Linguistics, 2024, pp. 11952–11967. DOI: 10.18653/v1/2024.findings-acl.711.

[55] Ken H Davis, R Biddulph, and Stephen Balashek. "Automatic recognition of spoken digits". In: *The Journal of the Acoustical Society of America* 24.6 (1952), pp. 637–642. DOI: 10.1121/1.1906946.

[56] DeepSeek-AI. "DeepSeek LLM: Scaling Open-Source Language Models with Longtermism". In: *arXiv preprint arXiv:2401.02954* (2024). DOI: 10.48550/arXiv.2401.02954.

[57] DeepSeek-AI. "DeepSeek-R1 Incentivizes Reasoning in LLMs through Reinforcement Learning". In: *Nature* 645.8081 (2025), pp. 633–638. DOI: 10.1038/s41586-025-09422-z.

[58] DeepSeek-AI. "DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model". In: *arXiv preprint arXiv:2405.04434* (2024). DOI: 10.48550/arXiv.2405.04434.

[59] DeepSeek-AI. "DeepSeek-V3 Technical Report". In: *arXiv preprint arXiv:2412.19437* (2024). DOI: 10.48550/arXiv.2412.19437.

[60] DeepSeek-AI. "DeepSeek-V4: Towards Highly Efficient Million-Token Context Intelligence". In: *arXiv preprint arXiv:2606.19348* (2026). DOI: 10.48550/arXiv.2606.19348.

[61] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

[62] Amandeep Singh Dhanjal and Williamjeet Singh. "A comprehensive survey on automatic speech recognition using neural networks". In: *Multimedia Tools and Applications* 83.8 (2024), pp. 23367–23412. DOI: 10.1007/s11042-023-16438-y.

[63] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. "A Survey on In-context Learning". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Miami, Florida, USA: Association for Computational Linguistics, 2024, pp. 1107–1128. DOI: 10.18653/v1/2024.emnlp-main.64.

[64] Samrat Dutta, Shreyansh Jain, Ayush Maheshwari, Souvik Pal, Ganesh Ramakrishnan, and Preethi Jyothi. "Error correction in asr using sequence-to-sequence models". In: *arXiv preprint arXiv:2202.01157* (2022). DOI: 10.48550/arXiv.2202.01157.

[65] Tao Fang, Xuebo Liu, Derek F Wong, Runzhe Zhan, Liang Ding, Lidia S Chao, Dacheng Tao, and Min Zhang. "Transgec: Improving grammatical error correction with translationese". In: *Findings of the association for computational linguistics: ACL 2023*. 2023, pp. 3614–3633. DOI: 10.18653/v1/2023.findings-acl.223.

[66] Tao Fang, Shu Yang, Kaixin Lan, Derek F Wong, Jinpeng Hu, Lidia S Chao, and Yue Zhang. "Is chatgpt a highly fluent grammatical error correction system? a comprehensive evaluation". In: *arXiv preprint arXiv:2304.01746* (2023). DOI: 10.48550/arXiv.2304.01746.

[67] Tanveer A Faruquie, Sumit Negi, and L Venkata Subramaniam. "Protecting sensitive customer information in call center recordings". In: *2009 IEEE International Conference on Services Computing*. IEEE. 2009, pp. 81–88. DOI: 10.1109/scc.2009.51.

[68] Maryam Fazel-Zarandi, Longshaokan Wang, Aditya Tiwari, and Spyros Matsoukas. "Investigation of error simulation techniques for learning dialog policies for conversational error recovery". In: *arXiv preprint arXiv:1911.03378* (2019). DOI: 10.48550/arXiv.1911.03378.

[69] Yuejiao Fei, Leyang Cui, Sen Yang, Wai Lam, Zhenzhong Lan, and Shuming Shi. "Enhancing Grammatical Error Correction Systems with Explanations". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, 2023, pp. 7489–7501. DOI: 10.18653/v1/2023.acl-long.413.

[70] Mariano Felice, Zheng Yuan, Øistein E Andersen, Helen Yannakoudakis, and Ekaterina Kochmar. "Grammatical error correction using hybrid systems and type filtering". In: *Proceedings of the eighteenth conference on computational natural language learning: shared task*. 2014, pp. 15–24. DOI: 10.3115/v1/W14-1702.

[71] Jonathan G Fiscus. "A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (ROVER)". In: *1997 IEEE workshop on automatic speech recognition and understanding proceedings*. IEEE. 1997, pp. 347–354. DOI: 10.1109/asru.1997.659110.

[72] Zhenxin Fu, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, and Rui Yan. "Style Transfer in Text: Exploration and Evaluation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018, pp. 663–670. DOI: 10.1609/aaai.v32i1.11330.

[73] Hayato Futami, Hirofumi Inaguma, Sei Ueno, Masato Mimura, Shinsuke Sakai, and Tatsuya Kawahara. "Distilling the Knowledge of BERT for Sequence-to-Sequence ASR". In: *Proceedings of Interspeech 2020*. 2020, pp. 3635–3639. DOI: 10.21437/Interspeech.2020-1179.

[74] Michael Gamon, Claudia Leacock, Chris Brockett, William B. Dolan, Jianfeng Gao, Dmitriy Belenko, and Alexandre Klementiev. "Using Statistical Techniques and Web Search to Correct ESL Errors". In: *CALICO Journal* 26.3 (2009), pp. 491–511. DOI: 10.1558/cj.v26i3.491-511.

[75] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. "Retrieval-augmented generation for large language models: A survey". In: *arXiv preprint arXiv:2312.10997* (2023). DOI: 10.48550/arXiv.2312.10997.

[76] Nadia Mushtaq Gardazi, Ali Daud, Muhammad Kamran Malik, Amal Bukhari, Tariq Alsahfi, and Bader Alshemaimri. "BERT applications in natural language processing: a review". In: *Artificial Intelligence Review* 58.6 (2025), p. 166. DOI: 10.1007/s10462-025-11162-5.

[77] Ankush Garg and Mayank Agarwal. "Machine translation: a literature review". In: *arXiv preprint arXiv:1901.01122* (2018). DOI: 10.48550/arXiv.1901.01122.

[78] Albert Gatt and Emiel Krahmer. "Survey of the state of the art in natural language generation: Core tasks, applications and evaluation". In: *Journal of Artificial Intelligence Research* 61 (2018), pp. 65–170. DOI: 10.1613/jair.5477.

[79] Zorik Gekhman, Dina Zverinski, Jonathan Mallinson, and Genady Beryozkin. "RED-ACE: Robust Error Detection for ASR using Confidence Embeddings". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 2800–2808. DOI: 10.18653/v1/2022.emnlp-main.180. URL: https://aclanthology.org/2022.emnlp-main.180.

[80] Gemini Team, Google. "Gemini: A Family of Highly Capable Multimodal Models". In: *arXiv preprint arXiv:2312.11805* (2023). DOI: 10.48550/arXiv.2312.11805.

[81] Eli Goldberg, Norbert Driedger, and Richard I Kittredge. "Using natural-language processing to produce weather forecasts". In: *IEEE Expert* 9.2 (1994), pp. 45–53. DOI: 10.1109/64.294135.

[82] Peiyuan Gong, Xuebo Liu, He-Yan Huang, and Min Zhang. "Revisiting Grammatical Error Correction Evaluation and Beyond". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 2022, pp. 6891–6902. DOI: 10.18653/v1/2022.emnlp-main.463.

[83] Takumi Gotou, Ryo Nagata, Masato Mita, and Kazuaki Hanawa. "Taking the correction difficulty into account in grammatical error correction evaluation". In: *Proceedings of the 28th International Conference on Computational Linguistics*. 2020, pp. 2085–2095. DOI: 10.18653/v1/2020.coling-main.188.

[84] Yvette Graham. "Improving evaluation of machine translation quality estimation". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2015, pp. 1804–1813. DOI: 10.3115/v1/P15-1174.

[85] Sylviane Granger. "The computer learner corpus: a versatile new source of data for SLA research". In: *Learner English on Computer*. Routledge, 2014, pp. 3–18. DOI: 10.4324/9781315841342-1.

[86] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks". In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 369–376. DOI: 10.1145/1143844.1143891.

[87] Spence Green, Jeffrey Heer, and Christopher D Manning. "The efficacy of human post-editing for language translation". In: *Proceedings of the SIGCHI conference on human factors in computing systems*. 2013, pp. 439–448. DOI: 10.1145/2470654.2470718.

[88] Roman Grundkiewicz, Christopher Bryant, and Mariano Felice. "A Crash Course in Automatic Grammatical Error Correction". In: *Proceedings of the 28th International Conference on Computational Linguistics: Tutorial Abstracts*. 2020, pp. 33–38. DOI: 10.18653/v1/2020.coling-tutorials.6.

[89] Roman Grundkiewicz and Marcin Junczys-Dowmunt. "The wiked error corpus: A corpus of corrective wikipedia edits and its application to grammatical error correction". In: *Advances in Natural Language Processing: 9th International Conference on NLP, PolTAL 2014, Warsaw, Poland, September 17-19, 2014. Proceedings 9*. Springer. 2014, pp. 478–490. DOI: 10.1007/978-3-319-10888-9_47.

[90] Roman Grundkiewicz, Marcin Junczys-Dowmunt, and Edward Gillian. "Human evaluation of grammatical error correction systems". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015, pp. 461–470. DOI: 10.18653/v1/D15-1052.

[91] Roman Grundkiewicz, Marcin Junczys-Dowmunt, and Kenneth Heafield. "Neural Grammatical Error Correction Systems with Unsupervised Pre-training on Synthetic Data". In: *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*. 2019, pp. 252–263. DOI: 10.18653/v1/W19-4427.

[92] Zijin Gu, Tatiana Likhomanenko, He Bai, Erik McDermott, Ronan Collobert, and Navdeep Jaitly. "Denoising lm: Pushing the limits of error correction models for speech recognition". In: *arXiv preprint arXiv:2405.15216* (2024). v1 title; later arXiv revisions (2026) are retitled "Revisiting ASR Error Correction with Specialized Models". DOI: 10.48550/arXiv.2405.15216.

[93] Ana Guerberof. "Productivity and quality in MT post-editing". In: *Beyond Translation Memories: New Tools for Translators Workshop*. 2009.

[94] Jie Gui, Tuo Chen, Jing Zhang, Qiong Cao, Zhenan Sun, Hao Luo, and Dacheng Tao. "A survey on self-supervised learning: Algorithms, applications, and future trends". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.12 (2024), pp. 9052–9071. DOI: 10.1109/tpami.2024.3415112.

[95] Anubhav Gupta. "Grammatical Error Detection Using Tagger Disagreement". In: *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. Ed. by Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. Baltimore, Maryland: Association for Computational Linguistics, June 2014, pp. 49–52. DOI: 10.3115/v1/W14-1706. URL: https://aclanthology.org/W14-1706/.

[96] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. New York: Springer, 2009. DOI: 10.1007/978-0-387-84858-7.

[97] Kenneth Heafield and Alon Lavie. "Combining Machine Translation Output with Open Source: The Carnegie Mellon Multi-Engine Machine Translation Scheme". In: *The Prague Bulletin of Mathematical Linguistics* 93.1 (2010), pp. 27–36. DOI: 10.2478/v10108-010-0008-4.

[98] George E. Heidorn, Karen Jensen, Lance A. Miller, Roy J. Byrd, and Martin S Chodorow. "The EPISTLE text-critiquing system". In: *IBM Systems Journal* 21.3 (1982), pp. 305–326. DOI: 10.1147/sj.213.0305.

[99] Michael Heilman, Aoife Cahill, Nitin Madnani, Melissa Lopez, Matthew Mulholland, and Joel Tetreault. "Predicting grammaticality on an ordinal scale". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2014, pp. 174–180. DOI: 10.3115/v1/p14-2029.

[100] Ralf Herbrich, Tom Minka, and Thore Graepel. "TrueSkill™: A Bayesian Skill Rating System". In: *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*. The MIT Press, Sept. 2007. ISBN: 9780262256919. DOI: 10.7551/mitpress/7503.003.0076. eprint: https://direct.mit.edu/book/chapter-pdf/2288966/9780262256919_ccs.pdf. URL: https://doi.org/10.7551/mitpress/7503.003.0076.

[101] Matthieu Hermet and Alain Désilets. "Using First and Second Language Models to Correct Preposition Errors in Second Language Authoring". In: *Proceedings of the Fourth Workshop on Innovative Use of NLP for Building Educational Applications*. Boulder, Colorado: Association for Computational Linguistics, June 2009, pp. 64–72. DOI: 10.3115/1609843.1609853.

[102] S. David Hernandez and Hiram Calvo. "CoNLL 2014 Shared Task: Grammatical Error Correction with a Syntactic N-gram Language Model from a Big Corpora". In: *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. Ed. by Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian

Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. Baltimore, Maryland: Association for Computational Linguistics, June 2014, pp. 53–59. DOI: 10.3115/v1/W14-1707. URL: https://aclanthology.org/W14-1707/.

[103] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97. DOI: 10.1109/MSP.2012.2205597.

[104] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[105] Matthew Honnibal and Ines Montani. "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing". To appear. 2017.

[106] Wenxin Hou, Han Zhu, Yidong Wang, Jindong Wang, Tao Qin, Renjun Xu, and Takahiro Shinozaki. "Exploiting adapters for cross-lingual low-resource speech recognition". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 30 (2021), pp. 317–329. DOI: 10.1109/taslp.2021.3138674.

[107] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. "Parameter-efficient transfer learning for NLP". In: *International conference on machine learning*. PMLR. 2019, pp. 2790–2799.

[108] Oleksii Hrinchuk, Mariya Popova, and Boris Ginsburg. "Correction of automatic speech recognition with transformer sequence-to-sequence model". In: *Icassp 2020-2020 ieee international conference on acoustics, speech and signal processing (icassp)*. IEEE. 2020, pp. 7074–7078. DOI: 10.1109/icassp40776.2020.9053051.

[109] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. "Hubert: Self-supervised speech representation learning by masked prediction of hidden units". In: *IEEE/ACM transactions on audio, speech, and language processing* 29 (2021), pp. 3451–3460. DOI: 10.1109/taslp.2021.3122291.

[110] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. "LoRA: Low-Rank Adaptation of Large Language Models". In: *International Conference on Learning Representations (ICLR)*. 2022. URL: https://openreview.net/forum?id=nZeVKeeFYf9.

[111] Yuchen Hu, Chen Chen, Chao-Han Huck Yang, Ruizhe Li, Dong Zhang, Zhehuai Chen, and Eng Siong Chng. "Gentranslate: Large language models are generative multilingual speech and machine translators". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2024, pp. 74–90. DOI: 10.18653/v1/2024.acl-long.5.

[112] Rudali Huidrom and Anja Belz. "A survey of recent error annotation schemes for automatically generated text". In: *Proceedings of the 2nd workshop on natural language generation, evaluation, and metrics (GEM)*. 2022, pp. 383–398. DOI: 10.18653/v1/2022.gem-1.33.

[113] John Hutchins. "Evaluation of machine translation and translation tools". In: *Iš: Survey of the State of the Art in Human Language Technology* (1997), pp. 418–419.

[114] Kristian Tangsgaard Hvelplund. "Eye tracking and the translation process: Reflections on the analysis and interpretation of eye-tracking data". In: *MonTI. Monografías de Traducción e Interpretación* Special Issue 1 (2014), pp. 201–223. DOI: 10.6035/monti.2014.ne1.6.

[115] Md Asadul Islam and Enrico Magnani. "Is this the end of the gold standard? A straightforward reference-less grammatical error correction metric". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021, pp. 3009–3015. DOI: 10.18653/v1/2021.emnlp-main.239.

[116] Linzhao Jia, Han Sun, Yuang Wei, Changyong Qi, and Xiaozhe Yang. "Epic: Error pattern informed correction for classroom asr with limited labeled data". In: *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2025, pp. 1–5. DOI: 10.1109/icassp49660.2025.10890578.

[117] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. "Mistral 7B". In: *arXiv preprint arXiv:2310.06825* (2023). DOI: 10.48550/arXiv.2310.06825.

[118] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. "Mixtral of

Experts". In: *arXiv preprint arXiv:2401.04088* (2024). DOI: 10.48550/arXiv.2401.04088.

[119] Vineet John, Lili Mou, Hareesh Bahuleyan, and Olga Vechtomova. "Disentangled Representation Learning for Non-Parallel Text Style Transfer". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, 2019, pp. 424–434. DOI: 10.18653/v1/P19-1041.

[120] Marcin Junczys-Dowmunt and Roman Grundkiewicz. "Log-linear Combinations of Monolingual and Bilingual Neural Machine Translation Models for Automatic Post-Editing". In: *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*. Berlin, Germany: Association for Computational Linguistics, 2016, pp. 751–758. DOI: 10.18653/v1/W16-2378.

[121] Marcin Junczys-Dowmunt and Roman Grundkiewicz. "Phrase-based Machine Translation is State-of-the-Art for Automatic Grammatical Error Correction". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2016, pp. 1546–1556. DOI: 10.18653/v1/D16-1161.

[122] Marcin Junczys-Dowmunt and Roman Grundkiewicz. "The AMU System in the CoNLL-2014 Shared Task: Grammatical Error Correction by Data-Intensive and Feature-Rich Statistical Machine Translation". In: *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. 2014, pp. 25–33. DOI: 10.3115/v1/W14-1703.

[123] Dan Jurafsky and James H Martin. *Speech and language processing*. Vol. 3. Pearson London, 2014.

[124] Moussa Kamal Eddine, Antoine Tixier, and Michalis Vazirgiannis. "Barthez: a skilled pretrained french sequence-to-sequence model". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021, pp. 9369–9390. DOI: 10.18653/v1/2021.emnlp-main.740.

[125] Masahiro Kaneko, Masato Mita, Shun Kiyono, Jun Suzuki, and Kentaro Inui. "Encoder-Decoder Models Can Benefit from Pre-trained Masked Language Models in Grammatical Error Correction". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020, pp. 4248–4254. DOI: 10.18653/v1/2020.acl-main.391.

[126] Yoav Kantor, Yoav Katz, Leshem Choshen, Edo Cohen-Karlik, Naftali Liberman, Assaf Toledo, Amir Menczel, and Noam Slonim. "Learning to combine Grammatical Error Corrections". In: *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*. Florence, Italy: Association for Computational Linguistics, 2019, pp. 139–148. DOI: 10.18653/v1/W19-4414.

[127] Anisia Katinskaia and Roman Yangarber. "GPT-3.5 for Grammatical Error Correction". In: *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. Ed. by Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue. Torino, Italia: ELRA and ICCL, May 2024, pp. 7831–7843. DOI: 10.63317/3yoruwn5jgvk. URL: https://aclanthology.org/2024.lrec-main.692/.

[128] Satoru Katsumata and Mamoru Komachi. "Stronger baselines for grammatical error correction using a pretrained encoder-decoder model". In: *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*. 2020, pp. 827–832.

[129] Slava Katz. "Estimation of probabilities from sparse data for the language model component of a speech recognizer". In: *IEEE transactions on acoustics, speech, and signal processing* 35.3 (1987), pp. 400–401. DOI: 10.1109/tassp.1987.1165125.

[130] Veton Kepuska and Gamal Bohouta. "Next-generation of virtual personal assistants (microsoft cortana, apple siri, amazon alexa and google home)". In: *2018 IEEE 8th annual computing and communication workshop and conference (CCWC)*. IEEE. 2018, pp. 99–103. DOI: 10.1109/ccwc.2018.8301638.

[131] Shreya Khare, Ashish R Mittal, Anuj Diwan, Sunita Sarawagi, Preethi Jyothi, and Samarth Bharadwaj. "Low Resource ASR: The Surprising Effectiveness of High Resource Transliteration." In: *Interspeech*. 2021, pp. 1529–1533. DOI: 10.21437/interspeech.2021-2062.

[132] Nikita Kitaev and Dan Klein. "Constituency Parsing with a Self-Attentive Encoder". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Iryna Gurevych and Yusuke Miyao. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 2676–2686. DOI: 10.18653/v1/P18-1249. URL: https://aclanthology.org/P18-1249/.

[133] Shun Kiyono, Jun Suzuki, Masato Mita, Tomoya Mizumoto, and Kentaro Inui. "An Empirical Study of Incorporating Pseudo Data into Grammatical Error Correction". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, 2019, pp. 1236–1242. DOI: 10.18653/v1/D19-1119.

[134] Guillaume Klein, Yoon Kim, Yuntian Deng, Vincent Nguyen, Jean Senellart, and Alexander M. Rush. "OpenNMT: Neural Machine Translation Toolkit". In: *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*. Boston, MA, 2018, pp. 177–184. URL: https://aclanthology.org/W18-1817/.

[135] Reinhard Kneser and Hermann Ney. "Improved backing-off for m-gram language modeling". In: *1995 international conference on acoustics, speech, and signal processing*. Vol. 1. IEEE. 1995, pp. 181–184. DOI: 10.1109/icassp.1995.479394.

[136] Masamune Kobayashi, Masato Mita, and Mamoru Komachi. "Revisiting meta-evaluation for grammatical error correction". In: *Transactions of the Association for Computational Linguistics* 12 (2024), pp. 837–855. DOI: 10.1162/tacl_a_00676.

[137] Maarit Koponen. "Comparing human perceptions of post-editing effort with post-editing operations". In: *Proceedings of the seventh workshop on statistical machine translation*. 2012, pp. 181–190.

[138] Maarit Koponen. "Is machine translation post-editing worth the effort? A survey of research into post-editing and effort". In: *The Journal of Specialised Translation* 25 (2016), pp. 131–148. DOI: 10.26034/cm.jostrans.2016.303.

[139] Maarit Koponen, Wilker Aziz, Luciana Ramos, and Lucia Specia. "Post-editing time as a measure of cognitive effort". In: *Proceedings of the AMTA 2012 Workshop on Post-Editing Technology and Practice (WPTP)*. San Diego, California, USA: Association for Machine Translation in the Americas, 2012. URL: https://aclanthology.org/2012.amta-wptp.2/.

[140] Oleksandr Korniienko and Kostiantyn Omelianchuk. *Enhancing Grammatical Correctness: The Efficacy of Large Language Models in Error Correction Task*. Ukrainian Catholic University, Lviv. 2024. URL: https://apps.ucu.edu.ua/wp-content/uploads/2024/09/MS-AMLV-2024-CR-No-08-Korniienko-COND.pdf.

[141] D Kraus. "Text generation in clinical medicine-a review". In: *Methods Inf. Med* 42 (2003). DOI: 10.1055/s-0038-1634209.

[142] Hans P. Krings. *Repairing Texts: Empirical Investigations of Machine Translation Post-Editing Processes*. Ed. by Geoffrey S. Koby. Kent, Ohio: Kent State University Press, 2001.

[143] Taku Kudo and John Richardson. "SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 66–71. DOI: 10.18653/v1/D18-2012.

[144] Karen Kukich. "Techniques for automatically correcting words in text". In: *Acm Computing Surveys (CSUR)* 24.4 (1992), pp. 377–439. DOI: 10.1145/146370.146380.

[145] Rishabh Kumar, Amrith Krishna, Ganesh Ramakrishnan, and Preethi Jyothi. "Post-asr correction in hindi: Comparing language models and large language models in low-resource scenarios". In: *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2026, pp. 636–645. DOI: 10.18653/v1/2026.eacl-short.45.

[146] Anoop Kunchukuttan, Sriram Chaudhury, and Pushpak Bhattacharyya. "Tuning a grammar correction system for increased precision". In: *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. 2014, pp. 60–64. DOI: 10.3115/v1/W14-1708.

[147] Julian Kupiec, Jan Pedersen, and Francine Chen. "A trainable document summarizer". In: *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*. 1995, pp. 68–73. DOI: 10.1145/215206.215333.

[148] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. "Efficient Memory Management for Large Language Model Serving with PagedAttention". In: *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23)*. ACM, 2023, pp. 611–626. DOI: 10.1145/3600006.3613165.

[149] Moreno La Quatra, Valerio Mario Salerno, Yu Tsao, and Sabato Marco Siniscalchi. "FlanEC: Exploring Flan-T5 for Post-ASR Error Correction". In: *2024 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2024, pp. 608–615. DOI: 10.1109/SLT61566.2024.10832257.

[150] Shaopeng Lai, Qingyu Zhou, Jiali Zeng, Zhongli Li, Chao Li, Yunbo Cao, and Jinsong Su. "Type-driven multi-turn corrections for grammatical error correction". In: *Findings of the association for computational linguistics: ACL 2022*. 2022, pp. 3225–3236. DOI: 10.18653/v1/2022.findings-acl.254.

[151] Vivian Lai, Alison Smith-Renner, Ke Zhang, Ruijia Cheng, Wenjuan Zhang, Joel Tetreault, and Alejandro Jaimes-Larrarte. "An Exploration of Post-Editing Effectiveness in Text Summarization". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Seattle, United States: Association for Computational Linguistics, 2022, pp. 475–493. DOI: 10.18653/v1/2022.naacl-main.35.

[152] Claudia Leacock and Martin Chodorow. "Automated grammatical error detection". In: *Automated essay scoring: A cross-disciplinary perspective* (2003), pp. 195–207. DOI: 10.4324/9781410606860-23.

[153] Kyusong Lee and Gary Geunbae Lee. "POSTECH grammatical error correction system in the CoNLL-2014 shared task". In: *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. 2014, pp. 65–73. DOI: 10.3115/v1/W14-1709.

[154] Yukyung Lee, Joonghoon Kim, Jaehee Kim, Hyowon Cho, Jaewook Kang, Pilsung Kang, and Najoung Kim. "Checkeval: A reliable llm-as-a-judge framework for evaluating text generation using checklists". In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. 2025, pp. 15782–15809. DOI: 10.18653/v1/2025.emnlp-main.796.

[155] Shane Legg. "Machine super intelligence". PhD thesis. Università della Svizzera italiana, 2008.

[156] Yichong Leng, Xu Tan, Wenjie Liu, Kaitao Song, Rui Wang, Xiang-Yang Li, Tao Qin, Ed Lin, and Tie-Yan Liu. "Softcorrect: Error correction with soft detection for automatic speech recognition". In: *proceedings of the AAAI conference on artificial intelligence*. Vol. 37. 11. 2023, pp. 13034–13042. DOI: 10.1609/aaai.v37i11.26531.

[157] Leo Leppänen, Myriam Munezero, Mark Granroth-Wilding, and Hannu Toivonen. "Data-driven news generation for automated journalism". In: *Proceedings of the 10th International Conference on Natural Language Generation*. 2017, pp. 188–197. DOI: 10.18653/v1/w17-3528.

[158] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 7871–7880. DOI: 10.18653/v1/2020.acl-main.703.

[159] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474.

[160] Jinyu Li. "Recent Advances in End-to-End Automatic Speech Recognition". In: *APSIPA Transactions on Signal and Information Processing* 11.1 (2022), e8. DOI: 10.1561/116.00000050.

[161] Junlong Li, Fan Zhou, Shichao Sun, Yikai Zhang, Hai Zhao, and Pengfei Liu. "Dissecting Human and LLM Preferences". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Bangkok, Thailand: Association for Computational Linguistics, 2024, pp. 1790–1811. DOI: 10.18653/v1/2024.acl-long.99.

[162] Junyi Li, Tianyi Tang, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. "Pre-trained language models for text generation: A survey". In: *ACM Computing Surveys* 56.9 (2024), pp. 1–39. DOI: 10.1145/3649449.

[163] Wei Li, Wen Luo, Guangyue Peng, and Houfeng Wang. "Explanation based In-Context Demonstrations Retrieval for Multilingual Grammatical Error Correction". In: *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Albuquerque, New Mexico: Association for Computational Linguistics, 2025, pp. 4881–4897. DOI: 10.18653/v1/2025.naacl-long.251.

[164] Wei Li and Houfeng Wang. "Detection-Correction Structure via General Language Model for Grammatical Error Correction". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Bangkok, Thailand: Association for Computational Linguistics, 2024, pp. 1748–1763. DOI: 10.18653/v1/2024.acl-long.96.

[165] Xiang Lisa Li and Percy Liang. "Prefix-tuning: Optimizing continuous prompts for generation". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2021, pp. 4582–4597. DOI: 10.18653/v1/2021.acl-long.353.

[166] Yinghao Li, Xuebo Liu, Shuo Wang, Peiyuan Gong, Derek F Wong, Yang Gao, He-Yan Huang, and Min Zhang. "TemplateGEC: Improving grammatical error correction with detection template". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2023, pp. 6878–6892. DOI: 10.18653/v1/2023.acl-long.380.

[167] Chin-Yew Lin. "Knowledge-based Automatic Topic Identification". In: *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*. Cambridge, Massachusetts, USA: Association for Computational Linguistics, 1995, pp. 308–310. URL: https://aclanthology.org/P95-1046/.

[168] Ruixi Lin and Hwee Tou Ng. "System Combination for Grammatical Error Correction Based on Integer Programming". In: *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*. INCOMA Ltd., 2021, pp. 824–829. DOI: 10.26615/978-954-452-072-4_094. URL: https://aclanthology.org/2021.ranlp-1.94/.

[169] Jiachang Liu, Dinghan Shen, Yizhe Zhang, William B Dolan, Lawrence Carin, and Weizhu Chen. "What makes good in-context examples for GPT-3?" In: *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd workshop on knowledge extraction and integration for deep learning architectures*. 2022, pp. 100–114. DOI: 10.18653/v1/2022.deelio-1.10.

[170] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing". In: *ACM computing surveys* 55.9 (2023), pp. 1–35. DOI: 10.1145/3560815.

[171] Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. "Multilingual denoising pre-training for neural machine translation". In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 726–742. DOI: 10.1162/tacl_a_00343.

[172] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692* (2019). DOI: 10.48550/arXiv.1907.11692.

[173] Llama Team, AI @ Meta. "The Llama 3 Herd of Models". In: *arXiv preprint arXiv:2407.21783* (2024). DOI: 10.48550/arXiv.2407.21783.

[174] Mengsay Loem, Masahiro Kaneko, Sho Takase, and Naoaki Okazaki. "Exploring effectiveness of GPT-3 in grammatical error correction: A study on performance and controllability in prompt-based methods". In: *Proceedings of the 18th workshop on innovative use of NLP for building educational applications (BEA 2023)*. 2023, pp. 205–219. DOI: 10.18653/v1/2023.bea-1.18.

[175] Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, 2015, pp. 1412–1421. DOI: 10.18653/v1/D15-1166.

[176] Rao Ma, Mengjie Qian, Mark Gales, and Kate Knill. "ASR Error Correction Using Large Language Models". In: *IEEE Transactions on Audio, Speech and Language Processing* 33 (2025). Earlier version: arXiv:2307.04172, "Can Generative Large Language Models Perform ASR Error Correction?", pp. 1389–1401. DOI: 10.1109/TASLPRO.2025.3551083.

[177] Nina Macdonald, Lawrence Frase, P Gingrich, and Stacey Keenan. "The writer's workbench: Computer aids for text analysis". In: *IEEE Transactions on Communications* 30.1 (1982), pp. 105–110. DOI: 10.1109/tcom.1982.1095380.

[178] Nitin Madnani, Joel Tetreault, and Martin Chodorow. "Exploring Grammatical Error Correction with Not-So-Crummy Machine Translation". In: *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Montréal, Canada: Association for Computational Linguistics, June 2012, pp. 44–53.

[179] Koki Maeda, Masahiro Kaneko, and Naoaki Okazaki. "IMPARA: Impact-Based Metric for GEC Using Parallel Data". In: *Proceedings of the 29th International Conference on Computational Linguistics*. Ed. by Nicoletta Calzolari, Chu-Ren Huang, Hansaem Kim, James Pustejovsky, Leo Wanner, Key-Sun Choi, Pum-Mo Ryu, Hsin-Hsi Chen, Lucia Donatelli, Heng Ji, Sadao Kurohashi, Patrizia Paggio, Nianwen Xue, Seokhwan Kim, Younggyun Hahm, Zhong He, Tony Kyungil Lee, Enrico Santus, Francis Bond, and Seung-Hoon Na. Gyeongju, Republic of Korea: International Committee on Computational Linguistics, Oct. 2022, pp. 3578–3588. URL: https://aclanthology.org/2022.coling-1.316/.

[180] Matthew V Mahoney. "Text compression as a test for artificial intelligence". In: *AAAI/IAAI* 970 (1999).

[181] Long Mai and Julie Carson-Berndsen. "Unsupervised domain adaptation for speech recognition with unsupervised error correction". In: *arXiv preprint arXiv:2209.12043* (2022). DOI: 10.21437/interspeech.2022-10565.

[182] Mishaim Malik, Muhammad Kamran Malik, Khawar Mehmood, and Imran Makhdoom. "Automatic speech recognition: a survey". In: *Multimedia Tools and Applications* 80.6 (2021), pp. 9411–9457. DOI: 10.1007/s11042-020-10073-7.

[183] Lidia Mangu and Eric Brill. "Automatic rule acquisition for spelling correction". In: *ICML*. Vol. 97. Citeseer. 1997, pp. 187–194.

[184] Inderjeet Mani. *Automatic summarization*. Vol. 3. John Benjamins Publishing, 2001.

[185] Inderjeet Mani, Barbara Gates, and Eric Bloedorn. "Improving Summaries by Revising Them". In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*. 1999, pp. 558–565. DOI: 10.3115/1034678.1034761.

[186] Louis Martin, Benjamin Muller, Pedro Ortiz Suarez, Yoann Dupont, Laurent Romary, Éric Villemonte de La Clergerie, Djamé Seddah, and Benoît Sagot. "CamemBERT: a tasty French language model". In: *Proceedings of the 58th annual meeting of the association for computational linguistics*. 2020, pp. 7203–7219. DOI: 10.18653/v1/2020.acl-main.645.

[187] Mark T Maybury. "Generating summaries from event data". In: *Information Processing & Management* 31.5 (1995), pp. 735–751. DOI: 10.1016/0306-4573(95)00025-c.

[188] AI Meta. "Llama 4: Multimodal Intelligence". In: *Meta AI Blog.* (2025). URL: https://ai.meta.com/blog/llama-4-multimodal-intelligence/.

[189] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space". In: *1st International Conference on Learning Representations (ICLR), Workshop Track Proceedings*. 2013. URL: https://arxiv.org/abs/1301.3781.

[190] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. "Recent advances in natural language processing via large pre-trained language models: A survey". In: *ACM Computing Surveys* 56.2 (2023), pp. 1–40. DOI: 10.1145/3605943.

[191] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. "Rethinking the role of demonstrations: What makes in-context learning work?" In: *Proceedings of the 2022 conference on empirical methods in natural language processing*. 2022, pp. 11048–11064. DOI: 10.18653/v1/2022.emnlp-main.759.

[192] Zeping Min and Jinbo Wang. "Exploring the integration of large language models into automatic speech recognition systems: An empirical study". In: *International Conference on Neural Information Processing*. Springer. 2023, pp. 69–84. DOI: 10.1007/978-981-99-8181-6_6.

[193] Mikołaj Morzy. "Automatic Speech Recognition". In: *Spoken Language Processing: Conversational AI for Spontaneous Human Dialogues*. Springer, 2025, pp. 53–92. DOI: 10.1007/978-3-031-88566-2_4.

[194] Bingshen Mu, Xucheng Wan, Naijun Zheng, Huan Zhou, and Lei Xie. "Mmger: Multi-modal and multi-granularity generative error correction with llm for joint accent and speech recognition". In: *IEEE Signal Processing Letters* 31 (2024), pp. 1940–1944. DOI: 10.1109/lsp.2024.3432275.

[195] Maria Nadejde and Joel Tetreault. "Personalizing Grammatical Error Correction: Adaptation to Proficiency Level and L1". In: *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*. Hong Kong, China: Association for Computational Linguistics, 2019, pp. 27–33. DOI: 10.18653/v1/D19-5504.

[196] Gayani Nanayakkara, Nirmalie Wiratunga, David Corsar, Kyle Martin, and Anjana Wijekoon. "Clinical dialogue transcription error correction using Seq2Seq models". In: *Multimodal AI in healthcare: A paradigm shift in health intelligence*. Springer, 2022, pp. 41–57. DOI: 10.1007/978-3-031-14771-5_4.

[197] Courtney Napoles, Maria Nădejde, and Joel Tetreault. "Enabling robust grammatical error correction in new domains: Data sets, metrics, and analyses". In: *Transactions of the Association for Computational Linguistics* 7 (2019), pp. 551–566. DOI: 10.1162/tacl_a_00282.

[198] Courtney Napoles, Keisuke Sakaguchi, Matt Post, and Joel Tetreault. "GLEU Without Tuning". In: *arXiv preprint arXiv:1605.02592* (2016). DOI: 10.48550/arXiv.1605.02592.

[199] Courtney Napoles, Keisuke Sakaguchi, and Joel Tetreault. "JFLEG: A Fluency Corpus and Benchmark for Grammatical Error Correction". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, 2017, pp. 229–234. DOI: 10.18653/v1/E17-2037.

[200] Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. "The CoNLL-2014 shared task on grammatical error correction". In: *Proceedings of the Eighteenth Conference on Computational*

Natural Language Learning: Shared Task. 2014, pp. 1–14. DOI: 10.3115/v1/W14-1701.

[201] Sharon O'Brien. "Methodologies for measuring the correlations between post-editing effort and machine translatability". In: *Machine translation* 19 (2005), pp. 37–58. DOI: 10.1007/s10590-005-2467-1.

[202] Daisuke Okanohara and Jun'ichi Tsujii. "A discriminative language model with pseudo-negative samples". In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. 2007, pp. 73–80.

[203] Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. "GECToR – Grammatical Error Correction: Tag, Not Rewrite". In: *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics, 2020, pp. 163–170. DOI: 10.18653/v1/2020.bea-1.16.

[204] Kostiantyn Omelianchuk, Andrii Liubonko, Oleksandr Skurzhanskyi, Artem Chernodub, Oleksandr Korniienko, and Igor Samokhin. "Pillars of Grammatical Error Correction: Comprehensive Inspection Of Contemporary Approaches In The Era of Large Language Models". In: *Proceedings of the 19th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2024)*. Mexico City, Mexico: Association for Computational Linguistics, 2024, pp. 17–33. URL: https://aclanthology.org/2024.bea-1.3/.

[205] OpenAI. "GPT-4 Technical Report". In: *arXiv preprint arXiv:2303.08774* (2023). DOI: 10.48550/arXiv.2303.08774.

[206] OpenAI. "OpenAI GPT-5 System Card". In: *arXiv preprint arXiv:2601.03267* (2025). DOI: 10.48550/arXiv.2601.03267.

[207] Robert Östling, Katarina Gillholm, Murathan Kurfalı, Marie Mattson, and Mats Wirén. "Evaluation of Really Good Grammatical Error Correction". In: *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. Ed. by Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue. Torino, Italia: ELRA and ICCL, May 2024, pp. 6582–6593. DOI: 10.63317/2y78sksnckdg. URL: https://aclanthology.org/2024.lrec-main.584/.

[208] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. "Training language

models to follow instructions with human feedback". In: *Advances in Neural Information Processing Systems*. Vol. 35. Curran Associates, Inc., 2022, pp. 27730–27744. DOI: 10.52202/068431-2011.

[209] Gerhard Paaß and Sven Giesselbach. "Foundation models for speech, images, videos, and control". In: *Foundation models for natural language processing: Pre-trained language models integrating media*. Springer, 2023, pp. 313–382. DOI: 10.1007/978-3-031-23190-2_7.

[210] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. "Librispeech: an asr corpus based on public domain audio books". In: *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2015, pp. 5206–5210.

[211] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. "Bleu: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318. DOI: 10.3115/1073083.1073135.

[212] Taehee Park, Heejin Do, and Gary Lee. "Leveraging What's Overfixed: Post-Correction via LLM Grammatical Error Overcorrection". In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. 2025, pp. 28183–28195. DOI: 10.18653/v1/2025.emnlp-main.1431.

[213] Douglas B Paul and Janet Baker. "The design for the Wall Street Journal-based CSR corpus". In: *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*. 1992. DOI: 10.21437/icslp.1992-277.

[214] Sara M Pearsell and Oliver Niebuhr. "Lost in the Noise: Evaluating ASR Performance in Industrial and Environment Noise". In: *2025 IEEE 8th International Conference on Industrial Cyber-Physical Systems (ICPS)*. IEEE. 2025, pp. 1–5. DOI: 10.1109/icps65515.2025.11087895.

[215] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202.

[216] Mohammad Reza Peyghan, Saman Soleimani Roudi, Saeedreza Zouashkiani, Sajjad Amini, Fatemeh Rajabi, and Shahrokh Ghaemmaghami. "Non-Intrusive Automatic Speech Recognition Refinement: A Survey". In: *arXiv preprint arXiv:2508.07285* (2025). DOI: 10.48550/arXiv.2508.07285.

[217] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. "AdapterFusion: Non-Destructive Task Composition for Transfer Learning". In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Association for Computational Linguistics, 2021, pp. 487–503. DOI: 10.18653/v1/2021.eacl-main.39.

[218] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. "AdapterHub: A Framework for Adapting Transformers". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 2020, pp. 46–54. DOI: 10.18653/v1/2020.emnlp-demos.7.

[219] Aske Plaat, Annie Wong, Suzan Verberne, Joost Broekens, Niki Van Stein, and Thomas Bäck. "Multi-step reasoning with large language models, a survey". In: *ACM Computing Surveys* 58.6 (2025), pp. 1–35. DOI: 10.1145/3774896.

[220] Vassilis Plachouras, Charese Smiley, Hiroko Bretz, Ola Taylor, Jochen L Leidner, Dezhao Song, and Frank Schilder. "Interacting with financial data using natural language". In: *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. 2016, pp. 1121–1124. DOI: 10.1145/2911451.2911457.

[221] Mirko Plitt and François Masselot. "A Productivity Test of Statistical Machine Translation Post-Editing in a Typical Localisation Context." In: *Prague Bull. Math. Linguistics* 93 (2010), pp. 7–16. DOI: 10.2478/v10108-010-0010-x.

[222] Maja Popović, Arle Lommel, Aljoscha Burchardt, Eleftherios Avramidis, and Hans Uszkoreit. "Relations between different types of post-editing operations, cognitive effort and temporal effort". In: *Proceedings of the 17th annual conference of the european association for machine translation*. 2014, pp. 191–198.

[223] Rohit Prabhavalkar, Takaaki Hori, Tara N Sainath, Ralf Schlüter, and Shinji Watanabe. "End-to-end speech recognition: A survey". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 32 (2023), pp. 325–351. DOI: 10.1109/taslp.2023.3328283.

[224] Rohit Prabhavalkar, Tara N Sainath, Yonghui Wu, Patrick Nguyen, Zhifeng Chen, Chung-Cheng Chiu, and Anjuli Kannan. "Minimum word error rate training for attention-based sequence-to-sequence models". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 4839–4843. DOI: 10.1109/icassp.2018.8461809.

[225] Vineel Pratap, Qiantong Xu, Anuroop Sriram, Gabriel Synnaeve, and Ronan Collobert. "MLS: A Large-Scale Multilingual Dataset for Speech Research". In: *Proceedings of Interspeech 2020*. 2020, pp. 2757–2761. DOI: 10.21437/Interspeech.2020-2826.

[226] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. "Deep learning for audio signal processing". In: *IEEE Journal of Selected Topics in Signal Processing* 13.2 (2019), pp. 206–219. DOI: 10.1109/jstsp.2019.2908700.

[227] Ernest Pusateri, Anmol Walia, Anirudh Kashi, Bortik Bandyopadhyay, Nadia Hyder, Sayantan Mahinder, Raviteja Anantha, Daben Liu, and Sashank Gondala. "Retrieval augmented correction of named entity speech recognition errors". In: *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2025, pp. 1–5. DOI: 10.1109/icassp49660.2025.10888936.

[228] Mengyang Qiu, Qingyu Gao, Linxuan Yang, Yang Gu, Tran Minh Nguyen, Zihao Huang, and Jungyeul Park. "Chinese Grammatical Error Correction: A Survey". In: *arXiv preprint arXiv:2504.00977* (2025). DOI: 10.48550/arXiv.2504.00977.

[229] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. "Pre-trained models for natural language processing: A survey". In: *Science China technological sciences* 63.10 (2020), pp. 1872–1897. DOI: 10.1007/s11431-020-1647-3.

[230] Muhammad Reza Qorib, Alham Fikri Aji, and Hwee Tou Ng. "Efficient and interpretable grammatical error correction with mixture of experts". In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. 2024, pp. 17127–17138. DOI: 10.18653/v1/2024.findings-emnlp.997.

[231] Muhammad Reza Qorib, Seung-Hoon Na, and Hwee Tou Ng. "Frustratingly easy system combination for grammatical error correction". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2022, pp. 1964–1974. DOI: 10.18653/v1/2022.naacl-main.143.

[232] Muhammad Reza Qorib and Hwee Tou Ng. "System Combination via Quality Estimation for Grammatical Error Correction". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, 2023, pp. 12746–12759. DOI: 10.18653/v1/2023.emnlp-main.785.

[233] Qwen Team. "Qwen2.5 Technical Report". In: *arXiv preprint arXiv:2412.15115* (2024). DOI: 10.48550/arXiv.2412.15115.

[234] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. "Robust speech recognition via large-scale weak supervision". In: *International conference on machine learning*. PMLR. 2023, pp. 28492–28518.

[235] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. "Language Models are Unsupervised Multitask Learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[236] Srijith Radhakrishnan, Chao-Han Huck Yang, Sumeer Ahmad Khan, Rohit Kumar, Narsis A Kiani, David Gomez-Cabrero, and Jesper N Tegner. "Whispering llama: A cross-modal generative error correction framework for speech recognition". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 2023, pp. 10007–10016. DOI: 10.18653/v1/2023.emnlp-main.618.

[237] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. "Direct preference optimization: Your language model is secretly a reward model". In: *Advances in neural information processing systems* 36 (2023), pp. 53728–53741. DOI: 10.52202/075280-2338.

[238] Vipul Raheja, Dhruv Kumar, Ryan Koo, and Dongyeop Kang. "CoEdIT: Text Editing by Task-Specific Instruction Tuning". In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Singapore: Association for Computational Linguistics, 2023, pp. 5274–5291. DOI: 10.18653/v1/2023.findings-emnlp.350.

[239] Danish N Raja and Kaisar J Giri. "AUDIRE: a comprehensive review of speech recognition technologies–methods, uses, and challenges". In: *International Journal of Speech Technology* 28 (2025), pp. 903–929. DOI: 10.1007/s10772-025-10213-0.

[240] U. Reimer and U. Hahn. "Text condensation as knowledge base abstraction". In: *[1988] Proceedings. The Fourth Conference on Artificial Intelligence Applications*. 1988, pp. 338–344. DOI: 10.1109/CAIA.1988.196128.

[241] Ehud Reiter and Robert Dale. "Building applied natural language generation systems". In: *Natural Language Engineering* 3.1 (1997), pp. 57–87. DOI: 10.1017/s1351324997001502.

[242] Ehud Reiter, Somayajulu Sripada, Jim Hunter, Jin Yu, and Ian Davy. "Choosing words in computer-generated weather forecasts". In: *Artificial Intelligence* 167.1-2 (2005), pp. 137–169. DOI: 10.1016/j.artint.2005.06.006.

[243] Stephen Richardson and Lisa Braden-Harder. "The experience of developing a large-scale natural language processing system: Critique". In: *Natural Language Processing: The PLNLP Approach*. Springer, 1993, pp. 77–89. DOI: 10.1007/978-1-4615-3170-8_7.

[244] Stephen Robertson and Hugo Zaragoza. "The Probabilistic Relevance Framework: BM25 and Beyond". In: *Foundations and Trends in Information Retrieval* 3.4 (2009), pp. 333–389. DOI: 10.1561/1500000019.

[245] Ana Rodrigues, Rita Santos, Jorge Abreu, Pedro Beça, Pedro Almeida, and Sílvia Fernandes. "Analyzing the performance of ASR systems: The effects of noise, distance to the device, age and gender". In: *Proceedings of the XX International Conference on Human Computer Interaction*. 2019, pp. 1–8.

[246] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. "A primer in BERTology: What we know about how BERT works". In: *Transactions of the association for computational linguistics* 8 (2020), pp. 842–866. DOI: 10.1162/tacl_a_00349.

[247] Sascha Rothe, Jonathan Mallinson, Eric Malmi, Sebastian Krause, and Aliaksei Severyn. "A Simple Recipe for Multilingual Grammatical Error Correction". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Ed. by Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli. Online: Association for Computational Linguistics, Aug. 2021, pp. 702–707. DOI: 10.18653/v1/2021.acl-short.89. URL: https://aclanthology.org/2021.acl-short.89.

[248] Anthony Rousseau, Paul Deléglise, and Yannick Esteve. "TED-LIUM: an automatic speech recognition dedicated corpus." In: *LREC*. 2012, pp. 125–129. DOI: 10.63317/48xafxbrk3qa.

[249] Alla Rozovskaya, Kai-Wei Chang, Mark Sammons, Dan Roth, and Nizar Habash. "The Illinois-Columbia system in the CoNLL-2014 shared task". In: *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. 2014, pp. 34–42. DOI: 10.3115/v1/W14-1704.

[250] Alla Rozovskaya and Dan Roth. "Annotating ESL Errors: Challenges and Rewards". In: *Proceedings of the NAACL HLT 2010 Fifth Workshop on Innovative Use of NLP for Building Educational Applications*. Ed. by Joel Tetreault, Jill Burstein, and Claudia Leacock. Los Angeles, California: Association for Computational Linguistics, June 2010, pp. 28–36. URL: https://aclanthology.org/W10-1004/.

[251] Alla Rozovskaya and Dan Roth. "How good (really) are grammatical error correction systems?" In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. 2021, pp. 2686–2698. DOI: 10.18653/v1/2021.eacl-main.231.

[252] Alla Rozovskaya, Mark Sammons, Joshua Gioja, and Dan Roth. "University of Illinois system in HOO text correction shared task". In: *Proceedings of the Generation Challenges Session at the 13th European Workshop on Natural Language Generation*. Association for Computational Linguistics, 2011, pp. 263–266. URL: https://aclanthology.org/W11-2843/.

[253] Rithik Sachdev, Zhong-Qiu Wang, and Chao-Han Huck Yang. "Evolutionary prompt design for llm-based post-asr error correction". In: *2025 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE. 2025, pp. 1–5. DOI: 10.1109/sips66314.2025.11261248.

[254] Hardik B Sailor, Ankur T Patil, and Hemant A Patil. "Advances in Low Resource ASR: A Deep Learning Perspective." In: *SLTU*. 2018, pp. 15–19. DOI: 10.21437/sltu.2018-4.

[255] Keisuke Sakaguchi and Benjamin Van Durme. "Efficient Online Scalar Annotation with Bounded Support". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, 2018, pp. 208–218. DOI: 10.18653/v1/P18-1020.

[256] Yusuke Sakai, Takumi Goto, and Taro Watanabe. "IMPARA-GED: Grammatical Error Detection is Boosting Reference-free Grammatical Error Quality Estimator". In: *Findings of the Association for Computational Linguistics: ACL 2025*. Vienna, Austria: Association for Computational Linguistics, 2025, pp. 25647–25654. DOI: 10.18653/v1/2025.findings-acl.1315.

[257] Jost Schatzmann, Blaise Thomson, and Steve Young. "Error simulation for training statistical dialogue systems". In: *2007 IEEE Workshop on Automatic Speech Recognition & Understanding (ASRU)*. IEEE. 2007, pp. 526–531. DOI: 10.1109/asru.2007.4430167.

[258] Holger Schielzeth. "Simple means to improve the interpretability of regression coefficients". In: *Methods in Ecology and Evolution* 1.2 (2010), pp. 103–113. DOI: 10.1111/j.2041-210X.2010.00012.x.

[259] Dorothy Senez. "Post-editing service for machine translation users at the European Commission". In: *Proceedings of Translating and the Computer 20*. 1998.

[260] Rico Sennrich, Alexandra Birch, Anna Currey, Ulrich Germann, Barry Haddow, Kenneth Heafield, Antonio Valerio Miceli Barone, and Philip Williams. "The University of Edinburgh's Neural MT Systems for WMT17". In: *Proceedings of the Second Conference on Machine Translation*. Ed. by Ondřej Bojar, Christian Buck, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, and Julia Kreutzer. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 389–399. DOI: 10.18653/v1/W17-4739. URL: https://aclanthology.org/W17-4739/.

[261] Claude E Shannon. "Prediction and entropy of printed English". In: *Bell system technical journal* 30.1 (1951), pp. 50–64. DOI: 10.1002/j.1538-7305.1951.tb01366.x.

[262] Minghao Shao, Abdul Basit, Ramesh Karri, and Muhammad Shafique. "Survey of different large language model architectures: Trends, benchmarks, and challenges". In: *IEEE access* 12 (2024), pp. 188664–188706. DOI: 10.1109/access.2024.3482107.

[263] Shuqian Sheng, Yi Xu, Tianhang Zhang, Zanwei Shen, Luoyi Fu, Jiaxin Ding, Lei Zhou, Xiaoying Gan, Xinbing Wang, and Chenghu Zhou. "RepEval: Effective text evaluation with LLM representation". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 2024, pp. 7019–7033. DOI: 10.18653/v1/2024.emnlp-main.398.

[264] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489. DOI: 10.1038/nature16961.

[265] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018). Earlier version: arXiv:1712.01815, "Mastering Chess and Shogi by

Self-Play with a General Reinforcement Learning Algorithm", pp. 1140–1144. DOI: 10.1126/science.aar6404.

[266] Matthew Snover, Bonnie Dorr, Rich Schwartz, Linnea Micciulla, and John Makhoul. "A Study of Translation Edit Rate with Targeted Human Annotation". In: *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*. Cambridge, Massachusetts, USA: Association for Machine Translation in the Americas, Aug. 2006, pp. 223–231. URL: https://aclanthology.org/2006.amta-papers.25/.

[267] Yixiao Song, Kalpesh Krishna, Rajesh Bhatt, Kevin Gimpel, and Mohit Iyyer. "GEE! Grammar Error Explanation with Large Language Models". In: *Findings of the Association for Computational Linguistics: NAACL 2024*. Mexico City, Mexico: Association for Computational Linguistics, 2024, pp. 754–781. DOI: 10.18653/v1/2024.findings-naacl.49.

[268] Alexey Sorokin. "Improved grammatical error correction by ranking elementary edits". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 2022, pp. 11416–11429. DOI: 10.18653/v1/2022.emnlp-main.785.

[269] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. "Curriculum learning: A survey". In: *International Journal of Computer Vision* 130.6 (2022), pp. 1526–1565. DOI: 10.1007/s11263-022-01611-x.

[270] Lucia Specia. "Exploiting Objective Annotations for Minimising Translation Post-editing Effort". In: *Proceedings of the 15th Annual Conference of the European Association for Machine Translation*. Ed. by Mikel L. Forcada, Heidi Depraetere, and Vincent Vandeghinste. Leuven, Belgium: European Association for Machine Translation, May 2011. URL: https://aclanthology.org/2011.eamt-1.12/.

[271] Lucia Specia, Najeh Hajlaoui, Catalina Hallett, and Wilker Aziz. "Predicting machine translation adequacy". In: *Proceedings of Machine Translation Summit XIII: Papers*. 2011.

[272] Lucia Specia, Marco Turchi, Nicola Cancedda, Nello Cristianini, and Marc Dymetman. "Estimating the sentence-level quality of machine translation systems". In: *Proceedings of the 13th Annual conference of the European Association for Machine Translation*. 2009.

[273] Somayajulu Sripada, Ehud Reiter, and Lezan Hawizy. "Evaluation of an NLG System using Post-Edit Data: Lessons Learnt". In: *Proceedings of the Tenth European Workshop on Natural Language Generation (ENLG-05)*. Ed. by Graham Wilcock, Kristiina Jokinen, Chris Mellish, and Ehud Reiter. Aberdeen, Scotland: Association

for Computational Linguistics, Aug. 2005. URL: https://aclanthology.org/W05-1615/.

[274] Sandeep Subramanian, Adam Trischler, Yoshua Bengio, and Christopher J. Pal. "Learning General Purpose Distributed Sentence Representations via Large Scale Multi-task Learning". In: *International Conference on Learning Representations (ICLR)*. 2018. URL: https://openreview.net/forum?id=B18WgG-CZ.

[275] Xin Sun, Tao Ge, Furu Wei, and Houfeng Wang. "Instantaneous grammatical error correction with shallow aggressive decoding". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2021, pp. 5937–5947. DOI: 10.18653/v1/2021.acl-long.462.

[276] Raymond Hendy Susanto, Peter Phandi, and Hwee Tou Ng. "System combination for grammatical error correction". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 951–962. DOI: 10.3115/v1/d14-1102.

[277] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc., 2014, pp. 3104–3112.

[278] Yujin Takahashi, Masahiro Kaneko, Masato Mita, and Mamoru Komachi. "ProQE: Proficiency-wise Quality Estimation dataset for Grammatical Error Correction". In: *Proceedings of the Thirteenth Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, 2022, pp. 5994–6000. DOI: 10.63317/2buebhwn2b5i. URL: https://aclanthology.org/2022.lrec-1.644/.

[279] Tomohiro Tanaka, Ryo Masumura, Hirokazu Masataki, and Yushi Aono. "Neural Error Corrective Language Models for Automatic Speech Recognition." In: *Interspeech*. 2018, pp. 401–405. DOI: 10.21437/interspeech.2018-1430.

[280] Maksym Tarnavskyi, Artem Chernodub, and Kostiantyn Omelianchuk. "Ensembling and Knowledge Distilling of Large Sequence Taggers for Grammatical Error Correction". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2022, pp. 3842–3852. DOI: 10.18653/v1/2022.acl-long.266.

[281] Midori Tatsumi. "Correlation between automatic evaluation metric scores, post-editing speed, and some other factors". In: *Proceedings of Machine Translation Summit XII: Posters*. 2009.

[282] Midori Tatsumi and Johann Roturier. "Source text characteristics and technical and temporal post-editing effort: what is their relationship". In: *Proceedings of the Second Joint EM+/CNGL Workshop: Bringing MT to the User: Research on Integrating MT in the Translation Industry*. 2010, pp. 43–52.

[283] Irina P Temnikova. "Cognitive Evaluation Approach for a Controlled Language Post–Editing Experiment." In: *LREC*. 2010. DOI: 10.63317/4tebu7o9dxk2.

[284] Joel Tetreault and Martin Chodorow. "Native judgments of non-native usage: Experiments in preposition error detection". In: *Coling 2008: Proceedings of the workshop on human judgements in computational linguistics*. 2008, pp. 24–32. DOI: 10.3115/1611628.1611633.

[285] Joel Tetreault, Martin Chodorow, and Nitin Madnani. "Bucking the trend: improved evaluation and annotation practices for ESL error detection systems". In: *Language Resources and Evaluation* 48 (2014), pp. 5–31. DOI: 10.1007/s10579-013-9243-2.

[286] Arda Tezcan, Veronique Hoste, and Lieve Macken. "Detecting Grammatical Errors in Machine Translation Output Using Dependency Parsing and Treebank Querying". In: *Proceedings of the 19th Annual Conference of the European Association for Machine Translation*. 2016, pp. 203–217. URL: https://aclanthology.org/W16-3409/.

[287] Arda Tezcan, Véronique Hoste, and Lieve Macken. "Estimating post-editing time using a gold-standard set of machine translation errors". In: *Computer Speech & Language* 55 (2019), pp. 120–144. DOI: 10.1016/j.csl.2018.10.005.

[288] Henry Thompson. "Strategy and tactics: A model for language production". In: *Papers from the... Regional Meeting. Chicago Ling. Soc. Chicago, Ill.* Vol. 13. 1977, pp. 651–668.

[289] Shubham Toshniwal, Anjuli Kannan, Chung-Cheng Chiu, Yonghui Wu, Tara N. Sainath, and Karen Livescu. "A Comparison of Techniques for Language Model Integration in Encoder-Decoder Speech Recognition". In: *2018 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2018, pp. 369–375. DOI: 10.1109/SLT.2018.8639038.

[290] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. "LLaMA: Open and Efficient Foundation Language Models". In: *arXiv preprint arXiv:2302.13971* (2023). DOI: 10.48550/arXiv.2302.13971.

[291] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. "Llama 2: Open Foundation and Fine-Tuned Chat Models". In: *arXiv preprint arXiv:2307.09288* (2023). DOI: 10.48550/arXiv.2307.09288.

[292] Ada Defne Tur, Adel Moumen, and Mirco Ravanelli. "Progres: Prompted generative rescoring on asr n-best". In: *2024 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2024, pp. 600–607. DOI: 10.1109/slt61566.2024.10832194.

[293] Alan M Turing. "Computing machinery and intelligence". In: *Parsing the Turing Test*. Springer, 2009, pp. 23–65. DOI: 10.1007/978-1-4020-6710-5_3.

[294] Ankit Vadehra, Bill Johnson, Gene Saunders, and Pascal Poupart. "Time Is Effort: Estimating Human Post-Editing Time for Grammar Error Correction Tool Evaluation". In: *Proceedings of the Fourth Workshop on Bridging Human-Computer Interaction and Natural Language Processing (HCI+NLP)*. Suzhou, China: Association for Computational Linguistics, Nov. 2025, pp. 178–196. DOI: 10.18653/v1/2025.hcinlp-1.15. URL: https://aclanthology.org/2025.hcinlp-1.15/.

[295] Jayashri Vajpai and Avnish Bora. "Industrial applications of automatic speech recognition systems". In: *International Journal of Engineering Research and Applications* 6.3 (2016), pp. 88–95.

[296] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017, pp. 5998–6008.

[297] Lucas Nunes Vieira. "Cognitive effort and different task foci in post-editing of machine translation: A think-aloud study". In: *Across Languages and Cultures* 18.1 (2017), pp. 79–105. DOI: 10.1556/084.2017.18.1.4.

[298] Lucas Nunes Vieira. "Indices of cognitive effort in machine translation post-editing". In: *Machine translation* 28.3 (2014), pp. 187–216. DOI: 10.1007/s10590-014-9156-x.

[299] Joachim Wagner, Jennifer Foster, and Josef van Genabith. "A comparative evaluation of deep and shallow approaches to the automatic detection of common grammatical errors". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. 2007, pp. 112–121.

[300] Longshaokan Wang, Maryam Fazel-Zarandi, Aditya Tiwari, Spyros Matsoukas, and Lazaros Polymenakos. "Data augmentation for training dialog models robust to speech recognition errors". In: *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*. 2020, pp. 63–70. DOI: 10.18653/v1/2020.nlp4convai-1.8.

[301] Peilu Wang, Zhongye Jia, and Hai Zhao. "Grammatical error detection and correction using a single maximum entropy model". In: *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. 2014, pp. 74–82. DOI: 10.3115/v1/W14-1710.

[302] Xiaoqiang Wang, Yanqing Liu, Jinyu Li, Veljko Miljanic, Sheng Zhao, and Hosam Khalil. "Towards contextual spelling correction for customization of end-to-end speech recognition systems". In: *IEEE/ACM transactions on audio, speech, and language processing* 30 (2022), pp. 3089–3097. DOI: 10.1109/taslp.2022.3205753.

[303] Xin Wang, Yudong Chen, and Wenwu Zhu. "A survey on curriculum learning". In: *IEEE transactions on pattern analysis and machine intelligence* 44.9 (2021), pp. 4555–4576. DOI: 10.1109/tpami.2021.3069908.

[304] Yiming Wang, Longyue Wang, Xiaodong Zeng, Derek F Wong, Lidia S Chao, and Yi Lu. "Factored statistical machine translation for grammatical error correction". In: *Proceedings of the eighteenth conference on computational natural language learning: Shared task*. 2014, pp. 83–90. DOI: 10.3115/v1/W14-1711.

[305] Yixuan Wang, Baoxin Wang, Yijun Liu, Qingfu Zhu, Dayong Wu, and Wanxiang Che. "Improving grammatical error correction via contextual data augmentation". In: *Findings of the Association for Computational Linguistics: ACL 2024*. 2024, pp. 10898–10910. DOI: 10.18653/v1/2024.findings-acl.647.

[306] Yu Wang, Yuelin Wang, Jie Liu, and Zhuo Liu. "A comprehensive survey of grammar error correction". In: *arXiv preprint arXiv:2005.06600* (2020). DOI: 10.48550/arXiv.2005.06600.

[307] Zhichao Wang, Kiran Ramnath, Bin Bi, Shiva Kumar Pentyala, Sougata Chaudhuri, Shubham Mehrotra, Zixu Zhu, Xiang-Bo Mao, Sitaram Asur, and Na Cheng. "A Comprehensive Survey of LLM Alignment Techniques: RLHF, RLAIF, PPO, DPO and More". In: *arXiv preprint arXiv:2407.16216* (2024). v1 title; later arXiv revisions (2026) are retitled "Reinforcement Learning for LLM Post-Training: A Survey". DOI: 10.48550/arXiv.2407.16216.

[308] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. "Finetuned Language Models Are Zero-Shot Learners". In: *International Conference on Learning Representations (ICLR)*. 2022. URL: https://openreview.net/forum?id=gEZrGCozdqR.

[309] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. "Emergent Abilities of Large Language Models". In: *Transactions on Machine Learning Research* (2022). URL: https://openreview.net/forum?id=yzkSU5zdwD.

[310] Joseph Weizenbaum. "ELIZA—a computer program for the study of natural language communication between man and machine". In: *Communications of the ACM* 9.1 (1966), pp. 36–45. DOI: 10.1145/365153.365168.

[311] Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. "CCNet: Extracting high quality monolingual datasets from web crawl data". In: *Proceedings of the twelfth language resources and evaluation conference*. 2020, pp. 4003–4012.

[312] Jian-Cheng Wu, Tzu-Hsi Yen, Jim Chang, Guan-Cheng Huang, Jimmy Chang, Hsiang-Ling Hsu, Yu-Wei Chang, and Jason S. Chang. "NTHU at the CoNLL-2014 shared task". In: *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. 2014, pp. 91–95. DOI: 10.3115/v1/W14-1712.

[313] Jiayang Wu, Wensheng Gan, Zefeng Chen, Shicheng Wan, and Philip S Yu. "Multimodal large language models: A survey". In: *2023 IEEE International Conference on Big Data (BigData)*. IEEE. 2023, pp. 2247–2256. DOI: 10.1109/bigdata59044.2023.10386743.

[314] Liyan Xu, Yile Gu, Jari Kolehmainen, Haidar Khan, Ankur Gandhe, Ariya Rastrow, Andreas Stolcke, and Ivan Bulyko. "Rescorebert: Discriminative speech recognition rescoring with bert". In: *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2022, pp. 6117–6121. DOI: 10.1109/icassp43922.2022.9747118.

[315] Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. "ByT5: Towards a token-free future with pre-trained byte-to-byte models". In: *Transactions of the Association for Computational Linguistics* 10 (2022), pp. 291–306. DOI: 10.1162/tacl_a_00461.

[316] An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, Junyang Lin, Kai Dang, Kexin Yang, Le Yu, Mei Li, Minmin Sun, Qin Zhu, Rui Men, Tao He, Weijia Xu, Wenbiao Yin, Wenyuan Yu, Xiafei Qiu, Xingzhang Ren, Xinlong Yang, Yong Li, Zhiying Xu, and Zipeng Zhang. "Qwen2.5-1M Technical Report". In: *CoRR* abs/2501.15383 (2025). DOI: 10.48550/ARXIV.2501.15383. arXiv: 2501.15383. URL: https://doi.org/10.48550/arXiv.2501.15383.

[317] Chao-Han Huck Yang, Yile Gu, Yi-Chieh Liu, Shalini Ghosh, Ivan Bulyko, and Andreas Stolcke. "Generative speech recognition error correction with large language models and task-activating prompting". In: *2023 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE. 2023, pp. 1–8. DOI: 10.1109/asru57964.2023.10389673.

[318] Chao-Han Huck Yang, Taejin Park, Yuan Gong, Yuanchao Li, Zhehuai Chen, Yen-Ting Lin, Chen Chen, Yuchen Hu, Kunal Dhawan, Piotr Żelasko, Chao Zhang, Yun-Nung Chen, Yu Tsao, Jagadeesh Balam, Boris Ginsburg, Sabato Marco Siniscalchi, Eng Siong Chng, Peter Bell, Catherine Lai, Shinji Watanabe, and Andreas Stolcke. "Large Language Model Based Generative Error Correction: A Challenge and Baselines for Speech Recognition, Speaker Tagging, and Emotion Recognition". In: *2024 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2024, pp. 371–378. DOI: 10.1109/SLT61566.2024.10832176.

[319] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu. "Harnessing the power of llms in practice: A survey on chatgpt and beyond". In: *ACM Transactions on Knowledge Discovery from Data* 18.6 (2024), pp. 1–32. DOI: 10.1145/3649506.

[320] Jingyuan Yang, Rongjun Li, and Wei Peng. "ASR Error Correction with Constrained Decoding on Operation Prediction". In: *Proceedings of Interspeech 2022*. 2022, pp. 3874–3878. DOI: 10.21437/Interspeech.2022-660.

[321] Xiangli Yang, Zixing Song, Irwin King, and Zenglin Xu. "A survey on deep semi-supervised learning". In: *IEEE transactions on knowledge and data engineering* 35.9 (2022), pp. 8934–8954. DOI: 10.1109/TKDE.2022.3220219.

[322] Helen Yannakoudakis, Øistein E Andersen, Ardeshir Geranpayeh, Ted Briscoe, and Diane Nicholls. "Developing an automated writing placement system for ESL learners". In: *Applied Measurement in Education* 31.3 (2018), pp. 251–267. DOI: 10.1080/08957347.2018.1464447.

[323] Helen Yannakoudakis, Ted Briscoe, and Ben Medlock. "A New Dataset and Method for Automatically Grading ESOL Texts". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Dekang Lin, Yuji Matsumoto, and Rada Mihalcea. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 180–189. URL: https://aclanthology.org/P11-1019/.

[324] Helen Yannakoudakis, Marek Rei, Øistein E Andersen, and Zheng Yuan. "Neural sequence-labelling models for grammatical error correction". In: *Proceedings of the 2017 conference on empirical methods in natural language processing*. Association for Computational Linguistics (ACL). 2017, pp. 2795–2806. DOI: 10.18653/v1/d17-1297.

[325] Michihiro Yasunaga, Jure Leskovec, and Percy Liang. "LM-Critic: Language Models for Unsupervised Grammatical Error Correction". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021, pp. 7752–7763. DOI: 10.18653/v1/2021.emnlp-main.611.

[326] Jingheng Ye, Shang Qin, Yinghui Li, Xuxin Cheng, Libo Qin, Hai-Tao Zheng, Ying Shen, Peng Xing, Zishan Xu, Guo Cheng, and Wenhao Jiang. "EXCGEC: A Benchmark for Edit-Wise Explainable Chinese Grammatical Error Correction". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 24. 2025, pp. 25678–25686. DOI: 10.1609/aaai.v39i24.34759.

[327] Jingheng Ye, Shang Qin, Yinghui Li, Hai-Tao Zheng, Shen Wang, and Qingsong Wen. "Corrections Meet Explanations: A Unified Framework for Explainable Grammatical Error Correction". In: *arXiv preprint arXiv:2502.15261* (2025). DOI: 10.48550/arXiv.2502.15261.

[328] Na Ye, Ling Jiang, Dandan Ma, Yingxin Zhang, Sanyuan Zhao, and Dongfeng Cai. "Predicting post-editing effort for English-Chinese neural machine translation". In: *2021 International Conference on Asian Language Processing (IALP)*. IEEE. 2021, pp. 154–158. DOI: 10.1109/IALP54817.2021.9675227.

[329] Heuiyeen Yeen, Minju Kim, and Myoung-Wan Koo. "I Learned Error, I Can Fix It!: A Detector-Corrector Structure for ASR Error Calibration." In: *INTERSPEECH*. 2023, pp. 2693–2697. DOI: 10.21437/interspeech.2023-2475.

[330] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. "Comparative study of CNN and RNN for natural language processing". In: *arXiv preprint arXiv:1702.01923* (2017). DOI: 10.48550/arXiv.1702.01923.

[331] Ryoma Yoshimura, Masahiro Kaneko, Tomoyuki Kajiwara, and Mamoru Komachi. "SOME: Reference-less sub-metrics optimized for manual evaluations of grammatical error correction". In: *Proceedings of the 28th International Conference on Computational Linguistics*. 2020, pp. 6516–6522. DOI: 10.18653/v1/2020.coling-main.573.

[332] Weizhe Yuan, Graham Neubig, and Pengfei Liu. "Bartscore: Evaluating generated text as text generation". In: *Advances in neural information processing systems* 34 (2021), pp. 27263–27277.

[333] Zheng Yuan, Shiva Taslimipoor, Christopher Davis, and Christopher Bryant. "Multi-class grammatical error detection for correction: A tale of two systems". In: *Proceedings of the 2021 conference on empirical methods in natural language processing*. 2021, pp. 8722–8736. DOI: 10.18653/v1/2021.emnlp-main.687.

[334] Anna Zaretskaya, Mihaela Vela, Gloria Corpas Pastor, and Miriam Seghiri. "Measuring post-editing time and effort for different types of machine translation errors". In: (2016).

[335] Mohammad Zeineldeen, Aleksandr Glushko, Wilfried Michel, Albert Zeyer, Ralf Schlüter, and Hermann Ney. "Investigating Methods to Improve Language Model Integration for Attention-Based Encoder-Decoder ASR Models". In: *Proceedings of Interspeech 2021*. 2021, pp. 2856–2860. DOI: 10.21437/Interspeech.2021-1255.

[336] Bolin Zhang, Jiahao Wang, Qianlong Du, Jiajun Zhang, Zhiying Tu, and Dianhui Chu. "A Survey on Data Selection for LLM Instruction Tuning". In: *Journal of Artificial Intelligence Research* 83 (2025). Earlier version: arXiv:2402.05123 (2024). DOI: 10.1613/jair.1.17625.

[337] Ding Zhang, Yangning Li, Lichen Bai, Hao Zhang, Yinghui Li, Haiye Lin, Hai-Tao Zheng, Xin Su, and Zifei Shan. "Loss-aware curriculum learning for chinese grammatical error correction". In: *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2025, pp. 1–5. DOI: 10.1109/icassp49660.2025.10889130.

[338] Hanqing Zhang, Haolin Song, Shaoyu Li, Ming Zhou, and Dawei Song. "A survey of controllable text generation using transformer-based pre-trained language models". In: *ACM Computing Surveys* 56.3 (2023), pp. 1–37. DOI: 10.1145/3617680.

[339] Jing Zhang. "Grammar Error Correction Based on Deep Learning: Incorporating Convolutional Neural Networks". In: *2025 2nd International Conference on Intelligent Computing and Robotics (ICICR)*. IEEE. 2025, pp. 104–108. DOI: 10.1109/icicr65456.2025.00025.

[340] Longkai Zhang and Houfeng Wang. "A unified framework for grammar error correction". In: *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. 2014, pp. 96–102. DOI: 10.3115/v1/W14-1713.

[341] Shaohua Zhang, Haoran Huang, Jicong Liu, and Hang Li. "Spelling error correction with soft-masked BERT". In: *Proceedings of the 58th annual meeting of the association for computational linguistics*. 2020, pp. 882–890. DOI: 10.18653/v1/2020.acl-main.82.

[342] Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Guoyin Wang, and Fei Wu. "Instruction Tuning for Large Language Models: A Survey". In: *ACM Computing Surveys* 58.7 (2026), pp. 1–36. DOI: 10.1145/3777411.

[343] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Zican Dong, Yupeng Hou, Beichen Zhang, Yingqian Min, Junjie Zhang, Peiyu Liu, Xiaolei Wang, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Yiwen Hu, Jian-Yun Nie, and Ji-Rong Wen. "A Survey of Large Language Models". In: *Frontiers of Computer Science* 20.12 (2026), p. 2012627. DOI: 10.1007/s11704-026-60308-3.

[344] Wei Zhao, Liang Wang, Kewei Shen, Ruoyu Jia, and Jingming Liu. "Improving grammatical error correction via pre-training a copy-augmented architecture with unlabeled data". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2019, pp. 156–165. DOI: 10.18653/v1/n19-1014.

[345] Yun Zhao, Xuerui Yang, Jinchao Wang, Yongyu Gao, Chao Yan, and Yuanfu Zhou. "BART Based Semantic Correction for Mandarin Automatic Speech Recognition System". In: *Proceedings of Interspeech 2021*. 2021, pp. 2017–2021. DOI: 10.21437/Interspeech.2021-739.

[346] Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, Hao Peng, Jianxin Li, Jia Wu, Ziwei Liu, Pengtao Xie, Caiming Xiong, Jian Pei, Philip S. Yu, and Lichao Sun. "A comprehensive survey on pretrained foundation models: A history from BERT to ChatGPT". In: *International Journal of Machine Learning and Cybernetics* 16.12 (2025), pp. 9851–9915. DOI: 10.1007/s13042-024-02443-6.

[347] Linchen Zhu, Wenjie Liu, Linquan Liu, and Edward Lin. "Improving asr error correction using n-best hypotheses". In: *2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE. 2021, pp. 83–89. DOI: 10.1109/asru51503.2021.9688210.

[348] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 19–27. DOI: 10.1109/iccv.2015.11.

APPENDICES

Appendix A

Non-Neural Based Text Generation Techniques

A summarization of historical NLG techniques [241] show that an NLG system can be broken down into the following components:

Content Determination Determines which information to focus on in the input sentence/data.

Text Structuring Determines the sequential ordering of the content information.

Sentence Aggregation Choosing the information to present in each output sentence.

Lexicalization Finding the words and phrases/template to generate output.

Referring Selecting the words/phrases to identify entities and objects.

Linguistic Realization Combining all information to form structurally sound output sentences.

The rule based generation techniques are often task/application specific in terms of the output template and information to present. The model goes through a decision making step where the information from the structured data/input text is chosen to focus on and then appropriate output information that corresponds to the input and the application is selected. Finally, templates and grammatical rule/parsing structure is used to generate the final output sentence by the NLG system.

The NLG task can be broken down into *late* and *early* tasks. The distinction between the two tasks allows us to decide what information to convey in the output and in which order. This type of planning can be referred back to Thompson [288]. Even though the literature on rule based NLG techniques is wide and varied it is not generalized in terms of approaches used and is highly domain and task specific. As a result, covering all of the approaches is out of the scope of this document. For a detailed description of the domain, I refer to the comprehensive summary presented by Gatt and Krahmer [78] and Reiter and Dale [241].

Similarly, for text-to-text generation systems, the approaches can be broken down into Shallow and Deep Generation techniques [184]. For the task of designing an automatic summarization system, the shallow technique primarily does sentence extraction. The objective is to select sentences from text that can best encapsulate the text and serve as a summarization [147]. There are a few limitations to this approach. Primarily, a single sentence or a set of sentences might not be exhaustive enough to cover all of the important information in the whole document. Also, it might lead to redundant information. On the other hand, selecting or generating other sentences can pose an out-of-context issue. To mitigate some of these challenges certain *smoothing* factors are selected. A sentence window is often selected to be able to refer surrounding sentences/text to capture and rearrange more information. Also, it can help us in adjusting reference and fix the generation template [185].

Deeper text generation techniques are primarily used for abstract generation and require external domain information and generation rules. They can refer to external background concepts and facts that might not be present in the original text. Based on the literature, the deep generation techniques can be divided in *template* and *concept* approaches. The template based generation techniques utilize the source documents to extract information and derive a generation layout appropriate for the task. After a template is learnt for the task, the extracted information is transformed to fill the template [187]. Concept based techniques utilize different extracted content information which act as abstract categories. The extracted information can be keywords from documents, supporting facts from external knowledge bases, etc. All this information can be used to obtain concept hierarchies and form a structure for text generation. This structure can be used to learn a generalization of concepts for text generation [240, 167].

Another popular application of text generation systems is for the task of dialogue systems and conversational agents. The objective of a dialogue system is to generate an output in response to a user specified input. Dialogue systems have been fairly researched in Artificial Intelligence as variants of the Turing test problem [293, 47]. The research done in this domain is based on statistical language modeling, lexical/contextual rule based

systems [310, 46], and, as a slot filling problem [20].

Jurafsky et al. categorize chatbots in two primary categories: rule-based and corpus-based systems [123]. Rule based systems were some of the initial chatbots. ELIZA [310] and PARRY [46] were among the first rule based conversational systems. ELIZA was designed to utilize and mimic the user statements and to generate the response using directives provided in the form of 'scripts'. ELIZA was tested using a script that emulates a Rogerian psychotherapist. Modern chatbots like ALICE also utilize a lot of the features of ELIZA.

PARRY was designed to study schizophrenia and utilized conditioning on top of ELIZA like regular expressions. If the system detects 'anger' or 'fear' it can modify its output based on the intensity of the affect.

Corpus based chatbots can be categorized in two categories: Information Retrieval (IR) based and sequence-to-sequence (Seq2Seq) trained chatbots. An IR based chatbot utilizes statistical techniques to perform some sort of ranking between a new user source utterance and a corpus of source-target utterance pairs, returning the target response utterance from the corpus that matches the new user utterance the best. I explain Seq2Seq-based techniques in the next section.

However, due to the availability of parallel bi-text training corpora and sufficient computing power, the rule based techniques were overshadowed by Statistical text generation techniques that utilize probability, co-occurrence and word frequency scores learnt on the training corpora for selecting the output text to generate. This trained generative model is referred to as a language model.

A statistical language model provides a probability distribution over a set of words (which is predetermined and referred to as the Vocabulary) and can be used to predict a succession of words. At the core of the language model is the Bayes rule:

$$P(w|C) = \frac{P(C|w)P(w)}{P(C)} \tag{A.1}$$

using this formula we can select a word w given a context C that gives the best probability score. Based on the input and the application, a statistical language model can utilize different input features. For example tasks like topic modeling and classification require sentences/paragraphs/documents as input whereas the task of text generation deals with words.

In terms of text generation, if we assume every word in a sentence to be a feature and a sentence $S = (w_1, w_2, ..., w_m)$ to be a sequential ordering of m words, then the probability

score for the sentence can be decomposed to:

$$\begin{aligned} P(w_1, w_2, ..., w_m) &= P(w_1)P(w_2|w_1)...P(w_m|w_{m-1}, ..., w_1) \\ &= \prod_{i=1}^{m} P(w_i|w_{i-1}, ..., w_1) \end{aligned} \tag{A.2}$$

However, for long range sentences solving this equation and estimating the probability score of a sentence with an absolute context becomes non-trivial, improbable and intractable. As a result, N-Gram Language Models were suggested and become popular. In an N-Gram LM setting we specify a predefined context size of N. This makes the word prediction at any given time dependent on the preceding $N-1$ words. This can be thought of as an Nth-order Markov model where the probability of predicting a word at any time step i dependent on the $i-1$ previous words can be estimated by considering the previous $N-1$ words or observations. Using this assumption, Equation (A.2) can be approximated by:

$$P(w_1, w_2, ..., w_m) \approx \prod_{i=1}^{m} P(w_i|w_{i-(n-1)}, ..., w_{i-1}) \tag{A.3}$$

where n is the N-gram order.

Utilizing an N-gram LM for text generation is based on the work done by Shannon [261]. In his work, Shannon was trying to predict the difference in the ability of humans and a statistical model to predict the next word based on a predefined n-gram character context. The results obtained show that increasing the context, i.e., value of n, results in a significant degradation of performance of statistical models. As an extension, this comparative evaluation between humans and an n-gram LM is referred to as the Shannon game [180, 155] and is considered to be a more precise test of model intelligence than the Turing test [293].

For an LM with a trigram as the context window ($n = 3$), the probability of a token can be estimated using the following equation:

$$P(w_i|w_{i-2}, w_{i-1}) = \frac{count(w_{i-2}, w_{i-1}, w_i)}{count(w_{i-2}, w_{i-1})} \tag{A.4}$$

If we consider a prediction vocabulary then the same score calculation can be utilized to predict the next word using:

$$\arg\max_{w_i} P(w_i|w_{i-2}, w_{i-1}) \tag{A.5}$$

There are also various smoothing approaches for LMs so that the model does not face mathematical in-operation errors like zero division error if certain text phrases are not observed in the training data. Some of the popular smoothing techniques are the Good-Turing Smoothing [129] and Modified Kneser-Ney Smoothing [135, 38]. A statistical LM provides a significant speed and simplicity improvement. It is also generalizable across a training corpus. However it suffers from out-of-vocabulary context issues and doesn't perform well on unseen data.

Appendix B

Multiple-Run Evaluation and Statistical Significance

Although the results tables report the average performance of the correction approaches across six runs, statistical significance was evaluated using the individual results from each seed rather than the averaged values, using a one-tailed Wilcoxon signed-rank test ($\alpha = 0.05$). In Chapter 3, the ASR source transcript WER (ASR Normalized) is fixed, so the six per-seed WER values of each correction approach are tested against the ASR Normalized WER, with the $\downarrow$ denoting a significant WER reduction. In Chapter 4, the six per-seed $F_{0.5}$ scores of each ensembling approach are paired with the Curriculum Learning (All) baseline scores, and a significant $F_{0.5}$ improvement is represented with $\uparrow$.

The six seed values were set to $\{100, 200, 300, 400, 500, 600\}$ and kept identical across all experiments, so that the baseline and proposed-method scores could be paired by seed for the signed-rank test. For several experiments, it was observed that multiple runs produced identical results. This can be attributed to two factors. (1) All correction models were fine-tuned from the same pre-trained checkpoint (BART or BARThez) with identical hyperparameter settings and training steps, stopping criteria and dataset (but different validation datasets), so the inference predictions converge to the same corrections. (2) The LLM-based inference experiments had a very low decoding temperature (for the correction tasks) to make the prediction specific and reduce paraphrasing/rewrites, so correction runs over identical inputs return identical outputs. These identical runs were retained in the average and in the paired test.

Appendix C

Supplementary Material for Low-Resource ASR Transcript Correction

C.1 Edit Type Distribution of the ASR Datasets

This appendix section reports the WER breakdown of all four ASR datasets used in Chapter 3 into substitution, deletion, and insertion components across all data splits (train/test/dev). Edits are counted relative to the target transcript, and all values are reported as WER percentages so the three components add up to the dataset WER - Table C.1. Each value is rounded independently, so the sum of the three components can differ from the dataset WER by ±0.01.

Table C.2 reports the WER breakdown for the No Insertion (NI) training data, where all ASR deletion edits have been removed from the source–target pairs so that the correction model does not need to predict missing content. The NI training data uses the same base LR sampled bitext training data (Section 3.6.5).

C.2 ASR Transcript Correction LLM Results

This section reports the ASR transcript correction results of all four instruction-tuned LLMs evaluated in Chapter 3 - Qwen2.5-7B, LLaMA3.1-8B, Mistral-7B and DeepSeek-R1-8B, on the public RED-ACE (English) and MLS (French) datasets (Tables C.3 and C.4).

Dataset	Split	WER (%)	Sub (%)	Del (%)	Ins (%)
Industry (English)	Train	10.68	4.26	4.54	1.87
	Dev	16.80	5.35	8.34	3.11
	Test	10.63	4.05	4.46	2.12
Industry (French)	Train	31.14	13.61	15.45	2.09
	Dev	31.33	13.31	15.78	2.23
	Test	28.71	12.93	13.92	1.86
RED-ACE (English)	Train (LR)	13.89	10.45	2.08	1.36
	Dev	13.74	10.32	1.96	1.47
	Test (Clean)	9.04	6.85	1.20	0.99
	Test (Other)	19.30	14.26	3.19	1.86
MLS (French)	Train (LR)	6.14	4.05	0.81	1.27
	Dev	7.76	4.97	1.51	1.28
	Test	7.05	4.11	1.38	1.56

Table C.1: The edit type breakdown for all four ASR datasets. The Train (LR) split for the public ASR datasets refers to the sampled low-resource bitext (Table 3.4).

Dataset	Split	WER (%)	Sub (%)	Del (%)	Ins (%)
Industry (English)	Train	6.40	4.33	0.03	2.03
Industry (French)	Train	18.53	15.70	0.15	2.67
RED-ACE (English)	Train (LR)	12.05	10.61	0.02	1.42
MLS (French)	Train (LR)	5.36	4.07	0.00	1.29

Table C.2: Edit type breakdown for the No Insertion (NI) training data, where all ASR deletion edits have been removed from the source–target pairs. The residual deletion rate is nonzero due to random WER alignment edit labelling.

Each LLM is evaluated using few-shot prompting (Kk, Section 3.7.2) and the synthetic n-best correction method (NB-Kk, Section 3.7.3) with $k \in \{0, 1, 3\}$ in-context examples. Qwen2.5-7B obtains the lowest WER of the four LLMs on both datasets, under both the few-shot and the n-best settings. As a result, it is the only LLM reported in Chapter 3 and evaluated on the industry dataset. The table follows the same structure as the results tables in Section 3.8.3.

For the other three LLMs, the in-context examples mostly help reduce over-editing rather than add correction. On MLS, LLaMA corrects a near-constant amount as k increases ($1.00 \rightarrow 1.01 \rightarrow 1.01$ from $k = 0$ to $k = 3$) while its introduced error falls from 6.08 to 3.42. Mistral behaves similarly, with introduced error dropping from 11.03 to 3.30 for a

Model	Dev Set			Test Set		
	WER	**WER_{Corr}**	**$\text{WER}_{\text{Introd}}$**	**WER**	**WER_{Corr}**	**$\text{WER}_{\text{Introd}}$**
Public — RED-ACE (English) LLM Results						
ASR Normalized	13.74	—	—	14.15	—	—
Qwen-K0	15.09	1.04 (7.60%)	2.39	15.50	1.04 (7.37%)	2.39
Qwen-K1	14.62	1.11 (8.10%)	1.99	15.06	1.11 (7.87%)	2.02
Qwen-K3	14.98	1.35 (9.85%)	2.59	15.43	1.37 (9.70%)	2.65
LLaMA-K0	17.84	1.34 (9.77%)	5.44	18.59	1.33 (9.38%)	5.77
LLaMA-K1	15.59	1.46 (10.62%)	3.30	15.80	1.45 (10.26%)	3.10
LLaMA-K3	14.94	1.64 (11.93%)	2.83	15.40	1.64 (11.58%)	2.89
Mistral-K0	20.12	1.13 (8.21%)	7.50	20.61	1.19 (8.42%)	7.65
Mistral-K1	19.36	1.26 (9.15%)	6.88	19.65	1.29 (9.09%)	6.78
Mistral-K3	18.24	1.36 (9.90%)	5.86	18.80	1.42 (10.01%)	6.07
DeepSeek-K0	20.28	0.63 (4.58%)	7.16	20.13	0.61 (4.32%)	6.59
DeepSeek-K1	14.91	0.43 (3.12%)	1.59	15.38	0.42 (2.98%)	1.65
DeepSeek-K3	14.69	0.51 (3.74%)	1.46	15.26	0.49 (3.43%)	1.60
Qwen-NB-K0	13.93	0.92 (6.69%)	1.11	14.38	0.98 (6.90%)	1.20
Qwen-NB-K1	13.93	0.87 (6.30%)	1.05	14.41	0.90 (6.34%)	1.15
Qwen-NB-K3	13.93	0.92 (6.70%)	1.11	14.37	0.96 (6.79%)	1.18
LLaMA-NB-K0	15.91	1.44 (10.46%)	3.60	16.73	1.50 (10.63%)	4.08
LLaMA-NB-K1	15.00	1.44 (10.50%)	2.70	15.63	1.52 (10.74%)	3.00
LLaMA-NB-K3	15.05	1.57 (11.41%)	2.88	15.55	1.64 (11.58%)	3.03
Mistral-NB-K0	17.33	1.35 (9.82%)	4.94	18.12	1.45 (10.24%)	5.42
Mistral-NB-K1	15.53	1.28 (9.34%)	3.07	16.30	1.35 (9.51%)	3.49
Mistral-NB-K3	15.24	1.25 (9.08%)	2.75	16.02	1.33 (9.40%)	3.19
DeepSeek-NB-K0	15.19	0.89 (6.47%)	2.34	15.67	0.90 (6.34%)	2.41
DeepSeek-NB-K1	15.08	0.93 (6.79%)	2.27	15.69	0.92 (6.53%)	2.46
DeepSeek-NB-K3	15.09	0.97 (7.04%)	2.31	15.72	0.97 (6.84%)	2.53
Qwen1M-K3	15.15	1.26 (9.19%)	2.67	15.56	1.30 (9.16%)	2.70
Qwen1M-K4500	20.32	1.56 (11.38%)	8.14	20.92	1.58 (11.18%)	8.35

Table C.3: Correction results of all four evaluated LLMs on the public RED-ACE (English) dataset, pooled over the *clean* and *other* splits. The model naming convention is described in Table 3.10.

corrected gain of only 0.15. The synthetic n-best candidates help the lower-performing LLMs when no in-context examples are provided. The remaining LLMs fail in two distinct

Model	Dev Set			Test Set		
	WER	**WER_{Corr}**	**$\text{WER}_{\text{Introd}}$**	**WER**	**WER_{Corr}**	**$\text{WER}_{\text{Introd}}$**
Public — MLS (French) LLM Results						
ASR Normalized	7.76	—	—	7.05	—	—
Qwen-K0	10.04	0.43 (5.52%)	2.71	9.02	0.55 (7.75%)	2.52
Qwen-K1	9.03	0.79 (10.24%)	2.06	8.06	0.87 (12.30%)	1.87
Qwen-K3	8.71	0.84 (10.86%)	1.80	7.59	0.99 (14.02%)	1.53
LLaMA-K0	13.26	0.99 (12.72%)	6.49	12.14	1.00 (14.16%)	6.08
LLaMA-K1	12.09	1.00 (12.94%)	5.34	11.33	1.01 (14.28%)	5.29
LLaMA-K3	9.96	0.99 (12.82%)	3.20	9.46	1.01 (14.34%)	3.42
Mistral-K0	18.05	0.80 (10.38%)	11.10	17.29	0.80 (11.31%)	11.03
Mistral-K1	11.66	0.89 (11.42%)	4.79	10.55	0.89 (12.56%)	4.38
Mistral-K3	10.28	0.91 (11.78%)	3.43	9.40	0.95 (13.46%)	3.30
DeepSeek-K0	35.97	0.86 (11.06%)	29.07	36.03	1.10 (15.56%)	30.07
DeepSeek-K1	14.80	0.90 (11.54%)	7.94	14.41	0.86 (12.16%)	8.22
DeepSeek-K3	13.61	0.85 (11.00%)	6.70	13.62	0.96 (13.62%)	7.53
Qwen-NB-K0	8.88	1.02 (13.11%)	2.14	8.14	1.04 (14.75%)	2.13
Qwen-NB-K1	8.64	0.96 (12.37%)	1.84	7.77	1.01 (14.39%)	1.73
Qwen-NB-K3	8.64	0.95 (12.25%)	1.83	7.97	1.00 (14.22%)	1.92
LLaMA-NB-K0	13.09	1.11 (14.26%)	6.44	12.50	1.22 (17.37%)	6.67
LLaMA-NB-K1	10.11	0.96 (12.38%)	3.32	9.52	1.17 (16.59%)	3.64
LLaMA-NB-K3	9.20	0.98 (12.63%)	2.42	8.43	1.14 (16.15%)	2.51
Mistral-NB-K0	12.60	1.04 (13.47%)	5.89	11.33	1.15 (16.24%)	5.43
Mistral-NB-K1	9.74	1.03 (13.29%)	3.01	8.76	1.07 (15.18%)	2.78
Mistral-NB-K3	9.34	1.01 (12.99%)	2.59	8.52	1.03 (14.64%)	2.51
DeepSeek-NB-K0	16.23	0.93 (11.96%)	9.40	16.87	0.91 (12.89%)	10.73
DeepSeek-NB-K1	13.02	0.97 (12.54%)	6.24	12.78	1.11 (15.71%)	6.84
DeepSeek-NB-K3	13.04	0.97 (12.49%)	6.25	13.12	0.99 (13.99%)	7.02
Qwen1M-K3	8.95	0.96 (12.37%)	2.15	7.99	1.04 (14.72%)	1.98
Qwen1M-K4500	9.04	0.71 (9.22%)	2.00	8.38	0.81 (11.43%)	2.13

Table C.4: Correction results of all four evaluated LLMs on the public MLS (French) dataset. The model naming convention is described in Table 3.10.

ways. Without in-context examples, they over-correct the source transcript, increasing the introduced error (MLS test: LLaMA-K0 6.08, Mistral-K0 11.03, DeepSeek-K0 30.07). With in-context examples, the DeepSeek reasoning model makes very few edits on RED-ACE,

correcting only 2.98-3.43% of the errors; the lowest correction WER of the four LLMs.

C.2.1 Per-split results on RED-ACE

Tables 3.15 and C.3 report RED-ACE results pooled over the *clean* and *other* splits. Tables C.5 and C.6 break the same models down by split. First, I observe that the relative correction rate (WER_{Corr} as a percentage of baseline errors) is higher on *clean* than on *other* for every model that corrects a non-negligible number of errors (the Seq2Seq baselines are near zero on both), while the *absolute* number of corrected errors is similar or slightly higher on *other*: the correctors fix a comparable number of errors on noisier input, but that input may contain roughly double the amount of errors to begin with. Second, the ranking of models is stable across splits: SDA achieves the lowest WER on both. The trends from the LLM results + SDA indicate that the pooled results are a good representative of both splits. The complete bitext few-shot setting (Qwen1M-K4500) introduces a similar amount of error on both splits (test: 7.77 for *clean*, 8.93 for *other*), even though the *other* split is noisier. Thus, the amount of introduced error (overcorrections) does not change with the amount of ASR Transcription source WER.

Model	Dev Set (clean)			Test Set (clean)		
	WER	**WER_{Corr}**	**$\text{WER}_{\text{Introd}}$**	**WER**	**WER_{Corr}**	**$\text{WER}_{\text{Introd}}$**
Public — RED-ACE (English) Results, clean split						
ASR Normalized	8.94	—	—	9.04	—	—
Baseline - BB	9.22	0.03 (0.37%)	0.32	9.35	0.02 (0.23%)	0.33
Baseline - BL	9.38	0.08 (0.93%)	0.53	9.54	0.10 (1.10%)	0.60
SDA - BB	7.70	1.88 (21.00%)	0.64	7.95	1.80 (19.97%)	0.71
SDA - BL	7.44	2.44 (27.35%)	0.95	7.78	2.32 (25.66%)	1.06
Qwen-K0	10.32	0.91 (10.17%)	2.29	10.25	0.96 (10.60%)	2.18
Qwen-K1	9.86	0.97 (10.87%)	1.90	9.72	1.04 (11.49%)	1.72
Qwen-K3	10.19	1.20 (13.44%)	2.45	10.07	1.30 (14.33%)	2.33
LLaMA-K0	13.22	1.18 (13.17%)	5.46	13.21	1.22 (13.48%)	5.39
LLaMA-K1	11.05	1.31 (14.71%)	3.43	10.50	1.32 (14.63%)	2.78
LLaMA-K3	10.08	1.49 (16.65%)	2.64	10.04	1.56 (17.25%)	2.56
Mistral-K0	15.21	0.95 (10.64%)	7.22	15.28	1.08 (11.91%)	7.32
Mistral-K1	14.61	1.07 (11.99%)	6.75	14.26	1.13 (12.52%)	6.36
Mistral-K3	13.48	1.16 (12.96%)	5.71	13.44	1.28 (14.21%)	5.68
DeepSeek-K0	16.91	0.52 (5.82%)	8.50	15.45	0.53 (5.85%)	6.94
DeepSeek-K1	10.33	0.36 (3.99%)	1.75	10.33	0.40 (4.38%)	1.69
DeepSeek-K3	10.08	0.41 (4.53%)	1.55	10.08	0.47 (5.21%)	1.51
Qwen-NB-K0	9.10	0.86 (9.68%)	1.03	9.36	0.91 (10.03%)	1.23
Qwen-NB-K1	9.11	0.80 (8.97%)	0.98	9.38	0.86 (9.49%)	1.20
Qwen-NB-K3	9.12	0.85 (9.46%)	1.03	9.33	0.92 (10.20%)	1.22
LLaMA-NB-K0	10.78	1.26 (14.08%)	3.10	11.84	1.35 (14.94%)	4.15
LLaMA-NB-K1	10.16	1.27 (14.17%)	2.49	10.50	1.40 (15.46%)	2.86
LLaMA-NB-K3	10.22	1.38 (15.45%)	2.67	10.47	1.47 (16.28%)	2.90
Mistral-NB-K0	12.46	1.18 (13.22%)	4.71	13.11	1.32 (14.65%)	5.40
Mistral-NB-K1	10.67	1.16 (12.95%)	2.89	11.38	1.24 (13.75%)	3.58
Mistral-NB-K3	10.41	1.13 (12.62%)	2.60	11.10	1.24 (13.69%)	3.30
DeepSeek-NB-K0	10.48	0.80 (8.93%)	2.34	10.72	0.81 (8.97%)	2.49
DeepSeek-NB-K1	10.33	0.84 (9.44%)	2.23	10.77	0.83 (9.23%)	2.57
DeepSeek-NB-K3	10.36	0.87 (9.79%)	2.30	10.85	0.87 (9.58%)	2.68
Qwen1M-K3	10.35	1.16 (12.94%)	2.57	10.25	1.23 (13.65%)	2.44
Qwen1M-K4500	15.48	1.32 (14.77%)	7.86	15.35	1.46 (16.14%)	7.77

Table C.5: Correction results of all evaluated models on the *clean* split of the public RED-ACE (English) dataset. Columns are as in Tables 3.15 and C.3, which report the same models pooled over both splits.

Model	Dev Set (other)			Test Set (other)		
	WER	**WER_{Corr}**	**$\text{WER}_{\text{Introd}}$**	**WER**	**WER_{Corr}**	**$\text{WER}_{\text{Introd}}$**
Public — RED-ACE (English) Results, other split						
ASR Normalized	18.95	—	—	19.30	—	—
Baseline - BB	19.18	0.05 (0.24%)	0.27	19.63	0.04 (0.23%)	0.37
Baseline - BL	19.43	0.15 (0.79%)	0.63	19.86	0.14 (0.70%)	0.69
SDA - BB	17.72	2.11 (11.14%)	0.88	18.07	2.14 (11.10%)	0.91
SDA - BL	17.34	2.90 (15.31%)	1.29	17.76	2.79 (14.46%)	1.24
Qwen-K0	20.26	1.19 (6.28%)	2.50	20.77	1.13 (5.85%)	2.60
Qwen-K1	19.77	1.27 (6.69%)	2.09	20.44	1.19 (6.16%)	2.32
Qwen-K3	20.18	1.52 (8.02%)	2.75	20.82	1.45 (7.52%)	2.97
LLaMA-K0	22.85	1.52 (8.03%)	5.42	24.02	1.44 (7.44%)	6.15
LLaMA-K1	20.50	1.62 (8.53%)	3.16	21.14	1.58 (8.21%)	3.42
LLaMA-K3	20.19	1.80 (9.51%)	3.04	20.80	1.72 (8.90%)	3.22
Mistral-K0	25.44	1.32 (6.96%)	7.80	25.98	1.31 (6.78%)	7.99
Mistral-K1	24.51	1.46 (7.69%)	7.02	25.07	1.44 (7.48%)	7.21
Mistral-K3	23.40	1.58 (8.33%)	6.03	24.20	1.55 (8.03%)	6.45
DeepSeek-K0	23.92	0.75 (3.95%)	5.71	24.84	0.69 (3.60%)	6.23
DeepSeek-K1	19.86	0.51 (2.67%)	1.42	20.47	0.45 (2.31%)	1.61
DeepSeek-K3	19.68	0.63 (3.34%)	1.36	20.49	0.50 (2.60%)	1.68
Qwen-NB-K0	19.18	0.98 (5.16%)	1.20	19.43	1.05 (5.42%)	1.17
Qwen-NB-K1	19.14	0.93 (4.93%)	1.13	19.46	0.94 (4.86%)	1.09
Qwen-NB-K3	19.15	1.00 (5.30%)	1.20	19.44	1.00 (5.18%)	1.14
LLaMA-NB-K0	21.46	1.63 (8.62%)	4.14	21.65	1.66 (8.60%)	4.00
LLaMA-NB-K1	20.25	1.63 (8.62%)	2.93	20.79	1.64 (8.52%)	3.13
LLaMA-NB-K3	20.28	1.77 (9.35%)	3.10	20.66	1.81 (9.37%)	3.16
Mistral-NB-K0	22.61	1.53 (8.08%)	5.19	23.16	1.58 (8.16%)	5.43
Mistral-NB-K1	20.81	1.42 (7.49%)	3.27	21.26	1.45 (7.51%)	3.41
Mistral-NB-K3	20.48	1.38 (7.27%)	2.90	20.97	1.43 (7.38%)	3.09
DeepSeek-NB-K0	20.30	0.99 (5.21%)	2.33	20.65	0.98 (5.10%)	2.33
DeepSeek-NB-K1	20.24	1.03 (5.44%)	2.32	20.64	1.01 (5.25%)	2.35
DeepSeek-NB-K3	20.20	1.07 (5.63%)	2.32	20.62	1.07 (5.55%)	2.38
Qwen1M-K3	20.35	1.38 (7.28%)	2.78	20.91	1.36 (7.05%)	2.97
Qwen1M-K4500	25.58	1.83 (9.64%)	8.45	26.53	1.71 (8.84%)	8.93

Table C.6: Correction results of all evaluated models on the *other* split of the public RED-ACE (English) dataset. Columns are as in Tables 3.15 and C.3, which report the same models pooled over both splits.

Appendix D

Supplementary Material for Edit Type Ensemble Models

D.1 LLM Model Specifications and Implementation Details

This section provides the technical specifications and system configurations for the Large Language Models (LLMs) used for the edit-based ensembling approach. All model inferences were run on the Killarney research cluster[1] managed by Vector Institute, Toronto[2].

We utilized multiple instruction-tuned models open for research and industry use, ranging from 7B to 8B parameters. The different models were selected to test a variety of architectural approaches for the text quality improvement tasks, and test our LLM based GEC ensembling approach.

Label	**Model**	**Parameters**
Qwen	Qwen2.5-7B-Instruct	7.61 B
LLaMA	Meta-Llama-3.1-8B-Instruct	8.03 B
DeepSeek	DeepSeek-R1-Distill-Qwen-7B	7.61 B
Mistral	Mistral-7B-Instruct-v0.3	7.24 B

Table D.1: GEC Ensemble LLM Models and Specifications

[1] https://docs.alliancecan.ca/wiki/Killarney
[2] https://vectorinstitute.ai/

D.1.1 Implementation Details

The LLM-based ensembling was implemented using the `vLLM` library [148], since it is highly capable at efficient GPU memory utilization during LLM inference. All experiments were performed using the Killarney research cluster using:

- `CUDA 12.6`
- 1 NVIDIA L40S GPU with 48 GB of VRAM.

Software Environment

To ensure reproducibility, all experiments were conducted using fixed software and package versions. The following packages were used for the LLM ensembling:

- `vllm == 0.16.0`
- `torch == 2.10.0`
- `transformers == 4.56.2`
- `Python == 3.11.5`

Inference Configuration

For all tasks, we set `gpu_memory_utilization=0.95` and `enable_prefix_caching=True` to optimize processing of the same instruction prompt prefixes across all LLM inference queries. A greedy decoding strategy (`temperature=0.0`) was used to ensure deterministic outputs, since we wish to reduce LLM paraphrasing and rewrites for GEC.

- **RMUAll Model Prediction Ensembling:** For the target prediction generation task conditioned on the M_R, M_M, M_U, and M_{All} GEC Model predictions, the model context was constrained to `max_model_len=2048` with a generation limit of 512 tokens. For ease of output parsing, the model was instructed to utilize a JSON-based generation structure. The stopping criteria were set to terminate at the closing brace (`"}"`).

- **Edit Evaluation Ensembling:** For the four GEC Model Prediction edits extraction and quality estimation using LLM task, a significantly larger generation context window was set to accommodate the source text, atomic edit, and the generated quality reasoning. For this task, the context was expanded to `max_model_len=4096`, with `max_tokens` increased to 1024. Since the model was instructed to generate the answer in a JSON structure, `"}"` and `"]"` tokens were used as the stopping criteria.

A single GPU was utilized for inference.

D.2 LLM Ensembling Task Instruction Prompts

GEC Prompt-based Edit Type Ensemble - Chapter 4.3.3 . Given a source text and multiple corrected hypotheses from edit-specialized models, an LLM is prompted to generate the final text correction. Figure 4.5 describes the LLM generation-based GEC Ensembling approach.

Standard Prompt: The LLM is instructed to generate the final correction text by considering all four model predictions, allowing it to freely modify or introduce extra edits.

Standard Prompt

You are an expert Grammar Error Correction (GEC) Ensemble system. You will be provided with candidate corrections from different GEC systems, each specializing in a specific edit type: substitutions, insertions, deletions, and a combined model that handles all edit types simultaneously. Your task is to analyze all the candidate correction edits and generate the most accurate final corrected version of the source text. Ensure the output is natural, fluent, and preserves the source meaning and tone. Generate the 'final_corrected_text' in a strictly valid JSON format.
Candidate Corrections:

```
{
  "source_text": "<Source Text>",
  "insertion_focused_edit": "<Insertion Edit Correction>",
  "deletion_focused_edit": "<Deletion Edit Correction>",
  "substitution_focused_edit": "<Substitution Edit Correction
     >",
  "combined_edits": "<All Edit Correction>"
}
{"final_corrected_text": "
```

No-Extra (NE) Prompt: The LLM is instructed to generate the final correction by only considering the edits present in the provided model predictions. The model is explicitly instructed not to introduce any new corrections beyond those suggested by the GEC edit-specialized models.

No-Extra Prompt

You are an expert Grammar Error Correction (GEC) Ensemble system. You will be provided with candidate corrections from different GEC systems, each specializing in a specific edit type: substitutions, insertions, deletions, and a combined model that handles all edit types simultaneously. Your task is to analyze these candidates and synthesize the most accurate final version of the source text. IMPORTANT: You must only consider the specific edits and substitutions suggested by the provided candidates. Do not introduce any independent stylistic changes, external paraphrasing, or original corrections of your own. Your role is strictly to ensemble the existing candidate edits into a single, fluent, and coherent sentence while strictly preserving the source meaning. Generate the 'final_corrected_text' in a strictly valid JSON format.
Candidate Corrections:

```
{
  "source_text": "<Source Text>",
  "insertion_focused_edit": "<Insertion Edit Correction>",
  "deletion_focused_edit": "<Deletion Edit Correction>",
  "substitution_focused_edit": "<Substitution Edit Correction
     >",
  "combined_edits": "<All Edit Correction>"
}
{"final_corrected_text": "
```

Edit Quality Classification Ensemble - Chapter 4.3.3 I introduce an LLM-based edit evaluation framework that somewhat matches the ESC-ensembling pipeline (Section 4.3.2) while adding semantic evaluation at the word level. Figure 4.6 describes the LLM edit-quality classification GEC Ensembling approach.

GEC Edit Evaluation Prompt

You are an expert Grammar Error Correction (GEC) edits evaluator. Given a source sentence and a candidate edit, evaluate the edit.
Your task:
1. Explain briefly WHY the edit is good or bad.
2. Assign ONE label from the following categories:
- harmful: introduces or worsens an error
- incorrect: wrong correction
- unnecessary: not needed but not harmful
- minor_improvement: small stylistic improvement
- useful: improves clarity or grammar
- important: fixes a clear grammatical issue
- essential: critical correction
Return STRICT JSON format:

```
{
  "edit_id": int,
  "reason": str,
  "label": str
}
```

Keep the reason concise (1 sentence).

D.3 Baseline LLM GEC Prompting Results

The GEC performance of the four LLM models in a zero-shot and few-shot setting. LLM-based GEC correction task baseline.

Model	Strategy	B19D			B19T			C14T		
		P	R	$F_{0.5}$	P	R	$F_{0.5}$	P	R	$F_{0.5}$
LLAMA	Zero Shot	31.17	49.25	33.64	38.34	65.65	41.81	48.50	56.79	49.96
	Few Shot - 1	33.79	48.45	35.97	41.90	65.02	45.11	51.52	54.06	52.01
	Few Shot - 3	34.44	48.83	36.60	42.28	65.76	45.53	52.14	54.38	52.57
QWEN	Zero Shot	31.02	49.68	33.54	37.87	66.55	41.44	49.34	58.68	50.96
	Few Shot - 1	34.24	49.54	36.50	41.46	66.33	44.82	52.15	56.13	52.90
	Few Shot - 3	34.29	49.68	36.55	41.59	66.44	44.95	52.26	56.88	53.12
DEEPSEEK	Zero Shot	25.93	37.85	27.67	33.38	55.66	36.28	45.03	48.23	45.64
	Few Shot - 1	31.12	36.12	32.01	37.78	53.91	40.18	49.31	43.00	47.91
	Few Shot - 3	32.17	36.04	32.88	39.32	54.46	41.64	50.37	42.61	48.60
MISTRAL	Zero Shot	24.34	47.21	26.95	29.63	64.17	33.20	41.29	57.55	43.76
	Few Shot - 1	26.04	47.49	28.63	32.07	64.38	35.64	44.01	57.39	46.16
	Few Shot - 3	26.77	47.59	29.34	32.65	63.79	36.18	44.62	56.58	46.59

Table D.2: Performance of LLAMA, QWEN, DEEPSEEK and MISTRAL under different prompting strategies on BEA19 Dev (B19D), BEA19 Test (B19T) and CoNLL14 Test (C14T). B19D and B19T are evaluated with ERRANT; C14T is evaluated with the M2 scorer.

D.4 Selected Edit-Type Thresholds

Config	LLM	τ_R	τ_M	τ_U	Dev $F_{0.5}$
BB	Llama	0.5	0.0	0.0	46.02
	Mistral	0.5	0.0	0.0	45.80
	Qwen	0.5	0.3	0.5	46.64
	DeepSeek	0.0	0.0	0.0	44.34
BL	Llama	0.5	0.1	0.1	49.19
	Mistral	0.5	0.0	0.1	49.06
	Qwen	0.5	0.3	0.1	49.27
	DeepSeek	0.0	0.0	0.0	48.17

Table D.3: Edit-type thresholds selected on BEA-19 Dev for the edit quality classification ensemble (τ_R: replacement, τ_M: insertion, τ_U: deletion). An edit is kept if its label score is $\geq \tau$; $\tau = 0.0$ keeps all edits of that type. Dev $F_{0.5}$ is the exact edit-match score used for selection, not the ERRANT score. The obtained thresholds are used to generate predictions for B19T and C14T.

D.5 Adapter Models Configuration

This section describes the adapter module configuration search for the PEFT edit-specific models (Section 4.5). I evaluated three adapter methods which keep the pre-trained BART model frozen.

Houlsby [107] adds two bottleneck modules to each Transformer layer, one after the multi-head attention block and one after the feed-forward block.

Pfeiffer [217] adds a single bottleneck module to each layer, after the feed-forward block.

LoRA [110] adds no new modules to the forward pass. It learns a low-rank update $\Delta W = BA$ of rank r for the existing weight matrices. I applied LoRA to the query, key and value projections and the feed-forward layers.

Each bottleneck module consists of a down-projection, a ReLU non-linearity, an up-projection and a residual connection. The size is determined by the reduction factor k, a bottleneck dimension of d/k for a model hidden size d.

The search was performed for the BART-Base A_{All} model after Stage 1 training, with 5 training epochs. I compared reduction factors $k \in \{16, 4, 2, 1\}$ for Pfeiffer and $k \in \{16, 4, 2\}$ for Houlsby, LoRA ranks $r \in \{16, 64\}$ (with $\alpha = 2r$), and learning rates of 3×10^{-4} and 1×10^{-3}. The best checkpoint of each configuration was selected on the FCE development set. Table D.4 compares the best configuration of each method with full fine-tuning.

Method	Best Configuration	Trained Params	FCE Dev $F_{0.5}$	CoNLL14 Test (M2)		
				P	R	$F_{0.5}$
Full Fine-Tuning	mean of 3 runs	139.4M (100%)	**52.00**	71.21	36.53	**59.84**
Pfeiffer	$k = 1$, lr 3×10^{-4}	14.2M (10%)	51.92	71.44	35.88	59.62
Houlsby	$k = 2$, lr 1×10^{-3}	14.2M (10%)	50.84	70.52	35.73	59.03
LoRA	$r = 16$, lr 3×10^{-4}	2.8M (2%)	47.43	67.39	31.16	54.68

Table D.4: Best configuration of each adapter method for the BART-Base All model after Stage 1 training. The FCE development set was used for checkpoint selection. Full fine-tuning is averaged over three runs (FCE Dev $F_{0.5}$ range 51.65-52.43).

The configuration search shows three things. (1) Adapter capacity has the largest impact on performance: the Pfeiffer adapter improves from 46.96 to 48.09, 50.21 and 51.92 FCE Dev $F_{0.5}$ as the reduction factor decreases from 16 to 4, 2 and 1, and the Houlsby adapter from 47.32 to 49.65 and 50.84 as k decreases from 16 to 4 and 2. (2) Longer training does not help: training for 10 instead of 5 epochs lowered the FCE Dev $F_{0.5}$ for every configuration tested (for example, from 51.92 to 50.44 for Pfeiffer $k = 1$). (3) LoRA performs worse than both bottleneck adapter methods, even with a $4\times$ larger rank ($r = 64$: 46.84). With a reduction factor of 1, the Pfeiffer adapter performs as well as full fine-tuning (within the three-run range), and was selected for all experiments in Section 4.5.

Training Configuration All adapter and fully fine-tuned models were trained with the same parameters: label smoothing of 0.1, the Adam optimizer without weight decay, a linear learning rate schedule, no gradient clipping, a batch size of 256, and a maximum sequence length of 128 tokens.

- **Stage 1:** 5 epochs with 500 warm-up steps; learning rate 3×10^{-4} (adapters) and 3×10^{-5} (full fine-tuning).
- **Stage 2:** 5 epochs with 30 warm-up steps; learning rate 3×10^{-5} (adapters) and 1×10^{-5} (full fine-tuning).

- **Decoding:** beam search with a beam size of 5, a maximum output length of 256 tokens, and no n-gram repetition blocking.

Software Environment

- `adapters == 1.3.0`
- `transformers == 4.57.6`
- `torch == 2.7.0`
- `Python == 3.9.16`
- NVIDIA Tesla P100 GPU with 12 GB of memory, FP32 precision.

Appendix E

Supplementary Material for Temporal Post-Editing Effort Estimation

E.1 MAE and Pearson Correlation for PEET Evaluation

I use this section to illustrate the difference between the Mean Absolute Error (MAE) and the Pearson correlation (r) metrics used to evaluate the PEET Scorer models (Section 5.5.3). I consider a toy example of five sentences with target post-editing times $T = [20, 25, 30, 35, 40]$ seconds, and compare the MAE and r scores of different prediction patterns in Table E.1.

Model	**Predictions (s)**	**MAE**	r
Case 1 - Constant mean predictor			
Constant mean	$[30, 30, 30, 30, 30]$	**6.0**	undefined
Tracks variation	$[5, 15, 30, 45, 55]$	10.0	**0.997**
Case 2 - Systematic vs. random error			
Over-estimate (+5 s)	$[25, 30, 35, 40, 45]$	5.0	1.00
Under-estimate (−5 s)	$[15, 20, 25, 30, 35]$	5.0	1.00
Random (±5 s)	$[25, 20, 35, 30, 45]$	5.0	0.82

Table E.1: MAE and Pearson correlation (r) for different prediction patterns on a toy example with target times $T = [20, 25, 30, 35, 40]$ seconds. The correlation for the constant mean predictor is undefined, since its predictions have zero variance.

Case 1: A constant predictor that always estimates the mean target time (30 seconds) obtains a lower MAE (6.0) than a model that tracks the variation in effort across sentences

(10.0), even though it provides no information about which sentences require more effort. The variation-tracking model over-estimates the spread of the target times, increasing its MAE, but its predictions co-vary with the target ($r = 0.997$). MAE favours predictions close to the centre of the target distribution, while r rewards capturing the variation in effort.

Case 2: All three models have the same MAE (5.0), with each prediction 5 seconds away from the target. The systematic over-estimation and under-estimation models obtain $r = 1.00$, since a constant offset does not change how the predictions co-vary with the target, while the random error model obtains $r = 0.82$. MAE cannot differentiate between systematic and random error, and r cannot detect a constant bias in the predictions, which is why I report both metrics for the PEET Scorer models.

E.2 Neural Regression Models for PEET Estimation

Model Type	**Input Format**
Sentence Edit	[MO] ⟨mo⟩ [TRG] ⟨trg⟩
Syntactic Variation	⟨mo-parse⟩ [TO] ⟨trg-parse⟩
#EW + Syntactic Variation	#EW - ⟨mo-parse⟩ [TO] ⟨trg-parse⟩
#EW + Syntax Structure	#EW - ⟨trg-pos⟩

Table E.2: The training data format for the BERT and RoBERTa LM. The example considers a sentence pair - ⟨mo⟩ and ⟨trg⟩, where "mo" is the Model Output correction made by a GEC Tool and "trg" is the post-edited target correction for "mo". ⟨mo-parse⟩ and ⟨trg-parse⟩ denote the constituency parses, and ⟨trg-pos⟩ the part-of-speech tags of the target. The special tokens [MO], [TRG] and [TO] denote sentence breaks in the input. #EW denotes the number of edited words between mo and trg.

Since semantics and syntax structure have been shown to impact PE effort [286, 8], I also trained neural-LM PEET Scorer models using flattened constituency parse trees [132] and part-of-speech syntax structure features for the source and target corrections, generated using the spaCy library [105].

Model Features	BERT-L		RoBERTa-L	
	r	MAE	r	MAE
Sentence Edit	0.552	17.73	**0.56**	17.97
Syntactic Variation	0.528	19.35	**0.564**	18.05
#EW + Syntactic Variation	**0.564**	17.16	0.561	16.88
#EW + Syntax Structure	**0.565**	18.57	0.565	18.74

Table E.3: Performance of Neural PEET models using different sequence model features over 5 runs. The results are shown as Pearson Correlation (r) and Mean Absolute Error (MAE) loss.

Pretrained LMs can also capture syntax structure internally [53], so I also train neural-LM models using only source-target sentence embeddings as features to estimate PEET. Since the statistical models work as well as Neural models, while being faster and more interpretable, I consider them for the PEET Scorer in the main paper. I describe the features (Table E.2) and results of the Neural PEET (Table E.3) model here.

E.3 Predictive Model Parameters

We train different statistical and neural predictive models to estimate the post-editing temporal effort. I use this section to describe the predictive models as well as the training parameters for the regression task.

Linear Regression: We use the Linear Regression (LR) model provided by the Scikit-Learn library[1]. To keep the weights of the features from getting arbitrarily high, I used the RidgeLinear model that also adds an L2 Regularizer to the model. I trained the model with default training parameters and $alpha = 1.0$.

Support Vector Regression: We also train Support Vector Regression (SVR) models from scikit-learn with the default training parameters and the "linear" kernel.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

BERT, RoBERTa Neural Models: To train neural predictive models, I fine-tuned the BERT-Large [61] and RoBERTa-Large [172] with a regression head. The models were trained using the Pfeiffer bottleneck adapters [217] which allowed us to reduce the training time. I utilized the AdapterHub library[2] for training the models with the default Pfeiffer adapter configuration [218]. Training was done for 50 epochs with a 10-epoch and .05 loss threshold early stopping. A learning rate of $1e-04$ was used. To train the models for the regression task, I added a one-label regression head and used the mean-square-error loss (MSELoss), which is part of the Huggingface[3] training pipeline.

E.4 PEET Scorer using SRC, MO and TRG Sentence Features

Model Features	**BERT-L**		**RoBERTa-L**	
	r	MAE	r	MAE
Sentence Edit	0.513	19.10	0.54	17.82

Table E.4: Neural PEET model performance over 5 runs using the source (SRC), GEC Tool Model Output (MO) and Target Correction (TRG) sentence features. The results are shown as Pearson Correlation (r) and Mean Absolute Error (MAE) loss.

Statistical Model	**Edit Feature Level**	r	**MAE**
Linear Regression	10	0.558	18.92
	106	0.557	18.89
SVR Linear	10	0.556	16.39
	106	0.561	16.21

Table E.5: PEET Statistical Model performance over 50 runs (different train-test data seed) using Incorrect and Ignored separated Edit features (Section 5.5.1) extracted from SRC, MO and TRG sentence triples. The results are presented as the Pearson Correlation (r), Mean Absolute Error (MAE) loss.

[2] https://adapterhub.ml/
[3] https://huggingface.co/

Model Features	Regression Coefficient	Model Features	Regression Coefficient	Model Features	Regression Coefficient	Model Features	Regression Coefficient	Model Features	Regression Coefficient
R:OTHER	7.73	M:DET	2.03	M:VERB	1.49	U:VERB	1.07	M:ADJ	0.36
U:OTHER	4.53	M:OTHER	1.98	R:VERB:FORM	1.48	M:ADV	0.93	R:CONJ	0.30
Sentence Correct	-3.11	R:DET	1.94	U:PUNCT	1.36	U:ADJ	0.79	U:NOUN:POSS	0.29
R:PREP	2.85	M:PREP	1.93	U:ADV	1.32	R:VERB:INFL	0.58	U:VERB:TENSE	0.25
R:PUNCT	2.84	R:MORPH	1.77	M:VERB:TENSE	1.32	R:ADV	0.53	M:PART	0.18
M:PUNCT	2.80	U:PREP	1.69	M:VERB:FORM	1.29	M:NOUN:POSS	0.52	U:PART	0.10
R:VERB	2.71	R:SPELL	1.66	U:NOUN	1.26	R:ADJ	0.51	R:NOUN:POSS	-0.06
R:NOUN	2.64	U:CONJ	1.64	M:NOUN	1.22	M:PRON	0.49	U:PRON	0.06
R:NOUN:NUM	2.32	U:DET	1.62	R:PRON	1.14	R:PART	0.42	U:VERB:FORM	0.05
R:ORTH	2.22	R:WO	1.58	R:VERB:SVA	1.11	R:ADJ:FORM	0.41	M:CONTR	0.02
R:VERB:TENSE	2.08	M:CONJ	1.52	U:CONTR	1.10	R:NOUN:INFL	-0.37	R:CONTR	0.02

Table E.6: The standardized regression coefficients of the LR model trained on all the big (55) edit features to measure the impact of each feature on PEET estimation.

E.5 Feature Impact on Post-Editing Time using Regression Coefficients

We utilize the regression coefficients of a Ridge-Linear Regression model to quantitatively calculate the impact of different edit type features on the time-to-correct value (Section 5.6.2). I list the estimated impact of all edit types here.

Model Features	Regression Coefficient
Substitutions (R)	14.05
Deletions (U)	6.71
Insertions (M)	5.28
Sentence Correct (C)	-2.33

Table E.7: The standardized regression coefficients of the LR model trained on the small (4) edit features to measure the impact of each feature on PEET estimation.

Model Features	PEET Correlation	Regression Coefficient
# of words in TRG	0.43	14.07
Substitutions (R)	0.47	6.76
# of Edited Words	0.52	6.46
# of Words in MO	0.43	-5.86
Deletions (U)	0.32	3.85
Sentence Correct (C)	-0.3	-2.63
Insertions (M)	0.28	0.66

Table E.8: The correlation of the features used to train the small-edits(4) Linear Regression (LR) model in Table 5.6. I also list the standardized regression coefficients to measure the impact of each feature on PEET estimation.

E.6 PEET Scorer Ranking and Comparison of GEC Tools with Human Judgment Rankings

We evaluate and rank 33 different GEC Tools and correction sets, part of 3 GEC Human Judgment Rankings, to estimate the quality of the PEET Scorer (Section 5.6.3). I list all the GEC Tools along with the Human Judgment and PEET Scorer rankings here.

Model Name	HJR Score	PEET Score	PEET Ranking
marian	76.99	21.82	1
lstm-r	74.48	22.45	3
lstm	74.3	22.39	2
nus	73.94	22.47	4
transformer	73.9	22.79	5
amu	70.68	23.27	6
input	68.15	23.3	7

Table E.9: PEET Scorer estimated average time-to-correct per sentence and ranking for 7 GEC Tool corrections on the FCE dataset (1936 Sentences), along with their Human Judgment Ranking (HJR), presented in *Napoles-FCE* [197] (Section 5.6.3). The 7 GEC Tools consist of Seq2Seq Neural Models.

Model Name	HJR Score	PEET Score	PEET Ranking
lstm-r	78.27	27.61	2
lstm	77.73	27.61	1
amu	75.98	28.35	5
input	75.89	27.72	3
marian	75.8	30.52	7
nus	75.78	28.34	4
transformer	71.53	29.77	6

Table E.10: PEET Scorer estimated average time-to-correct per sentence and ranking for 7 GEC Tool corrections on the WikiEd dataset (1984 Sentences), along with their Human Judgment Ranking (HJR), presented in *Napoles-Wiki* [197] (Section 5.6.3). The 7 GEC Tools consist of Seq2Seq Neural Models.

Table E.9-E.10 list the estimation scores for the 6 Seq2Seq GEC Tools ranked by Napoles, Nădejde, and Tetreault [197]. The chosen models were AMU [121], LSTM/LSTM-R [134], Marian [260], NUS [42], and, Transformer [296].

Table E.11 lists the quality judgment for the 12 GEC Tools that participated in the CONLL14 GEC Task [200] performed by Grundkiewicz, Junczys-Dowmunt, and Gillian [90]. AMU [122], CAMB [70], CUUI [249], IITB [146], IPN [102], NARA [200], NTHU [312], PKU [340], POST [153], RAC [22], SJTU [301], UFC [95], and UMC [304].

Table E.12 lists the recent GEC Tools evaluated by Kobayashi, Mita, and Komachi [136]. GPT-3.5 [48], T5 [247], TransGEC [65], BERT-Fuse [125], Riken-Tohoku [133], PIE [4], LM-Critic [325], TemplateGEC [166], GECToR-BERT [203], UEDIN-MS [91], GECToR-Ens [280], BART [158].

Model Name	HJR Score	PEET Score	PEET Ranking
AMU	0.628	25.8	8
RAC	0.566	26.61	13
CAMB	0.561	26.34	11
CUUI	0.55	25.91	9
POST	0.539	26.28	10
UFC	0.513	24.56	2
PKU	0.506	25.63	6
UMC	0.495	25.72	7
IITB	0.485	24.67	3
SJTU	0.463	24.84	4
INPUT	0.456	24.53	1
NTHU	0.437	26.6	12
IPN	0.3	25.62	5

Table E.11: PEET Scorer estimated average time-to-correct per sentence and ranking for 12 GEC Tool corrections on the CONLL14 dataset (1312 Sentences), along with their Human Judgment Ranking (HJR), presented in *Grundkiewicz-C14(EW)* [90] (Section 5.6.3). The 12 GEC Tools consist primarily of rule-based and statistical machine translation architecture.

Model Name	HJR Score	PEET Score	PEET Ranking
REF-F	0.992	30.53	15
GPT-3.5	0.743	26.04	14
T5	0.179	24.37	10
TransGEC	0.175	23.54	3
REF-M	0.067	24.04	8
BERT-Fuse	0.023	23.61	4
Riken-Tohoku	-0.001	23.36	2
PIE	-0.034	23.66	6
LM-Critic	-0.163	24.37	9
Template GEC	-0.168	25.21	13
GECToR-BERT	-0.178	23.78	7
UEDIN-MS	-0.179	23.36	1
GECToR-Ens	-0.234	23.62	5
BART	-0.3	24.75	12
INPUT	-0.992	24.53	11

Table E.12: PEET Scorer estimated average time-to-correct per sentence and ranking for 15 GEC Tool corrections on the CONLL14 dataset (1312 Sentences), along with their Human Judgment Ranking (HJR), presented in *SEEDA-C14-All(TS)* [136] (Section 5.6.3). The 15 GEC Tools consist of strong SOA Neural Models.