Evaluating and Improving Generative Models with Feedback

by

Dongfu Jiang

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

© Dongfu Jiang 2026

Examining Committee Membership

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

External Examiner:	Colin Raffel Associate Professor, Dept. of Computer Science, University of Toronto
Supervisor(s):	Wenhu Chen Assistant Professor, Dept. of Computer Science, University of Waterloo
Internal Members:	Jimmy Lin Professor, Dept. of Computer Science, University of Waterloo Yuntian Deng Assistant Professor, Dept. of Computer Science, University of Waterloo
Internal-External Member:	Weiyi Shang Professor, Dept. of Electrical & Computer 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 am the sole author of Chapters 1, 2, 7, and 8, of the overviews that open Parts I and II, and of the connective material added throughout the thesis. Chapters 3–6 reproduce the text of four co-authored papers, on each of which I was a first author or an equal-contribution first author, with only the edits needed to fit them into this thesis. The sources and my contributions to each are listed below.

To connect the studies in this thesis, I also carried out additional analyses and experiments beyond those reported in the original papers. In particular, I examined whether an evaluator that agrees with human judgments remains reliable when used as a reward for training, and I ran the feedback-design comparisons reported in Chapter 6.

Chapter 3 (Learned Feedback for Generated Text). *TIGERScore: Towards Building Explainable Metric for All Text Generation Tasks* was co-authored with Yishan Li, Ge Zhang, Wenhao Huang, Bill Yuchen Lin, and Wenhu Chen, and published in *Transactions on Machine Learning Research* in 2024. Yishan Li and I contributed equally. I co-conceived and led the project, developed the approach for reference-free and explanatory evaluation, contributed to the construction of MetricInstruct, designed and analyzed the experiments, and led the writing and revision of the paper.

Chapter 4 (Human Feedback for Generated Images and Videos). *GenAI Arena: An Open Evaluation Platform for Generative Models* was co-authored with Max Ku, Tianle Li, Yuansheng Ni, Shizhuo Sun, Rongqi Fan, and Wenhu Chen, and published in the Datasets and Benchmarks Track of NeurIPS 2024. Max Ku, Tianle Li, and I contributed equally. I led the project and carried out its main work, in that I designed and built the platform and its voting and ranking procedure, analyzed the votes, built and released GenAI-Bench, and wrote the paper.

Chapter 5 (Execution Feedback for Generated Code). *ACECODER: Acing Coder RL via Automated Test-Case Synthesis* was co-authored with Huaye Zeng, Haozhe Wang, Ping Nie, Xiaotong Chen, and Wenhu Chen, and published as a long paper at ACL 2025. Huaye Zeng and I contributed equally, and I am listed second. I co-conceived and co-led the project, developed the framing that connects synthesized test cases to preference construction and reinforcement-learning rewards, guided the experimental design and analysis, and contributed substantially to the writing and revision of the paper.

Chapter 6 (Environment Feedback for Agent Trajectories). *VerlTool: Towards Holistic Agentic Reinforcement Learning with Tool Use* was co-authored with Yi Lu,

Zhuofeng Li, Zhiheng Lyu, Ping Nie, Haozhe Wang, Alex Su, Hui Chen, Kai Zou, Chao Du, Tianyu Pang, and Wenhu Chen, and published in *Transactions on Machine Learning Research* in 2026, and it also received the Best Paper Award at the ICLR 2026 SPOT workshop. Yi Lu, Zhuofeng Li, Zhiheng Lyu, and I contributed equally, and I was the project leader. I conceived and led the project, coordinated the system architecture and integration, designed the empirical study across tool-use domains, analyzed the training dynamics, and led the writing and revision of the paper.

Chapter 7 discusses subsequent work by my collaborators and me that builds on these four papers. That work is described there to situate the contributions of this thesis, and none of it is claimed as one of them. Three other papers on which I was a first or co-first author are summarized in Appendix A.

Abstract

Generative models can produce many candidate outputs for a request, but deciding which output is good, and what a model should learn next, requires feedback, a judgment of how well an output serves its task. This thesis studies how to obtain such feedback and how to use it, for evaluating generated outputs and for improving the models that produce them. The two uses place different demands on feedback. For evaluation, feedback has to agree with human judgment and explain its judgment. For training, it has to stay reliable while a model is optimized against it.

The first part studies feedback for evaluation. For text, I show that feedback can be learned without a reference output, by training an evaluator on error analyses so that it locates and explains each error in a response, and its scores agree with human ratings nearly as well as GPT-4 used as a judge. For images and videos, I collect pairwise human votes on an open platform, use them to rank generative models, and release them as a benchmark on which the best multimodal judge tested agrees with people about half the time, so human preferences remain the reference for this content.

The second part studies feedback for improving models, where agreement with people on fixed outputs is not enough. A policy trained against the learned text evaluator learned to refuse rather than answer, raising its reward while its accuracy fell. For code, I synthesize tests for tens of thousands of programming problems and turn execution results into preferences and rewards, which improve both Best-of-N selection and reinforcement learning. For agents that use tools over several turns, I build a reinforcement-learning framework that runs tools inside the training loop, train agents in six domains, and examine two choices in the feedback they receive. Rewarding tool use raises tool calls without necessarily raising task performance, and how much of a tool's output the policy reads can change the result.

Across these studies, feedback that evaluates fixed outputs well can behave differently once a model is trained against it, so a training reward needs a measure it does not see, tracked while the model is optimized.

Acknowledgements

I am deeply grateful to my supervisor, Wenhu Chen, for his guidance and unwavering support throughout my doctoral studies. Our weekly one-on-one meetings and his patient, insightful discussions were invaluable to my growth as a researcher. Wenhu gave me the freedom to explore my own ideas while encouraging me to ground my research in genuine needs and real-world value. His judgment in identifying problems that are both meaningful and useful has profoundly shaped my research values and practice. He also supported my internships and encouraged me to pursue opportunities beyond academia. Many of his lessons revealed their full significance only years later, as I encountered similar challenges myself. Beyond research, Wenhu has been a caring mentor and a wonderful friend. He brought our group together through many gatherings and welcomed his students into his home for New Year's Eve dinners and conversations. In both research and life, he has been a true role model for me.

I am especially grateful to Bill Yuchen Lin, who first guided me into research. When my first project stalled for months in 2022–2023, he remained remarkably patient and helped me find a direction better suited to my strengths. His kindness and patience profoundly shaped the way I now mentor junior researchers. I have tried to extend the same generosity to them, and their growth has been among the most rewarding parts of my doctorate. I am also grateful for Bill's mentorship during my internship at Ai2. Although those three months did not, for various reasons, culminate in a completed project, his lessons stayed with me. Bill taught me how to structure a presentation, design clear slides, and speak at a measured pace.

I am equally grateful to Wei Ping, who mentored me during my nearly year-long internship at NVIDIA, where I contributed to the Nemotron models and first participated in large-scale language-model post-training. His approach to team leadership and his insights into industrial research helped me transition from an academic to an industrial research mindset. Through his example, I learned to collaborate within a large team and to balance scientific exploration with practical constraints.

I am also deeply grateful to Xiang Yue, from whom I learned how to design and evaluate benchmarks rigorously, lead a research team, and frame and present a paper effectively. I thank Qian Liu, Tianyu Pang, and Chao Du, who mentored me during my time at Sea AI Lab, for creating an open and intellectually stimulating research environment. They gave me the freedom to explore multimodal research and to develop open and extensible reinforcement-learning frameworks. I am also grateful to Jimmy Lin, Yejin Choi, Jiawei Han, and James Zou for their invaluable advice, thoughtful perspectives, and many insightful discussions throughout our collaborations.

I thank the members of my examining committee—Colin Raffel, Jimmy Lin, Yuntian Deng, and Weiyi Shang—for the time and care they devoted to reading this thesis and for their thoughtful questions and feedback.

This thesis grew from work with an extraordinary group of collaborators. I thank Ge Zhang, Xueguang Ma, Yuansheng Ni, Cong Wei, Max Ku, Weiming Ren, Kai Zhang, Yujie Lu, Wenhao Huang, Lipeng He, Sherman Siu, Disen Liao, Yiming Jia, Wentao Ma, Yifei Wang, Quy Duc Do, Ziyan Jiang, Xiaotong Chen, Zejun Ma, Zhiheng Lyu, Benjamin Schneider, Tianhao Liang, Ping Nie, Yubo Wang, Haozhe Wang, Haoxiang Zhang, Yuyu Zhang, Kai Zou, Jianwen Xie, Alex Su, Hui Chen, Yu Zhang, Pan Lu, Yuyang Bai, Shangbin Feng, Hangxiao Zhu, Ming Zhong, Zhangchen Xu, and Penghui Du. I am especially grateful to the students with whom I had the privilege to work: Xuan He, Yishan Li, Wyett (Huaye) Zeng, Zhuofeng Li, Yi Lu, Jialin Yang, Chi Ruan, and Yuxuan Zhang. Their ideas, energy, and care made the projects in this dissertation possible.

I am thankful to the Cheriton School of Computer Science and the broader research community at the University of Waterloo for providing an intellectually generous home for this work. I also thank my colleagues on the NVIDIA Nemotron and ADLR teams for the opportunity to test and extend these ideas in a different research setting.

This research was supported by the Graduate Research Studentship (GRS) and the International Doctoral Student Award at the University of Waterloo, as well as by a Vector Research Grant from the Vector Institute. The experiments in this thesis were made possible by computational resources provided by the Digital Research Alliance of Canada, the Vector Institute, Sea AI Lab, NVIDIA, and NetMind.AI.

Beyond research, I was lucky to have friends who made these years warm. I thank Songcheng Cai, Xiangchao Chen, Xingyu Fu, Chengsong Huang, Jiarong Liang, Ruoxi Ning, Renjie Pi, Xinyu Shi, Jinyan Su, Yiping Wang, and Wanru Zhao for the meals, trips, and long conversations that helped me through it, and Wanru in particular for her constant encouragement. Many more friends deserve thanks than I can name here, and I apologize to those I have left out.

Finally, my deepest thanks go to my family for their unwavering support. I grew up in the small city of Dengfeng in Henan Province, where I spent the first eighteen years of my life. I was then fortunate to spend four years at Zhejiang University, which opened the door to the world of academic research, followed by three years of doctoral study at Waterloo. As this journey has taken me farther from home, I have always known that my family stands behind me. Their love and support have formed the most steadfast foundation of my life and career. None of this would have been possible without them. Words cannot fully express my gratitude, and I will always remember where this journey began.

Table of Contents

Examining Committee Membership **ii**

Author's Declaration **iii**

Statement of Contributions **iv**

Abstract **vi**

Acknowledgements **vii**

List of Figures **xii**

List of Tables **xv**

List of Abbreviations **xix**

1 Introduction **1**

1.1 Feedback Sources and Uses 3

1.2 Research Questions and Contributions 6

1.3 Thesis Organization 8

2 Background **9**

2.1 Feedback for Evaluation 10

2.2 Learning from Feedback 14

2.3 Tool Interaction and Agent Training 16

I Evaluating Generative Models with Feedback **18**

3 Learned Feedback for Generated Text **20**

3.1 Introduction 20
3.2 Related Work 22
3.3 Method 23
3.4 Experiments 29
3.5 Analysis 35
3.6 Summary 38

4 Human Feedback for Generated Images and Videos **39**

4.1 Introduction 39
4.2 Related Work 41
4.3 GenAI-Arena: Design and Implementation 42
4.4 Benchmarks and Results Discussion 47
4.5 GenAI-Bench 52
4.6 Summary 53

II Improving Generative Models with Feedback **54**

5 Execution Feedback for Generated Code **56**

5.1 Introduction 56
5.2 Related Works 58
5.3 Methodology 60
5.4 Experiments 64
5.5 Summary 71

6 Environment Feedback for Agent Trajectories **73**
6.1 Introduction . . . 73
6.2 Related Work . . . 75
6.3 VerlTool Framework . . . 77
6.4 Experiments . . . 82
6.5 Feedback Design . . . 86
6.6 Summary . . . 88

7 Discussion **90**
7.1 Reward Hacking . . . 90
7.2 Subsequent Work . . . 91
7.3 Future Directions . . . 93

8 Conclusion **95**
8.1 Evaluating Generated Outputs . . . 95
8.2 Improving Generative Models . . . 96
8.3 How the Two Parts Connect . . . 96
8.4 Closing Remarks . . . 97

References **98**

A Other Work Completed During the Doctorate **118**

B Supplementary Material from the Published Papers **120**
B.1 TIGERScore (Chapter 3) . . . 120
B.2 GenAI-Arena (Chapter 4) . . . 134
B.3 AceCoder (Chapter 5) . . . 137
B.4 VerlTool (Chapter 6) . . . 139

C Supplementary Analyses of Multi-Turn Training **145**
C.1 Tokenization Stability . . . 145
C.2 Trajectory Masking . . . 146

List of Figures

1.1 One request, four sampled programs, and four sources of feedback. 4

3.1 The upper part shows the input and output format of TIGERScore, which takes an instruction and an input context, along with the model-generated output that is to be evaluated as the inputs, and output detailed breakdown error analysis with explanations. It is finetuned on our curated dataset `MetricInstruct`. The lower part shows the Kendall's correlation of different metrics w.r.t human ratings, from which we can see TIGERScore achieves the SoTA correlation on most generation tasks. 21

3.2 Overall pipeline of constructing `MetricInstruct` through the two-channel collection. The main difference is that the real-world channel collects outputs y' from real-world models, while the synthetic channel asks GPT-4 to synthesize an error output based on the provided reference. 26

3.3 Investigation of the influence of Real-World & Synthesic mix training on the 7B model. 35

3.4 Optimizing a policy against TIGERScore. 37

4.1 GenAI Arena contains three components: (1) text-to-image, text-to-video and image editing arena, which accept community voting to obtain the preference pairs. (2) The leaderboard utilizes the preference pairs to calculate elo ranking for all the evaluated models. (3) We further release GenAI-Bench to judge different multimodal LLM judges. 39

4.2 GenAI Arena User Voting Interface. 43

4.3 Winning fraction heatmap of different models for the three tasks in GenAI-Arena 49

4.4 Battle count heatmap of different models for the three tasks in GenAI-Arena (without Ties) . . . 49
4.5 Average Win Rate Against All Other Models (Assuming Uniform Sampling and No Ties) . . . 49
4.6 Example of votes from users on the GenAI-Arena for the three generative tasks . . . 51

5.1 Overall Workflow of our model: we start from the seed code dataset to create well-formatted questions and corresponding test cases. Then we adopt strong models to filter the noisy test cases. Finally, we adopt these test cases to harvest positive and negative program pairs for reward model training and RL. . . . 57

6.1 Overview of the VerlTool, a modularized and efficient framework for the **A**gentic **R**einforcement **L**earning with **T**ool Use (ARLT) training paradigm, where the RL workflow and tool execution are fully disaggregated for both efficiency and extensibility. . . . 79
6.2 Visualization of the Async Rollout pipeline design and its effect in saving time. . . . 80
6.3 Example of code design for adding a new tool in VerlTool via the plugin interface. . . . 81
6.4 Tokenization of LLM generated content "...`</python>`" and tool observation "`\n<result>`..." can produce different token lists using Qwen2.5 tokenizer under different strategies. . . . 82
6.5 Training dynamics using VerlTool on all 6 tasks. For each task, the corresponding test benchmarks are AIME24, NQ, Spider-Test, VStar, GAIA, and SWE-Verified. All models are trained and evaluated based on VerlTool framework. The actual evaluation performance (purple dash) can be higher due to the train-eval settings difference. The number of actions is averaged over all sampled responses in each batch. . . . 83
6.6 Auxiliary reward comparison in VT-Math. . . . 87

B.1 Distribution analysis of MetricInstruct training data for the context length, number of errors, and error severities. The left figure shows the context length distribution. The right-top plot illustrates the distribution of the per-instance number of errors. The right-bottom figure illustrates the counts of "Major" and "Minor" errors for all six tasks 123

B.2 Bootstrap of Elo Estimates (1000 Rounds of Random Sampling) 134

C.1 Tokenization comparison in SQL training. 146

C.2 Trajectory masking in Math-TIR. 147

List of Tables

1.1 Chapters, sources of feedback, and publications. 8

3.1 Definitions of evaluation aspects of TIGERScore for Instruction-following (Instruct) as an example. See full table in Table B.2 for aspect definitions of all 6 text generation tasks 25

3.2 The composition of our dataset. For synthetic data, the output is generated by asking GPT-4 to synthesize incorrect outputs that contain a few specific types of errors. For the datasets with ‡, we take their pre-released system outputs. For the others, we collect the system outputs on our own. 27

3.3 Overview of our main evaluation datasets. "# Samples" is computed by multipling "# Inputs" with the number of outputs systems, representing the total instances that a metric need to compute. 30

3.4 The Kendall correlation results of all the baseline metrics and TIGERScore on the evaluation datasets shown in Table 3.3. For each task, the metric with the highest correlation to average performance is highlighted in bold. The results of Pearson and Spearman are reported in Table B.15 and Table B.16 respectively, showcasing the same great performance of TIGERScore 31

3.5 Human evaluation results, the first question is asked per error in error analyses, and the others are per sample. Superior performance is indicated by higher numerical values. The most-voted rate of each task for each human evaluation aspect is bolded. 33

3.6 TIGERScore's score on the gold reference of the test set. For each task, the 0 column refers to the percentage that TIGERScore reduces 0 scores for the gold references. The > -2 column refers to the percentage that TIGERScore deducts less than 2 points from the gold references. 34

3.7 The accuracy of the error analysis from TIGERScore-13B assessed by GPT-4 that do not contain hallucinations or factual errors. 34
3.8 Ablation of the influence of multiple tasks mix or single task on the 13B model. 35
3.9 Best-of-8 selection with TIGERSCORE on GSM8K. 36

4.1 Comparison with different evaluation platforms on different properties. . . 41
4.2 The overview of all text-to-image generation models. 45
4.3 Overview of all the image editing models. 45
4.4 Overview of all text-to-video generation models. 45
4.5 GenAI-Arena Leaderboards. (Last updated on Oct 24th, 2024) 48
4.6 Expert Review for 350 sampled human votes 51
4.7 GenAI-Bench leaderboard designed to benchmark MLLMs's ability in judging the quality of AI generative contents by comparing with human preferences. Numbers are accuracy (%). 53

5.1 Dataset statistics of ACECODE-87K before and after test-case filtering. . . 63
5.2 Pass rates on ACECODE-87K after filtering (Qwen2.5-Coder-7B-Instruct; 16 responses per question; temperature 1.0). 63
5.3 ACECODE-RM's best-of-n results on several benchmarks. Specifically, -C means completion split and -I means instruct split of BigCodeBench. Each Δ is that reward model's best score across N in the column minus the greedy score, and might be off by 0.1 due to rounding. 64
5.4 ACECODER's Performance after RL tuning using Reinforcement++ algorithm. We start with 3 different initial policy models and 2 kinds of reward types, where *RM* means using our trained ACECODE-RM and *Rule* means using the binary pass rate. Results show consistent improvement across various benchmarks. 66
5.5 ACECODE-RM's performance against other open-sourced reward models in terms of Best-of-16 sampling for Llama-3.1-8B-Inst. We can see the top-ranked RM on Reward Bench get little improvements compared to ours. . . 69
5.6 ACECODE-RM's Performance on RM Bench against various other reward models. We can see that ACECODE-RM-32B model performed best in Coding, Chat, Normal, Hard, and Average scores against all other reward models. 69

5.7 Ablation study on test-case filtering. Results are Best-of-16 sampling performance. . . . 70
5.8 Comparison of AceCode-RM's performance trained on different base model, where AceCode-RM (Llama) is based on Llama-3.1-Inst-8B and AceCode-RM (Qwen) is based on Qwen-Coder-2.5-7B-Inst. Results are Best-of-16 sampling performance. . . . 70

6.1 Tool support comparison across different frameworks (up to update until August 23, 2025). RAGEN and ROLL focus on the puzzle environments like bandit, which we did not list here. . . . 76
6.2 Performance comparison of Synchronous vs Asynchronous approaches. Experiments conducted on 8 H100 GPUs. . . . 80
6.3 Results on Math-related benchmarks with Python interpreter tool. The best results are indicated in **bold** and the second-best results are underlined. . . . 83
6.4 Results of on knowledge-QA benchmarks. $^{\dagger}/^{\star}$ represents in-domain/out-domain datasets. . . . 84
6.5 Results on NL2SQL (left) and SWE-Verified (right) benchmarks in terms of pass rates. . . . 84
6.6 Results on visual reasoning (left) and agentic search benchmarks (right). . . . 85
6.7 Observation length comparison in VT-Search. . . . 88

7.1 Subsequent work by my collaborators and me. . . . 92

B.1 Prompt template for TIGERScore fine-tuning and inference . . . 122
B.2 Definitions of evaluation aspects of TIGERScore for the 6 text generation task. . . . 125
B.3 Prompting templates for summarization task . . . 126
B.4 Prompting templates for translation task . . . 126
B.5 Prompting templates for data2text task . . . 126
B.6 Prompting templates for mathQA task . . . 126
B.7 Prompting templates for long-form QA task . . . 127
B.8 Prompting templates for instruction-following task . . . 127

B.9 Prompt template to format the response into json format 127
B.10 Prompt template to generate data in the synthetic channel for all 6 tasks. . 128
B.11 Prompt template to generate data with free-form aspects in the synthetic channel for instruction-following task. Data generated from this template aims to improve TIGERScore's generalization ability on the unseen error cases and enhance the robustness. 129
B.12 Prompt template used to check whether the error analysis synthesized is reasonable or not. 130
B.13 An example prompt when creating the template of mathQA 130
B.14 The prompt template we used to identify whether there are hallucinated contents or factor errors in TIGERScore's response 131
B.15 The **Pearson** correlation results of all the baseline metrics and TIGERScore on the evaluation datasets shown in Table 3.3. For each task, the metric with the highest correlation to average performance is highlighted in bold. 132
B.16 The **Spearman** correlation results of all the baseline metrics and TIGERScore on the evaluation datasets shown in Table 3.3. For each task, the metric with the highest correlation to average performance is highlighted in bold. 133
B.17 Models used during the training and evaluation process of AceCode-RM. 138
B.18 Prompt Used for Converting Seed Code Dataset into LeetCode-style Questions and Test Cases 138
B.19 Prompt Used for Converting Seed Code Dataset using only the reference program without instruction into LeetCode-style Questions and Test Cases 139
B.20 Currently supported tools of VerlTool framework. 144

List of Abbreviations

BLEU	Bilingual Evaluation Understudy
CLIP	Contrastive Language–Image Pretraining
DAPO	Decoupled Clip and Dynamic Sampling Policy Optimization
DCI	Direct Corpus Interaction
DPO	Direct Preference Optimization
FID	Fréchet Inception Distance
GRPO	Group Relative Policy Optimization
KL	Kullback–Leibler (divergence)
LLM	Large Language Model
MLLM	Multimodal Large Language Model
MQM	Multidimensional Quality Metrics
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
RLHF	Reinforcement Learning from Human Feedback
RLVR	Reinforcement Learning with Verifiable Rewards
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
SQL	Structured Query Language
SWE	Software Engineering (task domain)
TIR	Tool-Integrated Reasoning

Chapter 1

Introduction

Large language models (LLMs) power conversational systems such as ChatGPT (OpenAI, 2023), allowing users to ask questions, request explanations, and draft or revise text through conversation. These models can also solve mathematical word problems and other tasks that require several reasoning steps (Wei et al., 2022b). With access to tools, they can act as coding agents that inspect repositories, edit files, and run tests, or as search agents that issue queries and read retrieved material to gather evidence (Yang et al., 2024b; Yao et al., 2023). Other generative models produce high-fidelity images and videos (Rombach et al., 2022; Ho et al., 2022). These tasks place different requirements on the outputs. A mathematical solution should be correct, while a generated image should follow the requested description. To assess these outputs, we need information about whether those requirements have been met, such as a person's judgment or the result of testing a program. This feedback can be used to evaluate outputs and guide further training.

Designing feedback for generated outputs has been a long-standing topic in natural language processing. Earlier work often focused on specific tasks, such as machine translation and summarization, for which researchers developed automatic evaluation metrics. For example, BLEU (Papineni et al., 2002) measured overlap between generated translations and human-written references, while ROUGE (Lin, 2004) provided a set of measures for comparing generated summaries with reference summaries. These tools allowed researchers to score outputs and compare systems repeatedly during development. Their scores could also serve as feedback during training. For example, Ranzato et al. (2016) used reinforcement learning to train sequence models to optimize evaluation metrics such as BLEU and ROUGE, giving the model a reward based on the sequence it generated. Feedback could also be learned from people rather than specified by an overlap metric. In summarization, Stiennon et al. (2020) collected human comparisons of candidate summaries, trained a re-

ward model to predict those preferences, and used its scores to fine-tune a summarization model through reinforcement learning. The criteria used to evaluate generated outputs could therefore also shape what a model learned to generate.

As LLMs and other generative models take on a wider range of tasks, we need richer forms of feedback for evaluation and model improvement than simple reference overlap measurements. For evaluation, feedback should closely reflect human judgments of quality and explain the reasons behind its assessments. For example, InstructScore produces both an evaluation score and a diagnostic report that explains errors in generated text (Xu et al., 2023d). For model improvement, feedback must remain reliable as models are optimized against it, ensuring that higher rewards correspond to better task performance. For example, DeepSeek-R1-Zero uses mathematical answer checks and coding tests as verifiable rewards for reinforcement learning, demonstrating how feedback from task outcomes can improve reasoning performance (Guo et al., 2025). For agents that act over several turns, the environment also returns tool outputs during the task, and a trainer must decide what to reward, what only to show the policy, and what to do with calls that fail.

However, obtaining feedback that meets these requirements remains challenging. Learned evaluators require training data that covers the types of errors they are expected to detect, along with human judgments against which their scores and explanations can be validated. Collecting such judgments is time-consuming and requires evaluation procedures tailored to different tasks. For model improvement, many coding datasets lack executable tests that provide reliable feedback, while training agents requires infrastructure that supports interactions with diverse tools and environments.

This thesis takes up these challenges in two parts, because judging generated outputs and using feedback to improve a model pose different research questions.

Part I studies feedback for evaluation, asking how to obtain feedback that better aligns with human judgments and provides understandable explanations. For generated text, I first show that it is possible to train a reference-free evaluator, TIGERScore, to identify errors in a response and explain each of them, while achieving good alignment with human ratings (Chapter 3). For image and video generation, I collect human preferences through a self-hosted voting platform, GenAI-Arena, to benchmark generative models. I further show that current multimodal LLMs agree with human votes on this content only about half the time, so for images and videos human preferences remain important, both for ranking generators and for judging the judges (Chapter 4).

Part II studies feedback for model improvement, asking how feedback can support the selection of better outputs and guide model learning. Unlike Part I, where feedback is used to judge fixed outputs, the outputs here come from a policy that is optimized against

the feedback itself. As a result, even feedback that agrees with human judgments can still be exploited through reward hacking. Reliable feedback for learning therefore needs to be grounded in verifiable signals and designed for each task specifically. For coding, I show that execution can provide feedback at scale, with AceCoder automatically constructing test cases and turning execution results into rewards, both for training reward models to select among candidates and for reinforcement learning directly (Chapter 5). For agentic tasks, I build VerlTool, a framework that runs tools inside the training loop, train agents in six domains, and examine two choices in the feedback they receive. I show that rewarding tool use raises tool calls without necessarily raising task performance, and that how much of a tool's output the policy reads can change the result (Chapter 6).

1.1 Feedback Sources and Uses

Across the studies in this thesis, feedback differs in where it comes from, and its source determines what it can see in an output, what it costs to obtain, and how it fails. The classical source is a human-written reference. Metrics such as BLEU (Papineni et al., 2002) and ROUGE (Lin, 2004) score a translation or summary by its word overlap with references, which is inexpensive and repeatable once references exist. Overlap, however, measures resemblance to one particular answer rather than correctness, and many tasks admit many correct answers or no reference at all. This thesis therefore works with feedback that assesses the output itself, obtained from learned evaluators, human judgments, program execution, and interaction with environments.

Figure 1.1 introduces an example used throughout this section to compare these four sources. The task is to implement `lis(nums)`, which returns the length of the longest strictly increasing subsequence of a list, and four candidate programs are shown. Programs A and C are both correct but use different algorithms, so no single reference would match both. Program B accepts equal values as an increase, and Program D fails on an empty input. Each source of feedback sees a different part of this picture, and the paragraphs below use the same four programs to show what each source detects, what it misses, and what it costs.

Learned feedback. Learned feedback comes from a model that assesses a generated output in the context of a task. The evaluator may be trained on human ratings, preference comparisons, or judgments produced by a stronger model. In Figure 1.1, an evaluator could explain that Program B uses `bisect_right`, allowing equal values to extend the

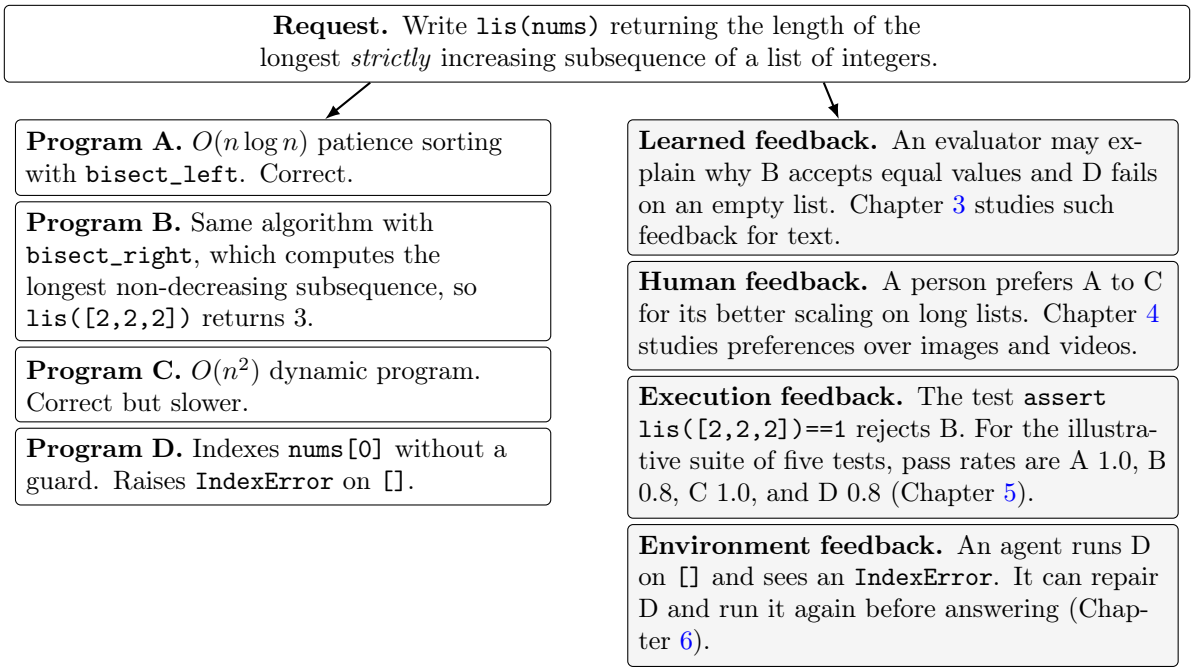


Figure 1.1: One request, four candidate programs, and four sources of feedback. The example is illustrative, with pass rates from a hypothetical suite of five tests. Tests distinguish B and D from A, but give A and C the same pass rate. A learned evaluator can explain an error without execution, although its judgment may be wrong.

subsequence, and suggest `bisect_left` instead. It could also identify the missing check for empty input in Program D. Such an evaluator can provide scores and explanations for many outputs without collecting a new human judgment for each one. Its assessment can still be mistaken. For example, it might overlook B's error or report a problem in the correct Program A, so the explanation itself needs to be checked. Chapter 3 studies this kind of feedback for generated text.

Human feedback. Human feedback comes from people assessing generated outputs. In the example, Programs A and C pass the same tests, but a person may prefer A because its $O(n \log n)$ algorithm scales better to long lists than C's $O(n^2)$ algorithm. The preference expresses a criterion that the pass rate does not capture. For images and videos, people may similarly weigh prompt fidelity, visible artifacts, and aesthetic preferences when comparing two outputs. Chapter 4 collects such comparisons to rank generators and assess automatic judges. These judgments take time to collect, and people may disagree about

which qualities matter most. A set of votes therefore reflects the prompts, voters, and comparison procedure through which it was collected.

Execution feedback. Execution feedback comes from running a generated program and checking its behavior against tests. For Program B, the test `assert lis([2,2,2]) == 1` fails because the program returns 3, and the test `assert lis([]) == 0` fails for Program D, which raises an `IndexError`. These results expose errors that are difficult to detect from the program's appearance alone. Once tests are available, they can be applied to any number of candidate programs without a separate human judgment for each one. Their scope is limited by what they check. A and C receive the same pass rate in the example despite their different computational costs, and a program may pass every available test while still failing on an input that no test covers. The tests themselves must also be correct, which becomes a concern once they are generated automatically rather than written by hand. Chapter 5 studies how to construct tests at scale and turn execution results into feedback for selection and learning.

Environment feedback. Models are increasingly used as agents that solve a task over several turns, calling tools such as a code interpreter, a search engine, or a database along the way. Environment feedback is what the environment returns during and after such a trajectory, and it takes more than one form. An agent that has written Program D can run `lis([])` before submitting, read the resulting `IndexError`, add a check for the empty list, and run the program again. The error message is an observation. It enters the policy's context and shapes the next action, but it is not a reward. The reward comes from a separate check on the final answer once the trajectory ends, and a trainer may add terms that score how the agent acted, such as whether it ran its program at all. The failed call raises a question of its own, since a tool error or a malformed action can be shown to the agent, penalized, or dropped from training. Among these, only the final check is tied to what the task requires. A term that pays for tool use can be collected by a call that does nothing, an observation can be cut short by a token cap or ignored by a trained policy, and dropping failed trajectories changes what the policy is trained on. Chapter 6 builds a framework that runs tools inside the training loop and examines two of these choices, the rewards for tool use and the observations the policy reads.

These four sources are distinguished by where the information originates rather than by the form in which it arrives. A preference pair may come from a person comparing two programs or from a difference in their test pass rates, and the two carry different information despite their identical form. Sources also feed one another. Chapter 5 turns

execution results into preference labels for training a learned reward model, and Chapter 6 returns them to an agent as observations.

Feedback from any source can serve three uses. In evaluation, it scores the programs a model produces so that models can be compared. In candidate selection, it picks among A, B, C, and D for one task. Neither changes the generator. In learning, test results or preference labels update the generator so that it produces programs like A and C more often. Part I studies the first use and Part II the other two. The move to learning adds a requirement that the first two uses do not have, that higher reward must keep corresponding to better task performance while the model is optimized against the feedback. When it does not, the policy exploits the feedback instead of solving the task, a failure known as reward hacking that Section 7.1 examines for each of the four sources.

1.2 Research Questions and Contributions

The two uses of feedback place different demands on it. For evaluation, feedback has to agree with human judgment and say why. For learning, it has to stay reliable while a model is optimized against it. Each part of the thesis takes up one of these demands through one research question.

RQ1. How can feedback on generated outputs be obtained that agrees with human judgment and explains its own?

RQ2. How can feedback be designed to remain reliable while a model is optimized against it?

Part I answers RQ1 with a trained evaluator that explains errors in text and an open platform that collects human votes on images and videos. For evaluation, the requirement is agreement with people and, for text, an explanation they can check. RQ2 asks more, because agreement measured on fixed outputs does not predict what happens once a policy is trained to score well under the feedback. At the end of Chapter 3, a policy trained against the learned evaluator learned to refuse rather than answer. Part II therefore takes its training feedback from observed outcomes, the results of running a program or acting in an environment, and keeps human judgment as the reference against which that feedback is checked rather than as the training signal. Chapter 7 compares how each of the four sources can be exploited.

Contributions.

1. **Explainable learned feedback for text generation** (Chapter 3). Feedback for text can be learned without a reference output. TIGERScore is trained on MetricInstruct, 42K examples over six tasks and 23 datasets, to produce a reference-free analysis of each error with a score. It correlates with human ratings better than every compared metric except GPT-4 used as a judge, which it trails by 2.65 points in average Kendall correlation, and human annotators rated 70.9% of its analyses 3 or higher out of 5. Its agreement with people has limits. As a Best-of-8 selector it barely beat a random sample, 92.2% against 91.3% on GSM8K.

2. **Human feedback for images and videos at scale** (Chapter 4). Human comparisons collected openly serve both to rank generators and to test automatic judges. GenAI-Arena collects anonymous pairwise votes on text-to-image generation, image editing, and text-to-video generation, ranks models with a Bradley–Terry fit, and releases the cleaned votes as GenAI-Bench. The best multimodal judge tested on it agrees with human votes about half the time.

3. **Execution feedback for code at scale** (Chapter 5). Execution results can supply the preferences and rewards that human labels supply elsewhere. AceCoder synthesizes tests for existing coding questions, turns the pass rates of sampled programs into preference pairs, and trains Bradley–Terry reward models. Best-of-64 selection with the 7B reward model raises Llama-3.1-8B-Instruct's average from 40.9 to 49.3, and reinforcement learning from Qwen2.5-Coder-7B-Base with a test-pass reward raises HumanEval+ from 53.0 to 78.0 in 80 steps.

4. **Environment feedback for agents** (Chapter 6). VerlTool supports reinforcement learning over multi-turn tool interactions through a unified tool server and asynchronous rollouts, with up to 1.97× rollout speedup in the reported comparison. Agents trained in six domains achieve competitive results against specialized systems. Additional experiments in Section 6.5 examine auxiliary rewards and observation length, distinguishing increased tool use from improved task performance and showing how the information returned by tools affects results.

The question of how candidate outputs should be judged first arose in earlier work on ranking and fusing the outputs of several language models (Jiang et al., 2023). Later work by my collaborators and me extends each chapter and is described in Section 7.2, and none of it is claimed as a contribution of this thesis.

1.3 Thesis Organization

Chapter 2 gives the background the research chapters rely on, how outputs are judged and ranked, how judgments become training rewards and how optimizing against them fails, and how agents learn from tool interaction over several turns. Part I studies feedback for evaluation, with learned feedback for text in Chapter 3 and human feedback for images and videos in Chapter 4. Part II studies feedback for improving models, with execution feedback for code in Chapter 5 and environment feedback for agents in Chapter 6. Each of these four chapters is adapted from one paper (Table 1.1), can be read on its own, and ends with its limitations and what the next chapter takes up. Chapter 7 compares how each of the four sources was or could be exploited during training, then places the chapters among subsequent work and future directions, and Chapter 8 returns to the two research questions. The analyses added for this thesis beyond the published papers are listed in the Statement of Contributions.

Chapter	Feedback	Output judged	Adapted from
3	learned	text	TIGERScore (Jiang et al., 2024c), TMLR 2024
4	human	images, videos	GenAI-Arena (Jiang et al., 2024b), NeurIPS 2024
5	execution	programs	AceCoder (Zeng et al., 2025), ACL 2025
6	environment	trajectories	VerlTool (Jiang et al., 2026), TMLR 2026

Table 1.1: The research chapters, the source of feedback each studies, the outputs it judges, and the publication each is adapted from.

Chapter 2

Background

This chapter reviews the background the research chapters rely on, how outputs are judged and ranked (Section 2.1), how judgments become training rewards and how optimizing against them can fail (Section 2.2), and how agents are trained through tool interaction over several turns (Section 2.3).

Notation. I use x for an input, such as a question or instruction, and $y = (y_1, \ldots, y_T)$ for an output of T tokens. A language model used to generate outputs is a policy π_θ, with parameters θ. An evaluator assesses an output given the task input and, when required, a reference answer. Its assessment may be a score, a preference between outputs, a pass or fail verdict, or an explanation of errors, and a preference pair (x, y_w, y_l) records that y_w was preferred to y_l, whether by a person or by a test. When a numerical assessment is used to select outputs or train a policy, I denote it by $R(x, y)$ and refer to it as a reward. R_ϕ denotes a learned reward model. I use π_{ref} for a fixed reference policy and β for the coefficient of a KL penalty. Notation specific to multi-turn interaction is introduced in Chapter 6.

Generative models. A language model generates an output by predicting each token from the input and the tokens already produced (Vaswani et al., 2017; Brown et al., 2020),

$$\pi_\theta(y \mid x) = \prod_{t=1}^{T} \pi_\theta(y_t \mid x, y_{<t}). \tag{2.1}$$

Sampling from these predictions can produce several different responses to the same input, which allows an evaluator to choose among candidates. The model can also be fine-tuned

on examples of a desired response, using supervised learning to minimize

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\sum_{t=1}^{T} \log \pi_\theta(y_t \mid x, y_{<t})\right], \tag{2.2}$$

where $\mathcal{D}$ contains input–response pairs. Instruction tuning applies this objective to natural-language instructions and their responses (Wei et al., 2022a; Chung et al., 2024). The same objective can train a model to evaluate. The input then contains a task and a candidate response, and the target is an assessment of that response. Chapter 3 uses this approach to train an evaluator that writes error analyses.

Image and video generation place different requirements on evaluation. Text-to-image models, including latent diffusion models (Rombach et al., 2022; Podell et al., 2024), can produce many images that satisfy the same description while differing in composition, realism, and visible artifacts. Instruction-based editing adds a source image and asks for a particular change (Brooks et al., 2023), and a successful edit must make that change while preserving relevant content elsewhere. Video generation also requires consistency across frames and appropriate motion (Ho et al., 2022; Blattmann et al., 2023). These requirements explain why matching one reference output is often insufficient and why the human comparisons in Chapter 4 consider several aspects of quality together.

2.1 Feedback for Evaluation

Reference-based and learned text metrics. Automatic text metrics allow researchers to evaluate many outputs without obtaining a separate human judgment for each one. BLEU (Papineni et al., 2002) measures n-gram overlap with reference translations, while ROUGE (Lin, 2004) measures overlap with reference summaries. Such comparisons are useful when references are available, but valid responses can use different words, and a small change can reverse meaning while preserving most of the wording. Studies of translation and other text-generation tasks have therefore examined where overlap scores agree with human judgments and where they fail, particularly when used to assess individual outputs (Callison-Burch et al., 2006; Novikova et al., 2017; Reiter, 2018).

Model-based metrics use pretrained representations or additional training to assess qualities that exact matching misses. BERTScore (Zhang et al., 2020) compares candidate and reference tokens through their contextual embeddings, allowing related words to match without being identical. BLEURT (Sellam et al., 2020) and COMET (Rei et al., 2020) learn to predict human assessments, while BARTScore (Yuan et al., 2021) uses conditional

generation probabilities and UniEval (Zhong et al., 2022) formulates evaluation aspects as questions. A reference-free metric instead evaluates the output using the task input without a reference answer, making it applicable when collecting references is difficult. TIGERScore is trained on error analyses written by GPT-4 and may inherit their errors, and its reliability on outputs unlike its training data has to be checked separately.

Error-based and explainable evaluation. Error-based evaluation records the problems behind an assessment, allowing a reader to inspect the judgment rather than only its score. In machine translation, Multidimensional Quality Metrics (MQM) provides categories for annotating errors and their severity (Freitag et al., 2021). An annotation can identify a mistranslated span, classify the problem, and distinguish a minor issue from one that changes the meaning. InstructScore (Xu et al., 2023d) trains a language model to produce a diagnostic report with error locations, types, explanations, and severities. This connects a score to specific claims about the output, although those claims can themselves be wrong or incomplete. Chapter 3 studies reference-free error analysis across text-generation tasks and evaluates both the scores and the analyses.

Automatic measures for images and videos. Visual evaluation also includes methods that measure different aspects of generated outputs. FID (Heusel et al., 2017) compares feature distributions of generated and real images, so it evaluates a set of samples rather than whether one image fulfills a particular request. CLIPScore (Hessel et al., 2021) uses image and text embeddings to assess their compatibility, providing a reference-free measure of alignment. ImageReward (Xu et al., 2023b), PickScore (Kirstain et al., 2023), and HPSv2 (Wu et al., 2023) learn from human preferences over generated images, while VBench (Huang et al., 2024b) evaluates video dimensions such as subject consistency and motion. CLIPScore measures alignment but not artifacts, and the preference-trained scorers are fixed to the comparisons they were trained on. Chapter 4 collects new human comparisons and measures how well automatic judges reproduce them.

LLM-as-a-judge. A language model can also be given an evaluation instruction and asked to score an output, compare two candidates, or explain its judgment. G-Eval (Liu et al., 2023b) uses a prompted language model to evaluate text along specified criteria, and MT-Bench (Zheng et al., 2023a) studies model judgments of conversational responses. This allows evaluation requirements to be expressed in natural language rather than implemented separately for every task. Judgments can also provide training targets for smaller or open models, as in Prometheus (Kim et al., 2024a,b). However, a judge may favor an

answer because it is longer, appears first, or resembles its own model's responses (Zheng et al., 2023a; Wang et al., 2024a; Dubois et al., 2024). Its agreement with people therefore needs to be measured for the outputs and evaluation procedure in which it will be used.

Multimodal language models extend this approach to visual outputs by reading an instruction together with images or video frames. For example, VIEScore (Ku et al., 2024a) prompts a multimodal model to assess conditional image generation and explain its scores. Chapter 4 measures how closely such judgments follow human preferences by asking multimodal models to judge the same image and video comparisons that people voted on.

Human evaluation and arenas. Human evaluation asks people to assess outputs according to the task requirements, either by assigning ratings or by comparing candidates. Comparisons avoid requiring every rater to interpret a numerical scale in the same way, although their reliability still depends on the task and annotation procedure (Kiritchenko and Mohammad, 2017). An arena collects such judgments through a platform where users compare anonymous model outputs. Chatbot Arena (Chiang et al., 2024), for example, presents two responses to a user's prompt and collects a preference before revealing the model identities. The resulting votes reflect requests made by users rather than only a fixed benchmark. They are slow to collect and noisy, and the resulting ratings depend on who voted, on which outputs were paired, and on which prompts.

Ranking from pairwise comparisons. Combining pairwise votes into a model ranking requires accounting for which opponents each model faced. A model that wins often against weak opponents need not be stronger than one that wins less often against strong opponents. The Bradley–Terry model (Bradley and Terry, 1952) assigns each model a strength s_i and expresses its probability of winning a decisive comparison as

$$P(i \succ j) = \sigma(s_i - s_j), \qquad \sigma(u) = \frac{1}{1 + e^{-u}}. \tag{2.3}$$

The strengths are fitted by maximizing the log-likelihood $\sum_{i \neq j} W_{ij} \log \sigma(s_i - s_j)$, where W_{ij} counts wins by i over j (Hunter, 2004). The model summarizes preferences with one strength per generator, so when preferences vary by prompt or opponent, the fitted ranking also depends on how frequently those comparisons occur.

The logistic form of the Elo rating system expresses the same expected score on a different scale (Elo, 1978),

$$E_A = \frac{1}{1 + 10^{(e_B - e_A)/400}}, \qquad e'_A = e_A + K(S_A - E_A), \tag{2.4}$$

where S_A is the observed score and K controls the size of the update. Elo updates ratings as comparisons arrive, whereas a Bradley–Terry fit estimates strengths from the full collection of votes. The two use the same probability model when $s_i = e_i \ln 10/400$, allowing fitted strengths to be reported on the familiar Elo scale. Chapter 4 uses a Bradley–Terry fit and examines the uncertainty and uneven comparisons behind its rankings.

Execution-based evaluation. When the output is a program, running it against tests checks its behavior on concrete inputs. Benchmarks such as HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) pair programming problems with tests. A common measure, pass@k, is the probability that at least one of k sampled programs passes all the tests (Kulal et al., 2019; Chen et al., 2021), while a program's pass rate is the fraction of tests it passes and can distinguish partially successful candidates. A test needs an input and an oracle that determines the expected behavior, and obtaining that oracle can be difficult (Barr et al., 2015). A pseudo-oracle uses another implementation as a source of expected behavior (Davis and Weyuker, 1981; Weyuker, 1982), providing evidence of agreement rather than a guarantee of correctness. Passing every test only establishes success on the inputs the suite contains. EvalPlus (Liu et al., 2023a) expands test suites to expose errors missed by the original checks. A flaky test can pass and fail on the same code and configuration (Luo et al., 2014), and passing functional tests does not establish efficiency. Chapter 5 generates tests at scale and filters them by checking agreement with a proxy solution.

Validating an evaluator. Evaluating an automatic metric requires judgments against which its assessments can be checked, a process often called meta-evaluation. For numerical ratings, Pearson's correlation measures linear association, while Spearman's correlation compares ranks. Kendall's correlation counts pairs of outputs that the metric and the reference judgments order in the same or opposite directions. With m scored outputs, its τ_a form is

$$\tau_K = \frac{\#\,\text{concordant pairs} - \#\,\text{discordant pairs}}{\binom{m}{2}}. \tag{2.5}$$

Chapter 3 reports Kendall's τ between evaluator scores and human ratings over the individual outputs of each task. For a pairwise judge, Chapter 4 instead reports how often the judge picks the same winner as the human vote on the same comparison.

2.2 Learning from Feedback

Reward models. Feedback can guide model improvement by identifying which outputs should be selected or made more likely during training. When obtaining a judgment for every sampled response is costly, a reward model learns to predict preferences from previously judged examples (Christiano et al., 2017; Stiennon et al., 2020; Ouyang et al., 2022). Each training example contains an input x, a preferred response y_w, and a less preferred response y_l. Applying the Bradley–Terry model to these responses, with strengths predicted by $R_\phi(x, y)$, gives the loss

$$\mathcal{L}_{\mathrm{RM}}(\phi) = -\mathbb{E}_{(x,y_w,y_l)}\left[\log \sigma\big(R_\phi(x, y_w) - R_\phi(x, y_l)\big)\right]. \tag{2.6}$$

This uses the same comparison model as arena ranking, but computes strength from the response instead of fitting a separate strength for each generator. The resulting reward model can score new responses, including those produced by a policy during training.

Benchmarks such as RewardBench (Lambert et al., 2025b) and RM-Bench (Liu et al., 2025b) test how often they rank a preferred response above its alternative. An outcome reward model assesses a completed response, while a process reward model assesses intermediate steps, allowing supervision to distinguish correct reasoning from a correct answer reached through faulty steps (Uesato et al., 2022; Lightman et al., 2024). The labels used for either form may come from human judgments, another model, or observed outcomes, and they determine what the reward learns to recognize. Chapter 5 uses differences in test pass rates to construct preference pairs and train outcome reward models for code.

Best-of-N selection. A reward model can improve the returned answer without updating the generator via best-of-N sampling, which returns the one with the highest reward of the N generated responses,

$$y^\star = y_{i^\star}, \qquad i^\star = \arg\max_{1 \le i \le N} R_\phi(x, y_i), \quad y_i \sim \pi_\theta(\cdot \mid x). \tag{2.7}$$

Its benefit depends on both the candidates available and the evaluator's ability to distinguish them, since selection cannot recover a correct answer that was never sampled. The relevant comparisons are among responses to the same input, which may be harder to distinguish than outputs sampled from different tasks. Keeping selected responses as supervised training targets extends this idea to rejection sampling fine-tuning (Dong et al., 2023), whereas the selection experiments in Chapters 3 and 5 keep the generator fixed.

RLHF and preference optimization. Reinforcement learning changes the policy so that responses with higher reward are more likely to be produced. In reinforcement learning from human feedback (RLHF), the reward is usually supplied by a model trained on human preferences (Stiennon et al., 2020; Ouyang et al., 2022). A common objective combines expected reward with a penalty for departing from a fixed reference policy,

$$\max_{\theta} \; \mathbb{E}_{x\sim\mathcal{D}}\left[\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[R_\phi(x,y)] - \beta\,\mathbb{D}_{\mathrm{KL}}\big(\pi_\theta(\cdot \mid x)\,\|\,\pi_{\mathrm{ref}}(\cdot \mid x)\big)\right]. \tag{2.8}$$

The KL penalty discourages departures from the reference, although it does not guarantee that the reward remains accurate on the new outputs. Proximal Policy Optimization (PPO) (Schulman et al., 2017) optimizes this objective on sampled responses, clipping the ratio between the updated and sampling policies' token probabilities so that one update cannot move the policy too far.

RLVR and GRPO. When an output can be checked by a program, the result can provide the reward directly. Reinforcement learning with verifiable rewards (RLVR) (Lambert et al., 2025a) includes checking mathematical answers and precise instruction constraints, and code-generation methods had earlier used unit-test outcomes for training (Le et al., 2022; Shojaee et al., 2023). A binary reward gives 1 to a correct response and 0 otherwise, while other checks return the fraction of requirements satisfied, such as a test pass rate. These rewards score many responses without a learned judge, but what they measure is whatever the check measures, an answer match or a set of tests.

Group Relative Policy Optimization (GRPO) (Shao et al., 2024) estimates advantages by comparing responses to the same input, avoiding a learned value function. For a group of G responses with rewards $R_1, \ldots, R_G$, it assigns response i the normalized advantage

$$\hat{A}_i = \frac{R_i - \mathrm{mean}(\{R_j\}_{j=1}^{G})}{\mathrm{std}(\{R_j\}_{j=1}^{G})}. \tag{2.9}$$

This advantage weights the response's generated tokens in a PPO-style clipped objective. When all rewards in a group are equal, the advantages are zero and the group contributes no policy-gradient signal. DeepSeek-R1-Zero (Guo et al., 2025) demonstrates how GRPO with rule-based rewards can improve mathematical and coding performance, and Chapter 6 applies group-based optimization to tool interactions.

Reward shaping and auxiliary terms. An agent rewarded only for reaching the exit of a maze may receive no positive reward during many unsuccessful attempts, making it

difficult to learn which actions lead toward the goal. Reward shaping supplements the task reward with auxiliary terms that provide additional guidance during learning (Ng et al., 1999). These terms can take the form of bonuses for intermediate progress or exploration, or penalties for invalid actions. Their purpose is to make useful behavior easier to learn, although improving the auxiliary reward does not necessarily improve task performance. Chapter 6 explores these choices in tool-interactive learning, examining how tool-use bonuses and execution-error penalties affect interaction behavior and final-answer accuracy.

Reward hacking and overoptimization. Optimizing a reward can expose differences between what it measures and what the task requires. For example, a policy rewarded for detailed answers may learn to add text that raises the score without improving the answer. This is an instance of Goodhart's law, under which a measure's usefulness can change when it becomes a target for optimization (Manheim and Garrabrant, 2018; Amodei et al., 2016). Reward hacking refers to behavior that exploits such differences, and formal analyses show that guarantees against reward gaming impose strong restrictions (Skalse et al., 2022). Gao et al. (2023) study overoptimization by increasing a learned proxy reward while measuring a separate gold reward, which eventually falls under both Best-of-N selection and reinforcement learning. An incomplete verifier or a behavioral bonus can be exploited the same way.

Agreement with people on fixed outputs, the test of Section 2.1, does not settle whether a reward can be trained on. Training moves the policy toward outputs the evaluator has not seen, and those are the outputs on which it may be wrong. The test that matters is therefore made during training, by measuring task performance with something independent of the reward, such as held-out accuracy or additional tests, and checking that it rises along with the reward. Chapters 3, 5, and 6 examine selection and training this way.

2.3 Tool Interaction and Agent Training

Language agents use external tools while solving a task, allowing later actions to depend on information obtained during the attempt. ReAct (Yao et al., 2023) alternates reasoning, actions, and observations, while Toolformer (Schick et al., 2023) trains a model to insert useful API calls. An action may run a program, issue a search query, inspect a database, or modify a file in a software repository (Wang et al., 2024c; Lewis et al., 2020; Yang et al., 2024b). The environment returns an observation, such as a program result, retrieved

passage, or error message, which becomes part of the context for the next action. Over several turns, these actions and observations form a trajectory ending in a final response or another termination condition. The interaction allows an agent to inspect the consequences of its actions and revise its approach before submitting an answer.

Observations and rewards serve different roles in this interaction. An interpreter traceback tells the policy what happened and can help it repair its code, while a task checker supplies a numerical assessment used for training. The trainer may also penalize the error, but returning the traceback does not itself specify such a penalty. Similarly, a retrieved passage can help answer a question without carrying a reward for retrieving it. The final outcome reward can be combined with auxiliary terms for behavior, as discussed in Section 2.2. Tool failures raise an additional choice, whether to return an error observation, assign a penalty, or exclude the failed trajectory from training.

Training on interleaved actions and observations requires tracking which tokens the policy generated. Observation masking excludes tool-returned tokens from the policy's generation loss while retaining them in the context for subsequent actions (Jin et al., 2025). A multi-turn trainer must also execute tool calls, preserve environment state across turns, and manage calls with different runtimes. Systems such as Search-R1 (Jin et al., 2025) and ReTool (Feng et al., 2026) train tool-using policies with outcome rewards, supported by reinforcement learning infrastructure such as veRL (Sheng et al., 2025). Chapter 6 introduces the multi-turn objective and a framework that serves many tools and tasks, then studies how outcome rewards, auxiliary terms, observations, and failed trajectories influence learning.

Part I

Evaluating Generative Models with Feedback

Overview of Part I

This part studies feedback for evaluation, asking how to obtain assessments that agree with human judgments. It examines learned feedback for generated text and human preferences collected directly for generated images and videos.

Chapter 3 studies whether an evaluator can provide reliable scores and explanations without a reference output. I develop TIGERScore, which learns from error analyses produced by a stronger model to identify and explain errors across text generation tasks. I evaluate its scores against human ratings and ask people to assess its explanations. The results show that a trained evaluator can produce scores that agree well with human ratings while explaining the errors it identifies.

Chapter 4 turns to image and video generation, where evaluation involves subjective judgments of visual quality and aesthetics. Human preferences are particularly valuable in this setting, as people can weigh these qualities alongside how well an output follows the user's request. I collect these preferences through GenAI-Arena, an open voting platform where users compare anonymous outputs generated for the same request. I use the collected votes to rank generative models and release them as GenAI-Bench to assess how well automatic judges agree with human preferences. I further show that the multimodal LLMs tested often disagree with human votes, limiting their ability to replace human evaluators. For images and videos, human preferences therefore remain the reference, both for ranking generators and for judging the judges.

Chapter 3

Learned Feedback for Generated Text

3.1 Introduction

This chapter studies the first source of feedback, a learned evaluator, for generated text. Though GPT-4 has shown to achieve strong evaluation correlation with humans (Zheng et al., 2023a), the open-sourced metrics are lagging significantly behind and mainly suffer from specific issues:

Dependency on references: Some evaluation metrics like ROUGE (Lin, 2004), BLEU (Papineni et al., 2002), COMET (Rei et al., 2020), InstructScore (Xu et al., 2023d) would require gold references to measure the quality. These metrics compare the generated output against one or more reference texts to assign the evaluation score. However, such an assumption can be highly unrealistic in many downstream applications where the gold reference is hard to collect.

Limited to specific domains: Some evaluation metrics are limited to specific domains, lacking the ability to generalize to broader text generation tasks. For example, COMET (Rei et al., 2020), BLEURT (Sellam et al., 2020), SESCORE2 (Xu et al., 2023c) and InstructScore (Xu et al., 2023d) are specifically designed for tasks like machine translation or WebNLG tasks.

Lack of attribution: Some evaluation metrics tend to directly output a score without any attributions, e.g., where the errors occur and why. For instance, BARTScore (Yuan et al., 2021), BERTScore (Zhang et al., 2020) and GPTScore (Fu et al., 2023) adopt the pre-trained language models' log likelihood as the evaluation metric. Such metrics do not

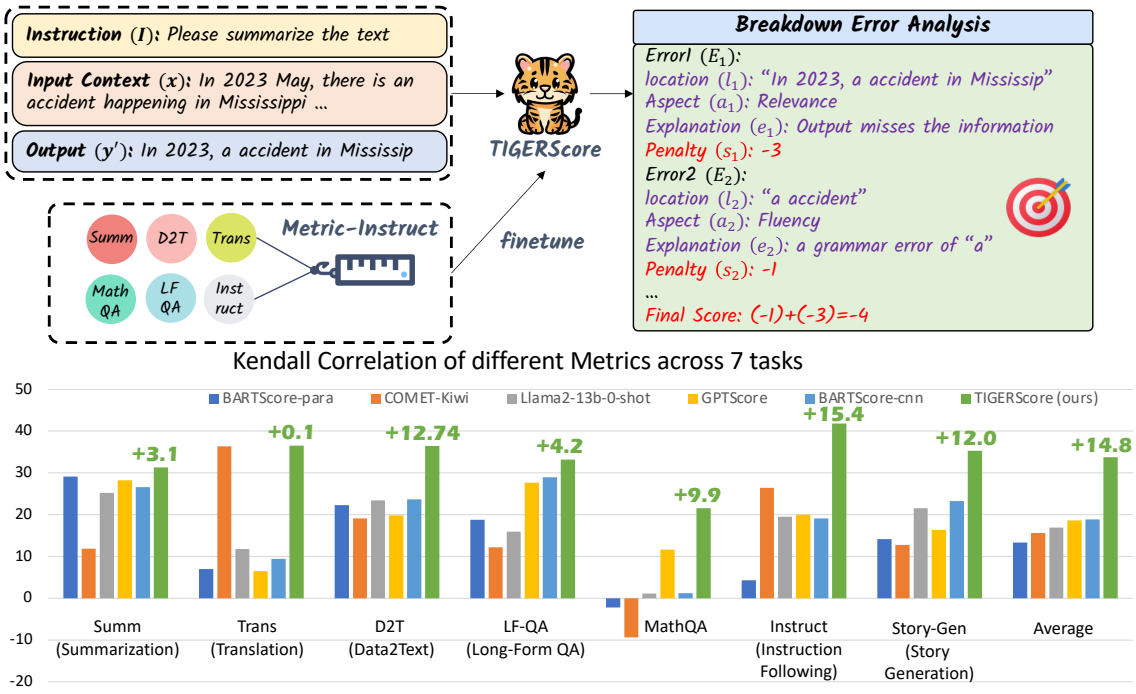


Figure 3.1: The upper part shows the input and output format of TIGERScore, which takes an instruction and an input context, along with the model-generated output that is to be evaluated as the inputs, and output detailed breakdown error analysis with explanations. It is finetuned on our curated dataset `MetricInstruct`. The lower part shows the Kendall's correlation of different metrics w.r.t human ratings, from which we can see TIGERScore achieves the SoTA correlation on most generation tasks.

provide the location and reason for the assigned score, which limits their trustworthiness and reliability.

To address these issues, we propose TIGERScore, a **T**rained metric that follows **I**nstruction **G**uidance to perform **E**xplainable and **R**eference-free evaluation. As shown in Figure 3.1, the input to TIGERScore consists of an instruction describing the task definition, the task input, and the system output. TIGERScore is capable of generating breakdown error analysis that can (1) locate each mistake, (2) explain the mistake and suggest revisions, and (3) provide a penalty score (between [-5, -0.5]) for each mistake. The final score for the system output can be calculated by summing up all the penalty scores.

TIGERScore is built by fine-tuning LLMs on our curated `MetricInstruct` dataset, which contains a total of 42K examples of (instruction, input, system output, error analysis), obtained from 23 diverse text generation datasets. The dataset includes system outputs from more than 50 systems, covering a wide variety of errors. The error analysis is curated by prompting GPT-4 (OpenAI, 2023) and filtered through various strategies. The tuned model TIGERScore has shown the highest overall Kendall's correlation with human ratings among open-source metrics on seven major text generation tasks as depicted in Figure 3.1. TIGERScore does not require any additional reference. We evaluate TIGERScore on 5 held-in datasets and 2 held-out datasets. As a reference-free metric, TIGERScore surpasses the best reference-free metrics (GPTScore (Fu et al., 2023)) by 15% and surpasses the best reference-based metrics (COMET-22 (Rei et al., 2022a)) by a margin of 8% in terms of Kendall's score. On the two held-out datasets, TIGERScore also leads the open-source metrics. We further employ humans to evaluate the explanations generated by TIGERScore and found that over 70% of the generated explanations were rated accurate. Our analysis shows that the success of TIGERScore is attributed to three key aspects in our curated `MetricInstruct`: (1) dataset diversity, (2) error coverage, and (3) high quality, which enable TIGERScore to generalize better on unseen tasks than the other open-source metrics.

3.2 Related Work

3.2.1 Instruction-driven Large language models

Instruction tuning, reviewed in Section 2, is normally used to make a model follow general instructions. This chapter uses the same training objective to activate one specialized capability, generating an error analysis according to an instruction, with data distilled from GPT-4 in the manner of Wang et al. (2023c) and Xu et al. (2023a).

3.2.2 Explainable Metrics

Section 2.1 introduces error-based evaluation and InstructScore. The related work here is specific to explainable metrics for text. The increasing focus on model interpretability has led to a surge in research dedicated to explainable metrics. Research in these fields aims to build a metric system for a certain task that is readable to humans and is expected to help the development of better text generation systems (Leiter et al., 2022). Early endeavors in

this area delved into explainability via multi-faceted evaluations, as exemplified by works such as Unieval (Zhong et al., 2022) and GPTScore (Fu et al., 2023). As LLM blooms, researchers began to directly prompt LLMs to create interpretable metrics.

One instance is PandaLM, trained on Llama to compare two responses pairwisely and provide a textual rationale for its decisions (Wang et al., 2023b). Another noteworthy approach is InstructScore, leveraging large language models as knowledge repositories to obtain premium error analysis examples (Xu et al., 2023d).

Despite these commendable advancements, most existing explainable metrics still require gold references and are often limited concerning the task domain. Our contribution distinguishes itself by offering a reference-free nature and the cross-task ability brought by instruction-tuning over large language models, aiming to serve as a universal explainable metric.

3.3 Method

3.3.1 TIGERScore

TIGERScore is built upon three design criteria: (1) It is driven by instructions, making it easily adaptable to any text generation task. (2) The evaluation process eliminates the need for a "gold standard" or perfect example for comparison. (3) It is highly explainable, as the model can generate an error analysis that helps the user understand each identified mistake and its associated penalty.

Background

The pursuit of improved metrics for text evaluation has been a significant focus since the inception of language models. Automatic n-gram-based metrics (Elliott and Keller, 2014; Callison-Burch et al., 2006; Isozaki et al., 2010) have always served as the default metric, computing the n-gram match F-1 score with the reference text until research highlighted their significant weaknesses in aligning with human preferences. Later, neural metrics were proposed to better capture the semantic similarity in addition to mere morphological similarity only by either computing based on neural representation (Zhang et al., 2020; Yuan et al., 2021) or directly fine-tuning with human preferences (Rei et al., 2020). It has also been demonstrated that multi-aspect scores, using the logits of large language

models with well-designed instructions for various aspects as prompts (Fu et al., 2023), could achieve an even higher correlation.

There have been some attempts to build explainable metrics leveraging the great capacity of large language models. For instance, UniEval (Zhong et al., 2022) constructs a multi-aspect evaluation system by individually training on aspect-specific data. PandaLM (Wang et al., 2023b) compares two responses for a given instruction and input to judge which is better, providing a short reason for its decision. InstructScore (Xu et al., 2023d) evaluates the quality of translation by training Llama-2 to compare the reference and translation, listing errors with structured information. However, none of these metrics has been able to address all the issues mentioned in section 3.1 concurrently.

Problem Formulation

Suppose y' is the system output from a given source context x with a specific natural language instruction I to describe the task definition. And y is a corresponding reference output. If a metric uses y, it's reference-based, otherwise, it's reference-free. For example, when T refers to "translation", an instruction I for that task could be "Translate the following text from Chinese to English". For each task type T, we ask the evaluation metric to focus on a few pre-defined evaluation aspects A_T like relevance, factuality, fluency, etc.

TIGERScore is a reference-free metric, defined by a function f to take the triple (I, x, y') as input, to produce a list of structured errors $\{E_1, ..., E_m\}$ as the otuput, called the error analysis content:

$$\{..., E_i, ...\} = \{..., (l_i, a_i, e_i, s_i), ...\} = f(I, x, y') \tag{3.1}$$

where (l_i, a_i, e_i, s_i) denotes specific information of the error E_i. Specifically, l_i points to the location of the error, and $a_i \in A_T$ is a pre-defined aspect to which this error belongs. e_i comprises both the explanation of this error and its revision suggestion. s_i is the penalty score reduced for this error, which lies in $[-5, -0.5]$. The final score of y' is computed as the sum of all the penalty scores: $s_{y'} = \sum_i s_i$. The range of the final score lies within $(-\infty, 0]$, where 0 means perfect generation and a lower score indicates worse quality and more severe errors. However, in practice, the number of detected errors is affected by both the training data and the generation length, thus leading to less than 10 errors most of the time during the evaluation.

Instruct	
Aspect	Definition
Comprehension	Evaluates how well the output understands the given instruction.
Accuracy	Measures the correctness of the output in relation to the instruction and the paired input context.
Informativeness	Assesses the relevancy and usefulness of the information provided by the output.
Coherence	Evaluates how logically the output flows and connects.

Table 3.1: Definitions of evaluation aspects of TIGERScore for Instruction-following (Instruct) as an example. See full table in Table B.2 for aspect definitions of all 6 text generation tasks

Multi-Aspect Evaluation

As stated in the problem formulation, each error is considered to be attributed to a certain aspect that the system output might make mistakes on. For each task, 3 or 4 aspects are designed with the help of both GPT-4 prompting and human supervision, detailed in subsection B.1.6. These aspects are designed to be both non-overlapping (mutually exclusive) and collectively exhaustive. For instance, 4 aspects to evaluate an output for the instruction-following tasks is shown in Table 3.1.

3.3.2 MetricInstruct

We present the `MetricInstruct` dataset, which is employed to fine-tune TIGERScore. The three underlying criteria for dataset construction are: (1) **dataset diversity**: we choose 23 distinctive datasets as the source context to cover enough generation tasks. (2) **error coverage**: we take system outputs generated from 50+ text generation systems to cover all types of errors and guarantee a balanced distribution. (3) **quality ensurance**: to ensure `MetricInstruct` is tailored to gather in-depth error analysis as detailed in section 3.3.1, we sourced it by prompting GPT-4 (OpenAI, 2023) and then filtered through different heuristics to eliminate low-quality error analysis.

Diverse Dataset Source

`MetricInstruct` incorporates samples from 23 distinctive text generation datasets, which are categorized into 6 major categories of text generation tasks. While the collection encompasses well-researched tasks such as summarization (Summ), translation (Trans), and data2text (D2T), it also introduces popular new tasks like Long-Form QA (LF-QA), MathQA, and instruction-following (Instruct). These latter tasks have witnessed limited

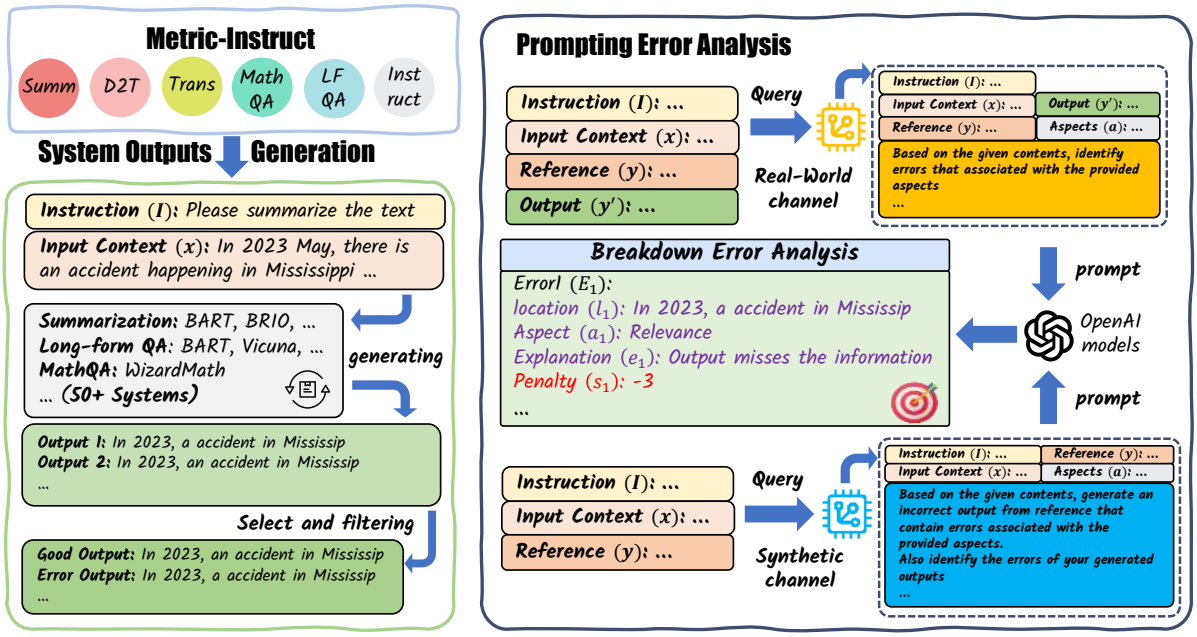


Figure 3.2: Overall pipeline of constructing `MetricInstruct` through the two-channel collection. The main difference is that the real-world channel collects outputs y' from real-world models, while the synthetic channel asks GPT-4 to synthesize an error output based on the provided reference.

evaluation research. Although the assessment of traditional tasks has dominated the research landscape, we posit that effectively evaluating these new tasks is crucial for constructing a comprehensive evaluator for all text generation domains.

In addition, we meticulously selected datasets for each task to ensure diverse coverage across the knowledge domain, as shown in Table 3.2. For instance, in the case of the LF-QA task, we utilize ASQA (Stelmakh et al., 2022) to include knowledge about ambiguous factoid questions, while FeTaQA (Nan et al., 2022) is employed to handle challenges related to tabular source question answering. We then selected samples from each dataset's training set, following the predefined constraints like maximum input and reference output lengths, topics balancing of instances, and so on.

Task	Real-World (training set)			Synthetic (training set)		
	Dataset	Output Source	# Sample	Dataset	Output Source	# Sample
Summarization (Summ)	SummEval‡, XSum, Newsroom,SAMSum	27 Systems	4339	CNN/DM,XSum, Gigaword,SAMSum	GPT-4	612
Translation (Trans)	WMT-22‡	18 Systems	5507	WMT-22	GPT-4	672
Data2Text (D2T)	WebNLG-2020‡ WikiTableText,ToTTo	17 Systems	4701	WikiTableText Dart,ToTTo	GPT-4	160
Long-Form QA (LF-QA)	ASQA,FeTaQA CosmosQA,ELI5	5 Systems	3370	ASQA,FeTaQA Cosmos QA,ELI5	GPT-4	2146
MathQA	GSM8K	5 Systems	4529	None		
Instruction-Following (Instruct)	MixInstruct‡	11 Systems	2248	AlpacaFarm,Dolly Guanaco,OASST	GPT-4	3014

Table 3.2: The composition of our dataset. For synthetic data, the output is generated by asking GPT-4 to synthesize incorrect outputs that contain a few specific types of errors. For the datasets with ‡, we take their pre-released system outputs. For the others, we collect the system outputs on our own.

Broad Coverage of Errors

As shown in Figure 3.2, our "system outputs" come from two channels, namely real-world system outputs and synthetic incorrect outputs, representing 2 main components of `MetricInstruct`.

We consider a wide range of systems and use their outputs as our evaluation input, as shown in Table 3.2. These outputs are either collected from pre-released outputs, like WMT-2021 official evaluation systems, or generated by us by prompting existing domain-specialized models, like BRIO (Liu et al., 2022), WizardMath (Luo et al., 2025), etc.

As illustrated in Figure 3.2, for those self-generated outputs, we sample 5 different outputs using top-p sampling for each instance using various output systems. After that, we use BARTScore to sort the 5 outputs. Outputs with lower BARTScore are heuristically considered to contain more errors and will be preferred to be chosen as the final hypothesis output y' to be evaluated.

Subsequently, we utilized meticulously designed prompting templates (see in B.1.7) to elicit standardized error analysis from GPT-4. The main idea is to provide them with the instruction I, input x, the reference output y, and system output y' along with the definitions of pre-defined aspects A_T, to query the OpenAI models to generate the list of errors, as described in section 3.3.1. We also report their correlation performance in Table 3.4, showcasing the reference-based ChatGPT results.

Synthetic: While real-world system outputs ensure the error distribution is aligned with practical scenarios, they might be overly tailored to the bias of these limited systems and omit certain error cases their outputs fail to cover. Therefore, we decided to synthesize more incorrect outputs that can balance under-represented errors by prompting GPT-4 with our designed templates.

Firstly, to complement the real-world system outputs to cover broader error cases, we prompt GPT-4 to deliberately generate designated erroneous outputs, modified from the existing reference output y, using the prompt template in Table B.10. By supplying GPT-4 with a combination of randomly selected aspects and their definitions A'_T, we control the aspect of errors it produces so that error aspects can be more balanced. Datasets used in this step are reported in the right column of Table 3.2.

Secondly, to improve TIGERScore's generalization capability in evaluation for tasks or errors it has not previously encountered, we employ a strategy that involves prompting GPT-4 to generate data with customized error aspects tailored to individual instructions. Our rationale is that while the manually designed aspects are logical, they may not be specific enough for tasks that involve following instructions where various types of instructions will be included. Besides, this can also prevent the trained model from being overfitted to the pre-defined aspects in subsection B.1.6. To do this, we supplement the first part of synthetic data by sampling 10k data points from the Alpaca-52k dataset and using the template in Table B.11 to prompt GPT-4 to get this second part of synthetic data.

Heurstic-based Filtering

We refine our preliminary raw dataset drawn from both **real-world** and **synthetic** channels using heuristic filtering techniques. Initially, we remove any anomalous data in JSON format. Instances marked by hallucinated or mismatched error locations, illogical severity labels, or excessively long outputs are considered flawed and excluded. Furthermore, given the reference-free nature of TIGERScore, an explanation that relies on reference outputs for justification is excluded. These steps eliminate approximately 35% of the initial data. Subsequently, we employ GPT-4 to assess the reasonableness of our error analysis, utilizing the template outlined in Table B.12. This phase further excludes about 15% of the data. Ultimately, we compile a high-quality dataset of 42,484 instances, designated as `MetricInstruct`.

3.3.3 Training Setup

TIGERScore is finetuned on Llama-2-7B and Llama-2-13B (Touvron et al., 2023) respectively with both's batch size being 128. The prompt templates used for both the fine-tuning and inference are given in Table B.1. The model's maximum context length is set to 1024. We use a cosine learning scheduler with the warmup ratio being 0.1. We finetune the 7B version on 4 A100 GPUs (80GB) for 3 epochs with a learning rate of 2e-5. And 13B version is run on 8 A100 GPUs (80GB) for 2 epochs with a learning rate of 1e-5. Inference of TIGERScore is conducted on a single A100 GPU with the assistance of the VLLM toolkit to increase the speed. (Kwon et al., 2023)

3.4 Experiments

3.4.1 Evaluation Datasets

We have gathered both the held-in and held-out datasets to compare the performance of TIGERScore with the existing baseline metrics. The basic statistics of some main datasets for each task are shown in Table 3.3. System outputs to be evaluated of each test dataset are either from official releases, such as SummEval (Fabbri et al., 2021), WebNLG-2020 (Zhou and Lampouras, 2020), WMT-22 (Freitag et al., 2022), and OpenMEVA (Guan et al., 2021), or by generating by ourselves, such as A-F-E-C (Stelmakh et al., 2022; Nan et al., 2022; Fan et al., 2019; Huang et al., 2019), GSM8K (Cobbe et al., 2021), LIMA (Zhou et al., 2023) and AlpacaEval (Li et al., 2023c).

Human preference scores are necessary to conduct the correlation analysis. To get the gold preference scores we used to conduct the evaluation experiments, we collected either existing human ratings like WMT-22 MQM scores for translation, or GPT-4 rated scores for Long-form QA. Details of the curation are shown in subsection B.1.5.

3.4.2 Baselines

We categorize the baselines into reference-based and reference-free metrics for fair comparison.

Reference-based: We choose popular metrics, including BLEU (Papineni et al., 2002), ROUGE (Lin, 2004), BERTScore (Zhang et al., 2020), BLEURT (Sellam et al., 2020)

and BARTScore (Yuan et al., 2021). Recent emerging metrics are also included, like COMET-22 (Rei et al., 2022a),UniEval (Zhong et al., 2022), GPTScore (Fu et al., 2023), and InstructScore (Xu et al., 2023d). Specifically, we use BARTScore-ref to denote that we adopt the ref-hypo scoring type. For GPTScore, we use FLAN-T5-base (Chung et al., 2024) as base models and use GPTScore-ref to denote that we adopt the f-1 average of the ref-hypo and hypo-ref scores. We also report the zero-shot results of GPT-3.5-turbo with the same prompting templates as those used for generating real-world channel data. Reporting this baseline will help us understand whether TIGERScore has surpassed its easy substitute using the cheap OpenAI model, thus proving its effectiveness.

Reference-free: We choose BARTScore, GPTScore and COMETKiwi (Rei et al., 2022b) as reference-free baselines to compare. Specifically, we use BARTScore-src to denote the src-hypo scoring type, thus making it a reference-free metric. For GPTScore, we still use the FLAN-T5-base model and use GPTScore-src to denote that we use the src-hypo scoring type. We also include the frequently used 0-shot results of Llama-2-13b (Touvron et al., 2023) and GPT-4 (OpenAI, 2023)for better comparison and to know more about the performance gaps with GPT-4.

Task	Eval Dataset	Output Source	# Inputs	# Samples
Held-in Evaluation Dataset (test set)				
Summ	SummEval	16 Systems	100	1600
Trans	WMT-22 (zh-en)	18 Systems	1875	33750
D2T	WebNLG-2020	18 Systems	179	2848
LF-QA	ASQA, FeTaQA, CosmosQA, ELI5	4 Systems	400	1600
Math QA	GSM8K	2 Systems	1319	2638
Held-out Evaluation Dataset (test set)				
Instruct	LIMA,AlpacaEval	9 Systems	500	4500
Story-Gen	OpenMEVA (ROC)	5 Systems	200	1000

Table 3.3: Overview of our main evaluation datasets. "# Samples" is computed by multipling "# Inputs" with the number of outputs systems, representing the total instances that a metric need to compute.

3.4.3 Main Results

We present a comprehensive analysis of TIGERScore across all 5 held-in tasks and 2 held-out task in Table 3.4 reporting Kendall correlation results. Additionally, we provide supplementary results on Pearson and Spearman correlations in the Appendix (see

Tasks→	Summ	Trans	D2T	LF-QA	MathQA	Instruct	Story-Gen	Average
GPT-based Metrics								
GPT-3.5-turbo (zero-shot)	**30.45**	32.30	30.38	20.91	**58.57**	17.73	3.26	27.65
GPT-4 (zero-shot)	29.32	**35.38**	**32.26**	**35.85**	46.63	**49.50**	**25.69**	**36.38**
Reference-based Metrics								
BLEU	8.71	14.50	23.13	7.73	17.25	35.92	-0.89	15.19
ROUGE-2f	10.67	13.19	24.74	11.73	18.07	34.59	1.78	16.40
InstructScore	20.86	40.44	30.21	15.64	-3.87	13.87	13.50	18.66
GPTScore-ref	10.80	18.74	27.47	22.13	14.86	25.40	12.78	18.88
BARTScore-cnn (hypo-ref)	10.00	21.06	27.04	20.67	**19.07**	24.70	18.58	20.16
BARTScore-para (hypo-ref)	10.41	24.90	28.42	20.24	14.10	26.13	12.11	19.47
BERTScore	17.39	31.57	30.74	17.70	9.41	35.61	2.00	20.63
BLEURT	12.69	36.12	**34.48**	23.11	2.88	27.94	19.18	22.34
UniEval (summ)	**35.89**	16.08	28.56	**29.32**	16.15	11.93	**31.22**	24.17
COMET-22	25.01	**42.79**	23.43	24.66	-4.52	**36.17**	27.52	**25.01**
Reference-free Metrics								
BARTScore-para (src-hypo)	29.12	7.01	22.32	18.80	-2.21	4.26	14.15	13.35
BARTScore-cnn (src-hypo)	26.63	9.40	23.69	28.93	1.23	19.09	23.29	18.89
Llama-2-13b-chat-0-shot	25.22	11.79	23.45	15.96	1.08	19.50	21.52	16.93
COMETKiwi	11.87	36.37	19.08	12.23	-9.38	26.46	12.78	15.63
GPTScore-src	28.20	6.50	19.81	27.64	11.64	20.04	16.36	18.60
TIGERScore-7B	28.79	33.65	32.44	33.93	19.98	38.13	29.72	30.95
TIGERScore-13B	**31.29**	**36.50**	**36.43**	**33.17**	**21.58**	**41.84**	**35.33**	**33.73**
Δ (ours - best reference-free)	+2	+0	+13	+4	+10	+15	+14	+15
Δ (ours - best reference-based)	-4	-6	+2	+4	+2	+5	+4	+8

Table 3.4: The Kendall correlation results of all the baseline metrics and TIGERScore on the evaluation datasets shown in Table 3.3. For each task, the metric with the highest correlation to average performance is highlighted in bold. The results of Pearson and Spearman are reported in Table B.15 and Table B.16 respectively, showcasing the same great performance of TIGERScore

Table B.15 and Table B.16). We average the performance across these tasks to gauge the general ability of the model.

Our results highlight the significant advantages of TIGERScore over other reference-free metrics. Notably, it has surpassed all other reference-free metrics in Kendall correlation. In Pearson correlation, it is the highest for 6 out of 7 tasks. This underscores the robustness and consistency of TIGERScore in evaluating text generation tasks.

Compared with reference-based baselines, TIGERScore generally outperforms most reference-based metrics. However, it does score lower than some task-specific metrics like UniEval (summ) in summarization, COMET-22 in translation. We consider these discrep-

ancies acceptable, because the compared metrics are reference-based and all specifically fine-tuned for a single task. Furthermore, TIGERScore achieves significantly higher overall correlation, compared with the cheap API substitute GPT-3.5-Turbo (zero-shot) and the Llama-2-13b-chat (zero-shot), proving its effectiveness. What's exciting to note is that TIGERScore-13b can achieve comparable correlation performance with GPT-4 (zero-shot), even higher on summarization, translation, data2text, and story generation.

3.4.4 Human Evaluation

To better understand the quality of the generated error analysis provided by TIGERScore, a random selection of 50 error analyses from each evaluation dataset was assessed by human experts who rated them from the following perspectives: 1) Reasonableness, 2) Comprehensiveness, 3) Effectiveness, 4) Overall Estimation, whose definitions are given in the following:

Reasonableness: The human experts directly pointed out which errors are problematic in error analyses, examining whether the analysis contained hallucination or illogical reasoning.

Comprehensiveness: The human experts carefully review the source, output, and error analyses to determine if there are any additional errors unnoticed by TIGERScore. Based on human experts' analysis, they give a score on a scale of 1 to 4, specifically focused on identifying potential errors that may have been overlooked in the original analysis conducted by TIGERScore.

Effectiveness: The revision suggestions in error analyses are evaluated by human experts, on a scale of 1 to 5, to determine their appropriateness and effectiveness in enhancing the output quantity.

Overall: The Human experts further assign an overall score on a scale of 1 to 5 based on the reasonableness, comprehensiveness, and effectiveness of the error analysis.

We report the detailed human evaluation results in Table 3.5, it is found that 64.3% of TIGERScore's error analyses are deemed reasonable, that is, the answer to the first question is "no errors in interpretation". This suggests that most error analyses accurately identified and explained errors. In 70.6% of cases, evaluators gave a positive score (3 or 4) for question 2, implying no missing errors were found. This demonstrates TIGERScore's effectiveness in comprehensive error analysis.

Overall, 70.8% of error analyses received positive ratings (3 to 5), indicating good quality and usefulness in identifying and explaining errors according to human experts.

Aspects	Explanation Error?		Overlooked Errors				Revision Suggestions					Overall Rating				
Rate→	No	Yes	1	2	3	4	1	2	3	4	5	1	2	3	4	5
Summ	**70**	35	2	**17**	15	16	6	4	**19**	7	14	3	10	**17**	7	13
Trans	**54**	25	3	8	17	**22**	2	7	17	6	**18**	3	6	15	9	**17**
D2T	19	**21**	1	8	10	**31**	11	8	9	3	**19**	11	10	4	7	**18**
LF-QA	**42**	19	4	10	11	**25**	5	8	14	7	**16**	6	8	10	6	**20**
MathQA	**39**	26	5	12	12	**21**	5	7	**19**	5	14	4	9	10	13	**14**
Instruct	5	**9**	5	5	8	**32**	**21**	3	5	2	19	9	4	3	7	**27**
Story-Gen	**66**	29	7	**16**	13	14	7	6	**16**	10	11	7	**12**	11	9	11
Total	**295**	164	27	76	86	**161**	57	43	99	40	**111**	43	59	70	58	**120**

Table 3.5: Human evaluation results, the first question is asked per error in error analyses, and the others are per sample. Superior performance is indicated by higher numerical values. The most-voted rate of each task for each human evaluation aspect is bolded.

3.4.5 Hallucination Analysis

Alignments on no-error outputs In order to understand the hallucinations generated by TIGERScore, we used TIGERScore to evaluate the gold reference, which is used in reference-based metrics, and expect TIGERScore not to hallucinate errors on these no-error instances. Results are shown in Table 3.6.

In tasks related to instruction-following and long-form QA, TIGERScore demonstrates a high level of accuracy, avoiding hallucinations in over 85% of cases. This highlights its proficiency in producing factual and error-free analysis. However, TIGERScore is less consistent in tasks like summarization, translation, and data-to-text conversion. In these areas, it often fails to achieve perfect scores (0), but still frequently identifies gold references as either flawless or only minimally flawed (with score reductions less than 2). This could be due to the subjective nature of these tasks, where minor errors, such as the substitution of similar words, maybe more open to interpretation. Additionally, TIGERScore faces challenges in tasks like MathQA and story generation. These difficulties may stem from the inherent complexity of MathQA problems and the subjective nature of story creation, as well as specific limitations of TIGERScore in these areas. Improving TIGERScore's performance in these challenging tasks remains a topic for future research.

Hallucinations analysis of TIGERScore outputs To better understand how TIGERScore handles hallucinations, we conducted experiments across seven different tasks. For each task, we ran TIGERScore-13B on 20 samples with errors in the system output. We then used GPT-4 to determine if these samples contained hallucinations or factual inac-

curacies, as outlined in the prompting templates found in Table B.14. According to the results in Table 3.7, approximately 89.28% of TIGERScore's error analyses are free from hallucinations or factual errors. We acknowledge the limitations of our study, including the small sample size and the reliance on GPT-4 rather than human evaluators. Nonetheless, our findings are significant, demonstrating that TIGERScore is effective at avoiding hallucinations in generated content.

Tasks→	Summ	Trans	D2T	LF-QA	MathQA	Instruct	Story-Gen	Average
Gold reference's score = 0								
TIGERScore-7B	16.00	3.57	45.51	84.75	34.36	73.98	34.00	41.74
TIGERScore-13B	48.00	21.01	23.03	94.50	25.28	86.38	46.00	49.17
Gold reference's score >-2								
TIGERScore-7B	68.00	72.48	78.65	84.75	34.36	92.48	34.00	66.39
TIGERScore-13B	97.00	83.63	94.94	94.50	25.28	96.14	49.00	77.21

Table 3.6: TIGERScore's score on the gold reference of the test set. For each task, the 0 column refers to the percentage that TIGERScore reduces 0 scores for the gold references. The > -2 column refers to the percentage that TIGERScore deducts less than 2 points from the gold references.

Tasks	Summ	Trans	D2T	LF-QA	MathQA	Instruct	Story-Gen	Total
Accuracy	95.00	95.00	90.00	85.00	90.00	75.00	95.00	89.28

Table 3.7: The accuracy of the error analysis from TIGERScore-13B assessed by GPT-4 that do not contain hallucinations or factual errors.

3.4.6 Ablation Study on Data Source

Ablation of Two Channel Data Collection To assess the effectiveness of our two-channel data in enhancing TIGERScore's performance, we fine-tune 3 models using different dataset setups for experiments: using only Real-World data (**ReW**), using only Synthetic data (**Syn**) and using the mixed of both (**Mix**). The results are detailed in the Spearman correlation in Figure 3.3. the **Mix** model outperformed both **ReW** and **Syn** in 5 of the 7 tasks. It also achieves an average correlation approximately 20% higher than **ReW** and 31% higher than **Syn**. Slight decreases in correlation for D2T and Story-Gen tasks are deemed as acceptable compromises for better overall performance.

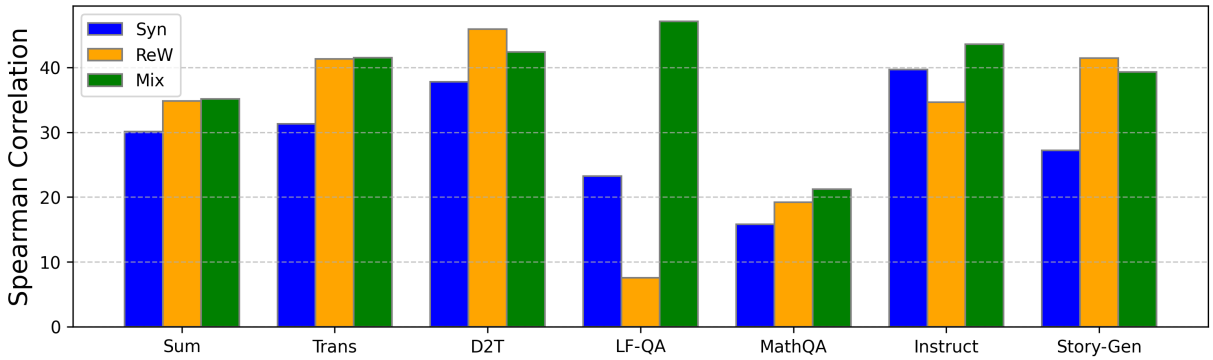


Figure 3.3: Investigation of the influence of Real-World & Synthesic mix training on the 7B model.

Ablation of Each Single Generation Task In order to investigate the contribution of each task in the `MetricInstruct`, we conducted experiments to see whether multi-task training will benefit the single task. We trained a model for each task and compared its performance to that of TIGERScore. As depicted in the Table 3.8, the multi-task learning paradigm does benefit almost all tasks' performance, except for MathQA. We contend that Math QA poses a significant challenge for LLMs, and a separately trained model is more adept at handling this task.

Metrics↓ Tasks→	Summ	Trans	D2T	LF-QA	MathQA	Instruct	Average
Single Task	35.55	41.65	39.75	41.60	**26.75**	41.19	37.75
Multi Task	**36.81**	**44.99**	**45.88**	**46.22**	23.32	**47.03**	**40.71**
Δ (%)	3.56	8.02	15.43	11.09	-12.83	14.18	7.84

Table 3.8: Ablation of the influence of multiple tasks mix or single task on the 13B model.

3.5 Analysis

The experiments above evaluate TIGERScore on outputs fixed in advance. I now examine whether its scores can also help select a better response from several candidates or serve as a reward for training the generator.

3.5.1 Best-of-N Selection

Best-of-N selection samples several responses to the same prompt and uses an evaluator to choose one, without changing the generator. I sample eight responses from Qwen2.5-7B-Instruct at temperature 0.7 for each AlpacaEval and GSM8K prompt, then score them with TIGERScore-7B. I select the response with the smallest total penalty, breaking ties at random. I compare this choice with a random response from the same eight candidates. On GSM8K, I also compare with simple length-based rules and an oracle that selects a correct response whenever one is available.

Table 3.9 shows that TIGERScore selection reaches 92.2% GSM8K accuracy, compared with 91.3% for random selection and 97.0% for the oracle. On a subset of 300 AlpacaEval prompts, an independent judge, an OpenAI model (gpt-6-luna), prefers the selected response to a random candidate in 52.3% of comparisons (standard error 2.9 percentage points). The limited gain is consistent with the scores providing little distinction among candidates. Of the AlpacaEval responses, 94% receive no penalty, while on GSM8K both many correct and many incorrect responses receive none. The evaluator's agreement with human ratings across different systems therefore does not necessarily translate into a reliable ranking of one model's responses to the same prompt.

Selection rule	GSM8K accuracy (%)
Random sample	91.3
Longest sample	89.5
Shortest sample	92.0
TIGERScore (smallest penalty)	92.2
Best of the eight (oracle)	97.0

Table 3.9: GSM8K accuracy when selecting one of eight responses from Qwen2.5-7B-Instruct. All rules use the same candidates; the oracle selects a correct response whenever one is available.

3.5.2 Training with the Evaluator

I next use the TIGERScore-7B score as the reward for training Qwen2.5-1.5B-Instruct with GRPO (Shao et al., 2024). The policy is trained for 200 steps on a mixture of instruction-following prompts from UltraFeedback (Cui et al., 2024) and GSM8K training problems. A response receives the evaluator's final score, so fewer detected errors give a

higher reward and no detected errors give the maximum score of zero. Separately, I track GSM8K test accuracy and the same judge's preference between the trained policy and its initial version on a fixed subset of AlpacaEval prompts.

Figure 3.4 shows that improving this reward does not improve the independent evaluations. The mean penalty on AlpacaEval responses approaches zero, while the judge increasingly prefers the initial policy and GSM8K accuracy falls from 70.9% to 18.7%. The policy increasingly produces refusals or disclaimers, including on answerable mathematics problems. These responses offer few substantive claims for the evaluator to mark as wrong, allowing the policy to avoid deductions without completing the task. A second run with a different data order reproduces this pattern.

These small-scale experiments illustrate two different requirements for using an evaluator beyond scoring fixed outputs. Selection needs scores that distinguish nearby candidates, and training needs a reward that continues to reflect task performance as the policy changes. Part II takes up these uses of feedback through program execution and interaction with environments.

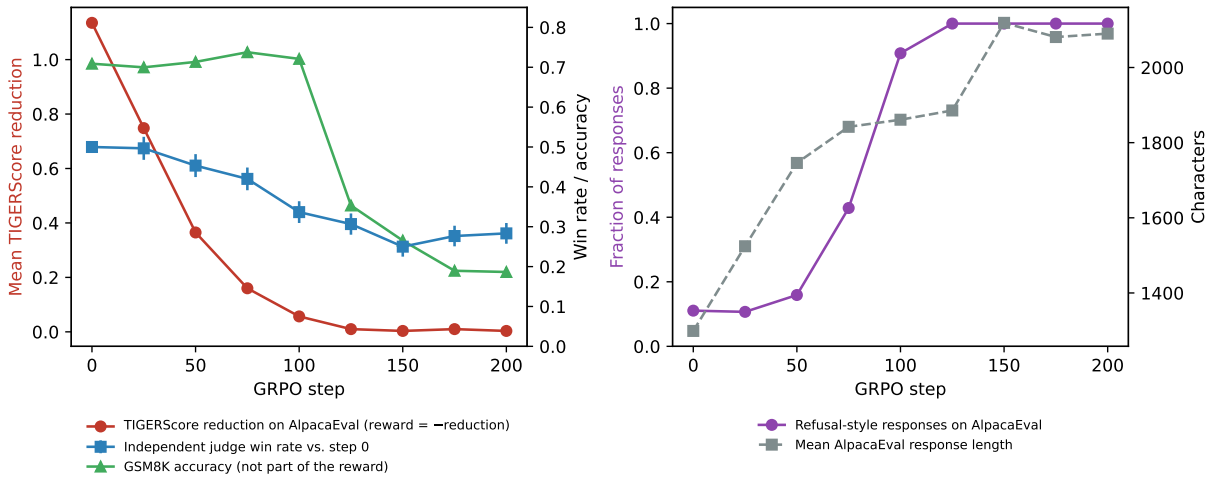


Figure 3.4: Optimizing a policy against TIGERScore (first of two runs). Left: the TIGERScore penalty on AlpacaEval responses falls toward zero, while an independent judge prefers the step-0 model increasingly often (bars: one standard error) and GSM8K accuracy, unseen by the reward, collapses after step 100. Right: the policy learns to refuse or disclaim, and its responses grow longer.

3.6 Summary

This chapter trained Llama-2 on filtered GPT-4 error analyses to obtain TIGERScore, a reference-free evaluator that locates and explains errors in a response. Its average Kendall correlation with human ratings on seven tasks, 33.73, is above every compared metric except zero-shot GPT-4 (36.38), and human raters scored 70.9% of its error analyses 3 or higher out of 5.

Two limits are visible in the chapter itself. The evaluator learns from GPT-4's analyses and inherits GPT-4's errors, and it is weakest on MathQA, where judging a solution requires solving it. Section 3.5 then tested the score beyond fixed outputs, as a selector and as a reward. As a Best-of-8 selector it barely separated one policy's samples (92.2% against 91.3% on GSM8K), and a policy trained against it with GRPO drove its penalties to near zero by learning to refuse, while an independent judge and GSM8K accuracy fell (two runs, 1.5B policy).

The approach rested on GPT-4 as a strong approximate judge to distill from, and on existing human ratings to check the result against. For images and videos there was no comparably strong judge to distill from, so Chapter 4 turns to human judgment itself.

Chapter 4

Human Feedback for Generated Images and Videos

4.1 Introduction

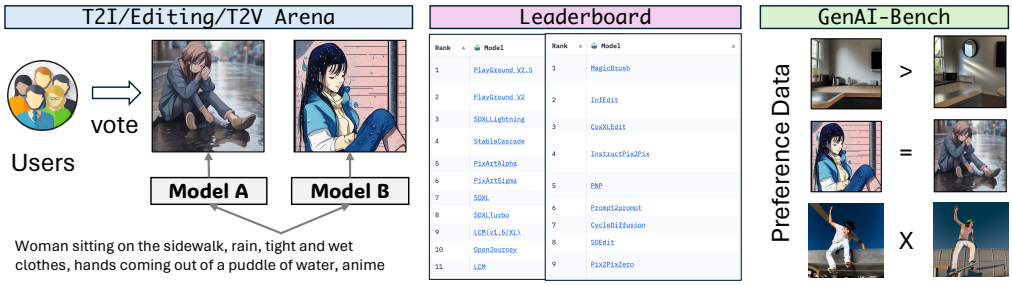


Figure 4.1: GenAI Arena contains three components: (1) text-to-image, text-to-video and image editing arena, which accept community voting to obtain the preference pairs. (2) The leaderboard utilizes the preference pairs to calculate elo ranking for all the evaluated models. (3) We further release GenAI-Bench to judge different multimodal LLM judges.

This chapter studies the second source of feedback, human judgment, for generated images and videos, where no evaluator as strong as GPT-4 was available to learn from. Despite the rapid progress of image and video generation, navigating through the multitude of available models and assessing their performance remains a challenging task Ramesh et al. (2022). Traditional evaluation metrics like PSNR, SSIM Wang et al. (2004), LPIPS Zhang et al. (2018), and FID Heusel et al. (2017), while valuable, offer very specific insights

into precise aspects of visual content generation. However, these metrics often fall short in providing a comprehensive assessment of overall model performance, especially when considering subjective qualities like aesthetics and user satisfaction Otani et al. (2023).

To address these challenges, we introduce GenAI-Arena, a platform designed to enable fair evaluation. Inspired by successful implementations in other domains Zheng et al. (2023a), GenAI-Arena offers a dynamic and interactive platform where users can generate images, compare them side-by-side, and vote for their preferred models. Such a platform not only simplifies the process of comparing different models but also provides a ranking system that reflects human preferences, thereby offering a more holistic evaluation of model capabilities. At the time of its release in early 2024, GenAI-Arena was the first evaluation platform with comprehensive evaluation capabilities across multiple properties. Unlike other platforms, it supports a wide range of tasks across text-to-image generation, text-guided image editing, and text-to-video generation, along with a public voting process to ensure labeling transparency. The votes are utilized to access the evaluation ability of Multimodal Large Language Model (MLLM) evaluators. Table 4.1 shows our platform excels in its versatility and transparency.

Since February 11th, 2024, we have collected over 9000 votes for three multimodal generative tasks. We constructed leaderboards for each task with these votes, identifying the state-of-the-art models as PlayGround V2.5, MagicBrush, and StableVideoDiffusion, respectively (until Oct 24th, 2024). Detailed analyses based on the votes are presented. For example, our plotted winning fraction heatmaps reveal that while the Elo rating system is generally effective, it can be biased by imbalances between "easy games" and "hard games". We also performed several case studies for qualitative analysis, demonstrating that users can provide preference votes from multiple evaluation aspects, which help distinguish subtle differences between the outputs and upload high-quality votes for Elo rating computation.

Automatically assessing the quality of generated visual content is a challenging problem for several reasons: (1) images and videos have many different aspects like visual quality, consistency, alignment, artifacts, etc. Such a multi-faceted nature makes the evaluation intrinsically difficult. (2) the supervised data is relatively scarce on the web. In our work, we release the user voting data as GenAI-Bench to enable further development in this field. Specifically, we calculate the accuracy between different image/video auto-raters (i.e. MLLM judges like GPT-4o, Gemini, etc.) with user preference to understand their judging abilities. Our results show that even the best MLLM, GPT-4o achieves at most 49.19% accuracy compared with human preference.

Table 4.1: Comparison with different evaluation platforms on different properties.

Platform	Text-To-Image Generation	Text-Guided Image Editing	Text-To-Video Generation	Human Label Transparency	Open/Public Voting Process	Judging MLLM judge
T2I-CompBench Huang et al. (2023)	✓	✗	✗	✗	✗	✗
HEIM Lee et al. (2023)	✓	✗	✗	✓	✗	✗
ImagenHub Ku et al. (2024b)	✓	✓	✗	✓	✗	✗
VBench Huang et al. (2024b)	✗	✗	✓	✓	✗	✗
EvalCrafter Liu et al. (2024d)	✗	✗	✓	✓	✗	✗
GenAI-Arena	✓	✓	✓	✓	✓	✓

4.2 Related Work

4.2.1 Generative AI Evaluation Metrics

Section 2.1 covers the automatic measures for images and videos in general. This subsection records the metrics the platform was designed against. Numerous methods have been proposed to evaluate the performance of multi-modal generative models in various aspects. In the context of image generation, CLIPScore Hessel et al. (2021) is proposed to measure the text-alignment of an image and a text through computing the cosine similarity of the two embeddings from CLIP Radford et al. (2021). IS Salimans et al. (2016) and FID Heusel et al. (2017) measure image fidelity by computing a distance function between real and synthesized data distributions. PSNR, SSIM Wang et al. (2004) assess the image similarity. LPIPS Zhang et al. (2018) and the follow-up works Fu et al. (2024); Ghazanfari et al. (2023) measure the perceptual similarity of images. More recent works leverage the Multimodal Large Language Model (MLLM) as a judge. T2I-CompBench Huang et al. (2023) proposed the use of miniGPT4 Zhu et al. (2023) to evaluate compositional text-to-image generation task. TIFA Hu et al. (2023) further adapted visual question answering to compute scores for the text-to-image generation task. VIEScore Ku et al. (2024a) leveraged MLLMs as a unified metric across image generation and editing tasks, reporting that MLLM has great potential in replacing human judges.

Metrics in similar fashions are also proposed for the video domain. For example, FVD Unterthiner et al. (2018) measures the coherence shifts and quality in frames. CLIP-SIM Radford et al. (2021) utilizes an image-text similarity model to assess the similarity between video frames and text. VBench Huang et al. (2024b) and EvalCrafter Liu et al. (2024d) also proposed different metrics for evaluating different aspects of the video generation task. However, these automatic metrics still lag compared with human preferences, achieving low correlation and thus giving doubts to their reliability.

4.2.2 Generative AI Evaluation Platforms

Arenas as a form of human evaluation are introduced in Section 2.1. The platforms below are the ones that existed for generative models at the time.

While auto-metric focuses on evaluating a single model's performance, evaluation platforms aim to systematically rank a group of models. Recently, several benchmark suites have been developed to comprehensively assess generative AI models. For image generation, T2ICompBench Huang et al. (2023) evaluates compositional text-to-image generation tasks, while HEIM Lee et al. (2023) offers a holistic evaluation framework that measures text-to-image tasks across multiple dimensions, including safety and toxicity. Similarly, ImagenHub Ku et al. (2024b) evaluates text-to-image, image editing, and other prevalent image generation tasks in a unified benchmark suite. For video generation, VBench Huang et al. (2024b) and EvalCrafter Liu et al. (2024d) provide structured evaluation approaches ensuring rigorous assessment. Despite their functionality, these benchmarks rely on model-based evaluation metrics, which are less reliable than human evaluation.

To address this issue, variable model arenas have been developed to collect direct human preferences for ranking models. Chatbot Arena by LMsys Chiang et al. (2024) is the pioneering platform in this regard, setting the standard for evaluation. Subsequent efforts have led to the creation of arenas for vision-language models Xu et al. (2025), TTS models Hugging Face Spaces (2024b), and tokenizers Hugging Face Spaces (2024a). However, there is no existing arena for generative AI models. To fill this gap, we propose GenAI-Arena as a complementary solution in this field.

4.3 GenAI-Arena: Design and Implementation

4.3.1 Design

GenAI-Arena is designed to offer an intuitive and comprehensive evaluation platform for generative models, facilitating user interaction and participation. The platform is structured around three primary tasks: text-to-image generation, image edition, and text-to-video generation. Each task is supported by a set of features that include an anonymous and a non-anonyumous battle playground, a direct generation tab, and a leaderboard as shown in Figure 4.2 . These features are designed to cater to both casual users and researchers, ensuring a democratic and accurate assessment of model performance.

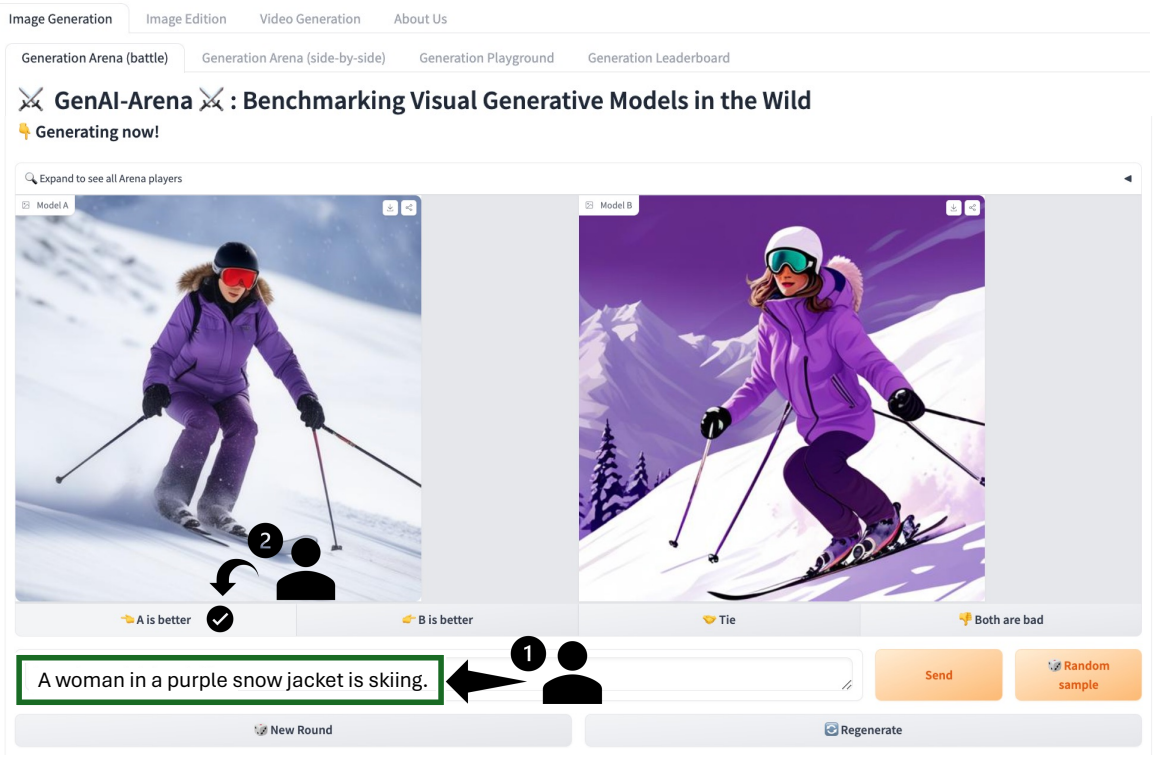


Figure 4.2: GenAI Arena User Voting Interface.

Standardized Inference To ensure a fair comparison between different models, we ported the highly dispersed codebase from the existing works and then standardized them into a unified format. During inference, we fixed the hyper-parameters and the prompt format to prevent per-instance prompt or hyper-parameter tuning, which makes the inference of different models fair and reproducible. Following ImagenHub Ku et al. (2024b), we build the new library of VideoGenHub (details in subsection B.2.2), which aims to standardize the inference procedure for different text-to-video and image-to-video models. We find the best hyper-parameters of these models to ensure their highest performance.

Voting Rules The anonymous battle section is designed to ensure unbiased voting and accurate evaluation of generative models. The rules for this section are as follows:

1. Users input a prompt, which is then used to generate outputs from two anonymous models within the same category of task.

2. The generated outputs from the two anonymous models are presented side-by-side for comparison.
3. Users can vote based on their preference using the options: 1) left is better; 2) right is better; 3) tie; 4) both are bad. These four options are being used to calculate Elo ranking.
4. Once the user has made their decision, they click the Vote button to submit their vote. It is important to ensure that the identity of the models remains anonymous throughout the process. Votes will not be counted if the model identity is revealed during the interaction.

4.3.2 Model Integration

In GenAI-Arena, we incorporate a diverse array of state-of-the-art generative models, covering a broad range of generative tasks including text-to-image generation, image edition, and text-to-video generation. To ensure comprehensive evaluations, the platform includes models that employ diverse underlying technologies, such as different types of architectures, training paradigms, training data and acceleration techniques. These variations can offer insights to understand these factors rigorously.

Text-to-Image Generation In Table 4.2, we list all the included text-to-image generation models. For example, SDXL, SDXL-Turbo, and SDXL-Lightning are all derived based on SDXL Podell et al. (2024), while SDXL-Turbo Sauer et al. (2024) and SDXL-Lightning Lin et al. (2024b) adopt different distillation method. We also include diffusion transformer models Peebles and Xie (2023) like PixArt-α and PixArt-σ. Playground V2 and Playground V2.5 are based on SDXL architecture, but trained by Playground.ai from scratch with an internal dataset. We have also included the latest released HunyuanDiT Li et al. (2024d), FLUX.1-dev Black Forest Labs (2024), FLUX.1-schnell Black Forest Labs (2024).

Text-guided Image Editing In Table 4.3, we list all the image editing models and approaches. Some of them are plug-and-play approaches without requiring any training, like Pix2PixZero Parmar et al. (2023), InfEdit Xu et al. (2024), SDEdit Meng et al. (2022), etc. These methods can be applied to a broad range of diffusion models. Some of the models like PnP Tumanyan et al. (2023) and Prompt2Prompt Hertz et al. (2023) require DDIM inversion, which takes much longer time than the other approaches. We also include specialized

Table 4.2: The overview of all text-to-image generation models.

Model	Size	Method	Resolution	#Steps
OpenJourney openjourney.ai (2023)	1B	SD-2.1 + MidJourney Dataset	512x512	50
LCM Luo et al. (2023a)	1B	SD-2.1 + Consistency Distillation	512x512	4
SDXL Podell et al. (2024)	3.5B	Latent Diffusion Model (LDM)	1K×1K	50
SDXL-Turbo Sauer et al. (2024)	3.5B	LDM + Distillation	1K×1K	1
SDXL-Lightning Lin et al. (2024b)	3.5B	LDM + Distillation	1K×1K	4
PixArt-α Chen et al. (2024c)	0.6B	Diffusion Transformer (DiT)	1K×1K	50
PixArt-σ Chen et al. (2024b)	0.6B	DiT + Weak-to-Strong	4K×4K	50
StableCascade Pernías et al. (2024)	1.5B + 3.6B	Würstchen	1K×1K	20+10
Playground V2 Li et al. (2024b)	3.5B	LDM	1K×1K	50
Playground V2.5 Li et al. (2024a)	3.5B	LDM	1K×1K	50
FLUX.1-dev Black Forest Labs (2024)	12B	Guidance-distilled DiT + Flow Matching	1K×1K	20
FLUX.1-schnell Black Forest Labs (2024)	12B	Timestep-distilled DiT + Flow Matching	1K×1K	4
Kolors Kwai-Kolors (2024)	2.6B	LDM + ChatGLM3	1K×1K	50
HunyuanDiT Li et al. (2024d)	1.5B	DiT + multilingual text encoder	1K×1K	50
Stable Diffusion 3 Stability AI (2024b)	8B	Multimodal DiT	1K×1K	50
AuraFlow fal.ai (2024)	6.8B	Flow-based Model	1K×1K	50

Table 4.3: Overview of all the image editing models.

Model	Trained?	Method	Runtime
Pix2PixZero Parmar et al. (2023)	Zero-shot	Editing Direction Discovery + Attention Control	21s
SDEdit Meng et al. (2022)	Zero-shot	Iteratively Denoising through SDE	13s
CycleDiffusion Wu and De la Torre (2023)	Zero-shot	Reconstructable Encoder for Stochastic DPMs	9s
Prompt2Prompt Hertz et al. (2023)	Zero-shot	Prompt-based Cross-attention Control	120s
PnP Tumanyan et al. (2023)	Zero-shot	Feature and Self-attention Injection	120s
InfEdit Xu et al. (2024)	Zero-shot	Consistent Model + Uni-Attention Control	5s
InstructPix2Pix Brooks et al. (2023)	Trained	Instruction-based Fine-tuning with Synthetic Data	12s
MagicBrush Zhang et al. (2023a)	Trained	Instruction-based Fine-tuning with Annotated Data	12s
CosXLEdit Stability AI (2024a)	Trained	Cosine-Continuous EDM VPred schedule	50s

Table 4.4: Overview of all text-to-video generation models.

Model	Base	Len	FPS	Dataset	Resolution	#Steps
AnimateDiff Guo et al. (2024b)	SD-1.5	2s	8	WebVid10M	512×512	25
AnimateDiff-Turbo Guo et al. (2024b)	SD-1.5	2s	8	WebVid10M	512×512	4
ModelScope Wang et al. (2023a)	SD-1.5	2s	8	WebVid10M	256×256	50
LaVie Wang et al. (2025d)	SD-1.5	2s	8	Vimeo25M	320×512	50
StableVideoDiffusion Blattmann et al. (2023)	SD-2.1	2.5s	10	LVD-500M	576×1024	20
VideoCrafter2 Chen et al. (2024a)	SD-2.1	2s	16	WebVid10M	320×512	50
T2V-Turbo Li et al. (2024c)	VideoCrafter2	2s	8	WebVid10M	320×512	4
OpenSora Zheng et al. (2024b)	Pixart-α	2s	16	WebVid10M	320×512	50
OpenSora v1.2 Zheng et al. (2024b)	Pixart-α	2s	16	WebVid10M	320×512	50
CogVideoX-2B Yang et al. (2025)	DiT	2s	8	35M videos + 2B images	480×720	50

trained image editing models like InstructP2P Brooks et al. (2023), MagicBrush Zhang et al. (2023a) and CosXLEdit Stability AI (2024a).

Text-to-Video Generation In Table 4.4, we list all the text-to-video generation models. We include different types of models. For example, AnimateDiff Guo et al. (2024b), ModelScope Wang et al. (2023a), Lavie Wang et al. (2025d) are initialized from SD-1.5 and continue trained by injecting a motion layer to capture the temporal relation between frames. In contrast, StableVideoDiffusion Blattmann et al. (2023) and VideoCrafter2 Chen et al. (2024a) are iniialized from SD-2.1. Besides these models, we also include OpenSora Zheng et al. (2024b), which utilizes a Sora-like diffusion transformer Peebles and Xie (2023) architecture for joint space-time attention.

4.3.3 Elo Rating System

Section 2.1 introduces the Bradley–Terry model, the Elo probability model, and the online Elo update. Following Chatbot Arena Chiang et al. (2024), we use a Bradley–Terry fit on the Elo scale rather than online Elo updates, so that the ratings do not depend on the order in which votes arrive.

Bradley–Terry Model Estimation Since the Bradley–Terry (BT) model does not account for ties, we first duplicate all the votes, then allocate half of the "tie" votes to the scenario where model i wins and the other half to the scenario where model j wins in practice. We fit the model using `LogisticRegression` from `sklearn`.

Confidence Interval To further investigate the variance of the estimated Elo rating, we use the "sandwich" standard errors described in Huber (1967). That is, for each round, we record the estimated Elo rating based on the same number of battles sampled from the previous round. This process continues for 100 rounds. We select the lowest sampled elo rating as the lower bound of the confidence interval, and the highest sampled elo rating as the upper bound of the elo rating.

Selection of battle pair With a limited number of games, choosing which two players to match up is a crucial issue. The simplest approach, which we currently use, is to randomly select two players. However, this can introduce bias, with some models getting significantly more matches than others. A vote-aware selection system that increases the probability of

selecting less-played models and lowers it for more-played ones is needed, and we plan to explore this in future Arena improvements.

4.3.4 GenAI-Museum

Current GenAI-Arena runs the model on the Hugging Face Zero GPU system Hugging Face (2024). As shown in Table 4.3, the time for a single generative inference usually ranges from 5 to 120 seconds. Unlike the auto-regression language model, where inference acceleration techniques like VLLM Kwon et al. (2023), SGLang Zheng et al. (2023b) generate responses in less than a second, diffusion model community does not have such powerful infrastructure. Therefore, pre-computation becomes a necessary way to mitigate computational overhead and streamline user interaction.

To achieve this, we serve GenAI-Museum as a pre-computed data pool comprising various inputs from existing datasets or user collection, along with each model's output. Based on this, a `"Random Sample"` button shown in Figure 4.2 is additionally implemented to facilitate the random generation of prompts and the immediate retrieval of corresponding images or videos. This functionality operates by sending requests to our deployed GenAI-Museum every time `"Random Sample"` button is hit, receiving input and two random model's pre-computed outputs. In this way, we save the computation time on the GPU, enable users to do instant comparisons and votes on the UI, and balance the votes for each unique input so we gradually collect votes for a full combination of all models. The input prompts were sampled from ImagenHub Ku et al. (2024b) and VBench Huang et al. (2024b). To prevent the bias in the prompt distribution, we also periodically update the input prompts with the lastest collected real-world human votes. We make sure every prompt is filtered via NSFW detector before adding them.

4.4 Benchmarks and Results Discussion

4.4.1 Arena Leaderboard

We report our leaderboard at the time of paper publishing in Table 4.5. For image generation, we collected 6300 votes in total. The currently top-1 model is Playground V2.5, released by Playground.ai, which follows the same architecture as SDXL but is trained with a private dataset. In contrast, SDXL only ranks in the thirteenth position, lagging

significantly behind. Such finding highlights the importance of the training dataset. StableCascade is ranked in the sixth place in the leaderboard, which utilizes a highly efficient cascade architecture to lower the training cost. According to Würstchen Pernías et al. (2024), StableCascade only requires a 10% training cost of SD-2.1, yet it can beat SDXL significantly on our leaderboard. This highlights the importance of the diffusion architecture to achieve strong performance. For image editing, a total of 1154 votes have been collected. MagicBrush, InFEdit, CosXLEdit, and InstructPix2Pix ranked higher as they can perform localized editing on images. PNP preserves the structure with feature injections, thus limiting the edit variety. The older methods such as Prompt-to-Prompt, CycleDiffusion, SDEdit, and Pix2PixZero, frequently result in completely different images during editing despite the high-quality images, which explains the lower ranking of these models. For text-to-video, there is a total of 2024 votes. StableVideoDiffusion leads with the highest Elo score, suggesting it is the most effective model. Close behind, CogVideoX-2B ranks second. The following VideoCrafter2 and AnimateDiff have very close elo scores, showing nearly equivalent capabilities. LaVie, OpenSora, ModelScope, and AnimateDiff-Turbo follow with decreasing scores, indicating progressively lower performance.

Table 4.5: GenAI-Arena Leaderboards.
(Last updated on Oct 24th, 2024)

(a) Text-to-Image (Top-10)

Model	Elo	95% CI
PlayGround V2.5	1122	+19/-20
FLUX.1-dev	1114	+45/-42
FLUX.1-schnell	1085	+43/-46
Playground V2	1072	+18/-22
Kolors	1069	+32/-39
StableCascade	1057	+17/-21
HunyuanDiT	1030	+25/-27
PixArt-α	1020	+17/-19
SDXL-Lightning	1020	+17/-15
PixArt-σ	1019	+22/-20

(b) Image Editing

Model	Elo	95% CI
MagicBrush	1108	+32/-28
InfEdit	1075	+26/-32
CosXLEdit	1066	+31/-29
InstructPix2Pix	1038	+32/-24
PNP	998	+35/-34
Prompt2prompt	988	+26/-23
CycleDiffusion	943	+26/-26
SDEdit	924	+24/-23
Pix2PixZero	858	+25/-30

(c) Text-to-Video

Model	Elo	95% CI
StableVideoDiffusion	1148	+31/-28
CogVideoX-2B	1106	+71/-71
T2V-Turbo	1085	+36/-32
VideoCrafter2	1068	+20/-22
AnimateDiff	1068	+25/-21
LaVie	996	+25/-21
OpenSora	912	+23/-23
OpenSora v1.2	894	+54/-71
ModelScope	862	+25/-22
AnimateDiff-Turbo	861	+22/-20

4.4.2 Discussion and Insights

Winning Fraction and Elo Rating We visualize the winning fraction heatmap in Figure 4.3, where each cell represents the actual winning fraction of Model A over Model B. The models are ordered by their Elo rating in the heatmap. Horizontally across each row, the winning fraction of Model A increases as the Elo rating of Model B decreases, demonstrating the effectiveness of the Elo rating system in ranking different models.

(a) Text-to-Image (b) Image Editing (c) Text-to-Video

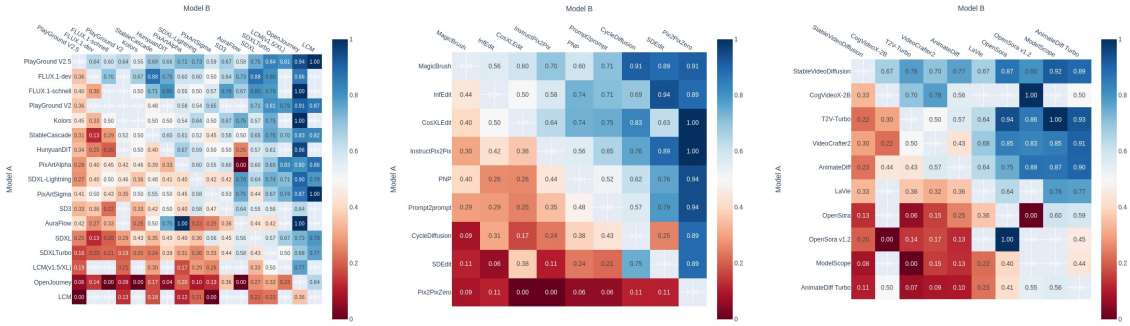


Figure 4.3: Winning fraction heatmap of different models for the three tasks in GenAI-Arena

(a) Text-to-Image (b) Image Editing (c) Text-to-Video

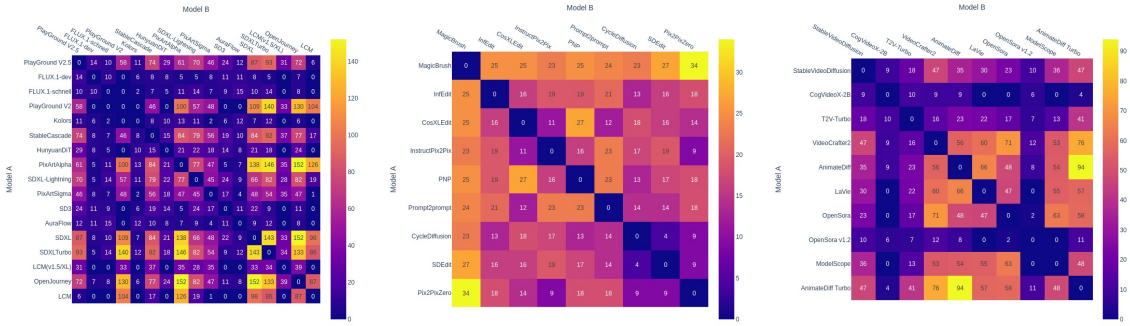


Figure 4.4: Battle count heatmap of different models for the three tasks in GenAI-Arena (without Ties)

(a) Text-to-Image (b) Image Editing (c) Text-to-Video

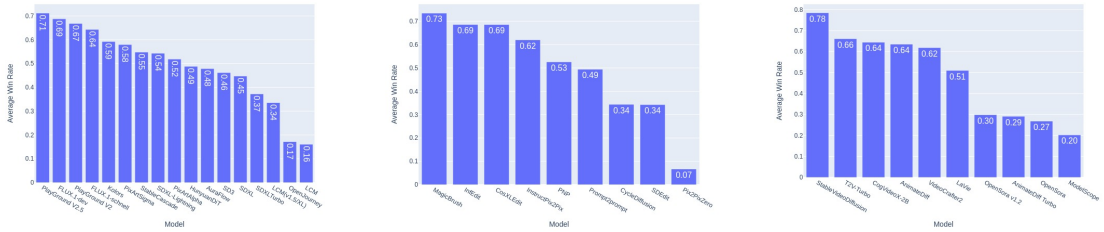


Figure 4.5: Average Win Rate Against All Other Models (Assuming Uniform Sampling and No Ties)

Specific cells in the heatmap reveal notable findings. For instance, although PlayGround 2.5 achieves the state-of-the-art (SOTA) Elo rating in the Text-to-Image task, its winning fraction over PixArt-σ is only 0.58, which is below 60%. The higher Elo rating of T2V-Turbo might be due to our Arena collecting more votes from "easy games" with low-ranked models and fewer from "harder games" with high-ranked models. For example, the number of battles between PlayGround V2.5 and SDXL-Turbo (93) is way more than PlayGround V2.5 with other models (around 50) in Figure 4.4.

These anomalies highlight potential drawbacks of the Elo rating system: (1) a reliable and robust Elo rating requires a large amount of voting data, and (2) the estimated Elo rating may be biased by the imbalance between "easy games" and "harder games," as they carry similar weight in the estimation.

As shown in Figure 4.5, we observe that the average win rates of the top-ranked models are all quite similar, none exceeding 80%. This indicates that there is no dominant, highly powerful model in text-to-image, image editing, or text-to-video generation at this time. The community is still awaiting a "ChatGPT moment"—the release of a breakthrough model with transformative capabilities.

Quality assessment of collected human votes Since our arena users come from different backgrounds and have different preferences, we conduct an expert review on a small set of sampled human vote to ensure there are no severe quality issues of our collected votes. We let different authors review 50 items for each set. A total of 350 items from our GenAI-Bench are evaluated. During the annotations, we skipped those bad items due to NSFW or technical issues, and we finally collected 303 valid evaluations. For each vote, 3 available labels for provided for annotating:

- Clearly Reasonable Vote: This vote will be clearly agreed by most of the people.
- Vague Vote: The current vote makes sense. But it's also reasonable if other vote is selected.
- Wrong Vote: This vote will be clearly disagreed by most of the people.

We report the distribution of valid votes in Table 4.6a, and find that 86.57% of the votes are valid without NSFW issues. Among these valid votes, about 76.24% of the votes are clearly reasonable votes and 93.07% of the votes are either clearly reasonable or vaguely reasonable, as shown in Table 4.6b. We believe this shows the reliability of our preference data.

Table 4.6: Expert Review for 350 sampled human votes

(a) Distribution of Valid Votes

# Valid votes	# NSFW	# Tech issue	Total
303	17	30	350
86.57%	4.86%	8.57%	100%

(b) Distribution of quality labels

# Clearly Reasonable Vote	# Vague Vote	# Wrong Vote	Total
231	51	21	303
76.24%	16.83%	6.93%	100%

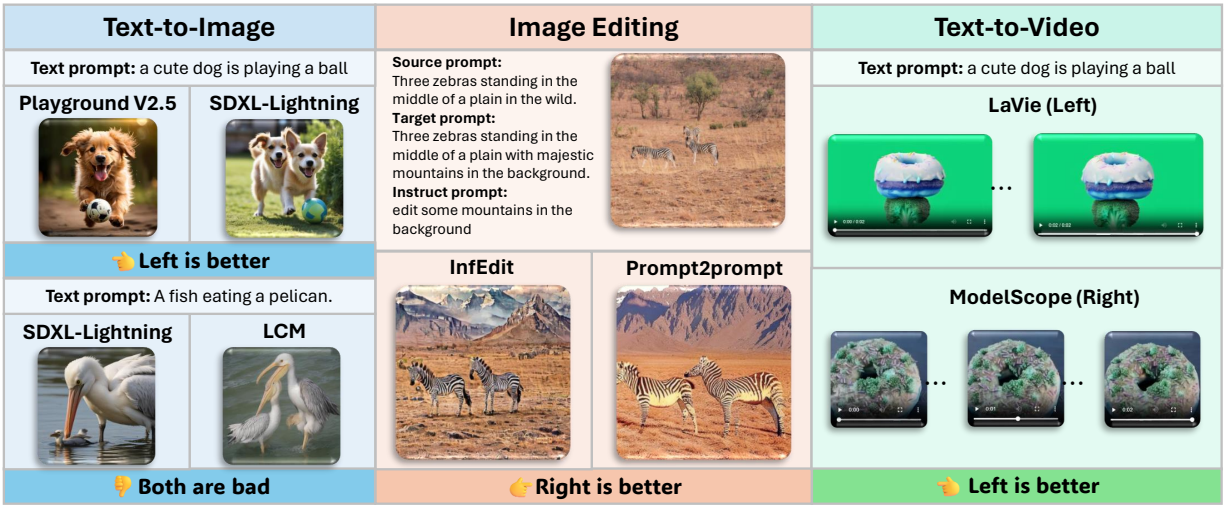


Figure 4.6: Example of votes from users on the GenAI-Arena for the three generative tasks

Case Study We present case studies in Figure 4.6, showcasing the votes collected for three generative tasks. These cases demonstrate that GenAI-Arena users can provide high-quality votes, even for the most advanced models. For instance, in the text-to-image task, the image generated by PlayGround V2.5 was preferred over that of SDXL-Lightning for the prompt "a cute dog is playing with a ball," as the latter depicted two dogs instead of one. Users can clearly distinguish and vote based on the quality of the outputs, even when both models complete the task. In the image editing task, the edited image from Prompt2Prompt appeared more natural than the one from InfEdit, leading users to make a definitive vote. Similarly, votes collected for the text-to-video task were also of high quality.

4.5 GenAI-Bench

4.5.1 Dataset

We applied Llama Guard Inan et al. (2023) as an NSFW filter to ensure that the user input prompt is appropriate for a wide range of audiences and protects users of the benchmark from exposure to potentially harmful or offensive content. In the text-to-image generation task, we collect 4.3k anonymous votes in total and there are 1.7k votes left after filtering for the safe content. We observe a large amount of the prompt is filtered out due to sexual content, which takes up 85.6% of the abandoned data. In the text-guided image editing task, we collect 1.1k votes from users before filtering. After applying Llama Guard, there are 0.9k votes for the image edition being released. In this task, 87.5% of the unsafe inputs contain violent crimes, and the other 12.5% is filtered out resulting from sex-related crimes. For text-to-video generation task, our platform collects 1.2k votes before post-processing. After cleaning it with the NSFW filter, we release the remaining 1.1k votes. All of the unsafe data abandoned in this task is due to the sexual content. We released the current version of GenAI-Bench[1] on the HuggingFace Dataset website, with an MIT license to allow the reuse with or without modification.

4.5.2 GenAI-Bench Leaderboard

To construct the GenAI-Bench leaderboard, we propmt MLLMs to output preference labels of AI generated contents, where templates are defined in subsection B.2.3. Specifically, We selected MLLMs including GPT-4o Hurst et al. (2024), Gemini-1.5-Pro Team et al. (2024), Idefics2 Laurençon et al. (2024), etc., and ask them to output 4 labels: "[[A>B]]", "[[B>A]]", "[[A=B=GOOD]]", and "[[A=B=BAD]]". We then compare them with actual human preference labels collected through the GenAI-Arena using the exact match metric. As shown in Table 4.7, open-source model still lag behind close-source MLLMs such as GPT-4o and Gemini, indicating a lack of generalization ability in vision reasoning of open-source MLLMs. We also tried models including Fuyu Bavishi et al. (2023), Kosmos-2 Peng et al. (2023), Otter Li et al. (2025a), Mantis Jiang et al. (2024a), etc., but found that they cannot follow the instruction well to output reasonable labels.

[1] https://huggingface.co/datasets/TIGER-Lab/GenAI-Bench

Table 4.7: GenAI-Bench leaderboard designed to benchmark MLLMs's ability in judging the quality of AI generative contents by comparing with human preferences. Numbers are accuracy (%).

Model	Image Generation	Image Editing	Video Generation	Average
Random	25.36	25.90	25.16	25.47
Idefics1 Laurençon et al. (2023)	0.81	5.66	0.19	2.22
InstructBLIP Dai et al. (2023)	3.11	19.80	3.74	8.89
QwenVL Bai et al. (2023)	26.63	14.91	2.15	14.56
CogVLM Wang et al. (2024b)	29.34	0.00	24.60	17.98
VideoLLaVA Lin et al. (2024a)	37.75	26.66	0.00	21.47
BLIP-2 Li et al. (2023a)	26.34	26.01	16.93	23.09
MiniCPM-V-2.5 Hu et al. (2024b)	37.81	25.24	6.55	23.20
LLaVA-1.6-7B Liu et al. (2024c)	22.65	25.35	21.70	23.24
Idefics2 Laurençon et al. (2024)	42.25	27.31	16.46	28.67
LLaVA-1.5-7B Liu et al. (2024b)	37.00	26.12	30.40	31.17
Gemini-1.5-Pro Team et al. (2024)	44.67	**55.93**	46.21	48.94
GPT-4o Hurst et al. (2024)	**45.59**	53.54	**48.46**	**49.19**

4.6 Summary

This chapter developed GenAI-Arena, an open platform for collecting human preferences over generated images and videos. Users compare anonymous outputs for the same request, providing feedback for ranking models across text-to-image generation, image editing, and text-to-video generation. I released these comparisons as GenAI-Bench to assess how well automatic judges agree with human preferences. The multimodal LLMs tested frequently disagree with human votes, and GPT-4o, the strongest judge tested, matched the human label on 49.19% of comparisons, so human preferences remain the reference for evaluating visual generation.

The collected preferences reflect the users and prompts represented in the arena, so the resulting rankings may not generalize to other user groups or settings. The benchmark measures agreement with individual votes, which can differ across people, and its findings are limited to the automatic judges evaluated in this study. Together with Chapter 3, this chapter studies how learned evaluators and direct human comparisons can provide feedback for evaluation, with human judgments serving as a common reference. Part II moves from evaluating generated outputs to using feedback for candidate selection and model learning, studying how program execution and interaction with environments can support these uses.

Part II

Improving Generative Models with Feedback

Overview of Part II

Part I studied feedback for evaluating generated outputs, while this part poses the second research question, how feedback can be designed to remain reliable while a model is optimized against it. Using feedback as a training reward requires it to remain reliable as the policy changes through optimization. Although TIGERScore can provide scores at scale, the small-scale experiment in Section 3.5.2 shows that a policy trained against it increasingly produces refusals that receive favorable scores, while its performance on independent evaluations declines. Policy optimization can thus reach responses that the evaluator's training data did not cover and exploit what it does not see.

Chapter 5 studies execution feedback for code, where a generated program can be run and checked against tests. I develop AceCoder, which synthesizes and filters test cases for existing coding questions and uses execution results to construct preference pairs. These pairs train reward models for selecting among candidate programs, while reinforcement learning uses either a learned reward or a direct test-pass reward. The results show gains in candidate selection and, with direct test-pass rewards, policy training, while also revealing limitations of the learned reward when used for optimization. Execution therefore provides a source of feedback for model improvement, although its reliability depends on the tests and on how their outcomes are used.

Chapter 6 extends this setting from evaluating a finished program to learning through multi-turn interaction with tools and environments. I develop VerlTool, a framework that supports these interactions during reinforcement learning across different domains. Tool observations enter the policy's context during the task, while final task checks and auxiliary terms provide training rewards. The chapter then asks whether rewarding tool use also improves task performance, and how much of a tool's output the policy needs to read. The experiments show that encouraging more tool use does not necessarily improve task performance. This extends the study of feedback from checking a completed output to designing the information and rewards an agent receives throughout a task.

Chapter 5

Execution Feedback for Generated Code

5.1 Introduction

This chapter opens Part II, which asks how feedback can improve a model rather than only judge its outputs. Improving a model with feedback requires a reward for every sampled output, and code is the setting where that reward can come from running the output itself.[1] The models to be improved are coder models such as Qwen2.5-Coder (Hui et al., 2024) and DeepSeek-Coder (Guo et al., 2024a), which are pre-trained and then supervised fine-tuned (SFT) on large-scale coding data from web resources like Common Crawl or GitHub.

Though strong performance has been achieved through SFT (Luo et al., 2023b; Wei et al., 2024), very few models have explored the potential of reinforcement learning (RL) (Ouyang et al., 2022), which has proven effective in other domains such as mathematical reasoning like DeepSeek-R1 (Guo et al., 2025). We argue that this absence of RL-based training in coder models is primarily due to two key challenges:

(1) Lack of reliable reward signals for code generation. In tasks such as mathematical problem-solving, rewards can be easily derived from rule-based string matches with reference answers (Guo et al., 2025) or large-scale human annotations (Ouyang et al., 2022). In contrast, evaluating code quality typically requires executing test cases to measure the pass rate, making reward signal design more complex. This also explains why existing reward models like Skywork (Liu et al., 2024a) can hardly generalize to the coding domain (see subsection 5.4.4).

[1]The material in this chapter is adapted from Zeng et al. (2025). Huaye Zeng and I contributed equally to that work.

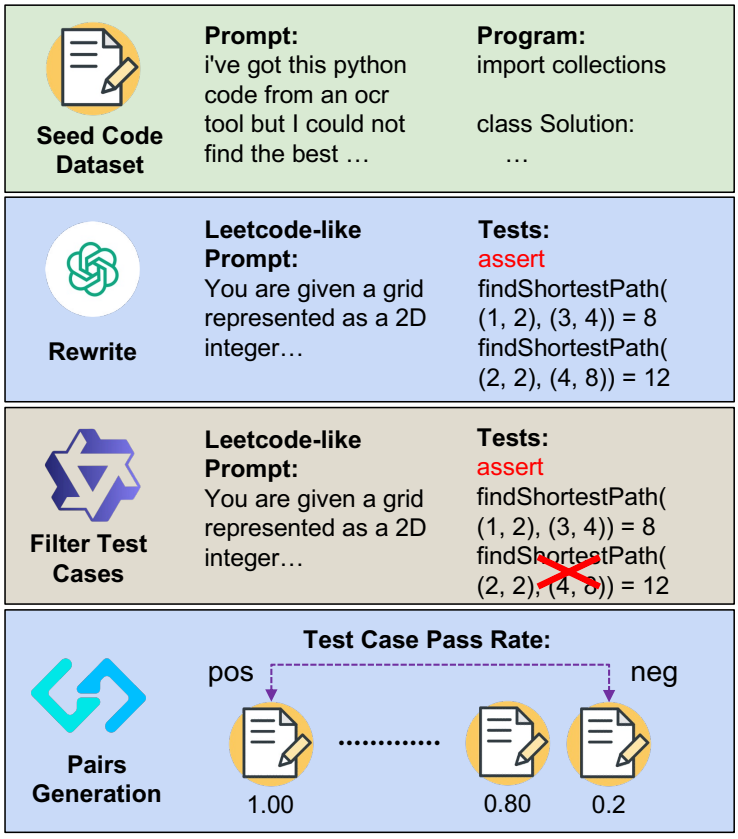


Figure 5.1: Overall Workflow of our model: we start from the seed code dataset to create well-formatted questions and corresponding test cases. Then we adopt strong models to filter the noisy test cases. Finally, we adopt these test cases to harvest positive and negative program pairs for reward model training and RL.

(2) Scarcity of large-scale coding datasets with reliable test cases. Most existing coding datasets like APPS (Hendrycks et al., 2021a; Chen et al., 2021) heavily rely on costly human expert annotations for test cases, which limits their scalability for training purposes. The largest dataset is TACO (Li et al., 2023b), containing 25K examples collected from popular coding competition websites, sources that have already been extensively utilized during the pre-training phase.

Therefore, we curate AceCode-87K, on which we trained our reward models: AceCode-RM-7B and AceCode-RM-32B. Comprehensive experiments of best-of-N sampling show that AceCode-RM can significantly boost existing LLM's performance on coding benchmarks. For example, AceCode-RM-7B can improve the performance

of Llama-3.1-8B-Instruct by an average of 8.4 points across the 4 coding benchmarks, i.e. HumanEval (Liu et al., 2023a), MBPP (Liu et al., 2023a), BigCodeBench (Zhuo et al., 2025) and LiveCodeBench (Jain et al., 2025a). Even for the stronger coder model Qwen2.5-Coder-7B-Instruct, our "7B+7B" combination still gets an average of 2.6 improvements. AceCode-RM-32B is even more powerful, which pushes the former two numbers to 9.0 and 4.4 respectively, showcasing the effectiveness of AceCode-RM.

Additionally, both AceCode-RM-7B and AceCode-RM-32B models demonstrate strong performance on the RM Bench (Liu et al., 2025b), outperforming existing state-of-the-art reward models across various categories. Specifically, AceCode-RM-32B achieves the highest average score of 76.1, leading in key categories such as Coding, Chat, Normal, and Hard tasks. Even the smaller AceCode-RM-7B model shows notable competitiveness, excelling particularly in Coding and Hard categories.

Furthermore, we adopt AceCode-RM-7B and test case pass rate separately to do reinforcement learning with reinforce++ (Hu, 2025) over coder models. Experiments show 2.1 and 0.7 points of average improvement when starting from Qwen2.5-7B-Ins and the Qwen2.5-Coder-7B-Ins respectively, making the latter even more powerful than GPT-4-Turbo on benchmarks like MBPP. Inspired by the recent DeepSeek-R1 (Guo et al., 2025), we also perform RL training directly from the Qwen2.5-Coder-7B-base model and saw a surprising 25% improvement on HumanEval-plus and 6% improvement on MBPP-plus (Liu et al., 2023a) with merely 80 optimization steps (48 H100 GPU hours). These improvements are also generalizable to other more difficult benchmarks.

At the time of publication, this was the first work to perform reward model training and reinforcement learning for code generation using a fully automated pipeline that synthesizes large-scale reliable tests.

5.2 Related Works

5.2.1 Synthesizing Test Cases

Automatic test generation is a widely used approach for verifying the correctness of LLM-generated programs. Prior work has commonly employed the same LLM that generates the programs to also generate test cases, selecting the most consistent program from multiple sampled outputs in a self-consistency manner (Chen et al., 2023; Huang et al., 2024a; Jiao et al., 2024). However, these generated test cases often suffer from significant hallucinations. To address this issue, Algo (Zhang et al., 2023b) introduced the use of an oracle

program solution to improve test case quality. While similar in spirit to our test case filtering approach, Algo constructs its oracle solution by exhaustively enumerating all possible combinations of relevant variables, whereas we leverage a stronger coder LLM to generate the oracle solution. Beyond using test cases as verification signals, Clover (Sun et al., 2023) enhances program verification by performing consistency checks between code, docstrings, and formal annotations, incorporating formal verification tools alongside LLMs.

5.2.2 Reinforcement Learning for LLM

RLHF, PPO, GRPO, and reinforcement learning with verifiable rewards are reviewed in Section 2.2. The work below applies them to code.

Given that coding is an inherently verifiable task, recent studies have explored RL techniques that leverage direct execution accuracy as a reward signal. PPOCoder (Shojaee et al., 2023) and CodeRL (Le et al., 2022) demonstrated the effectiveness of PPO-based RL for coding tasks, while RLEF (Gehring et al., 2025) extended this approach to multi-turn settings by incorporating execution feedback at each step. StepCoder (Dou et al., 2024) refined the reward mechanism by assigning rewards at a more granular level, considering only successfully executed lines of code. Additionally, DSTC (Liu et al., 2024e) explored the application of Direct Preference Optimization (DPO) to code generation by using self-generated test cases and programs.

Despite these advancements, most prior RL-based approaches for coding have been constrained by the use of pre-annotated datasets such as APPS (Hendrycks et al., 2021a), which consists of only 5,000 examples, with most problems having a single test case. This limited data availability poses challenges to scalable RL training. Furthermore, the potential of reward models for coding remains largely unexplored. In this chapter, we address these limitations by automatically synthesizing test cases and leveraging trained reward models for reinforcement learning, demonstrating the scalability and effectiveness of our approach.

5.2.3 Automated Test Generation in Software Engineering

Automated test generation is an established area of software engineering. Randoop (Pacheco et al., 2007), EvoSuite (Fraser and Arcuri, 2011), and Pynguin (Lukasczyk and Fraser, 2022) generate executable tests for existing programs, using execution feedback or search to explore program behavior. Research on the test oracle problem studies how expected behavior can be established, including the use of alternative implementations

as pseudo-oracles (Barr et al., 2015; Weyuker, 1982). More recently, TestGen-LLM (Alshahwan et al., 2024) combines LLM-generated tests with build, execution, and coverage checks, while EvalPlus (Liu et al., 2023a) expands code-generation benchmarks with LLM-generated and mutation-based test inputs. These contributions provide context for AceCoder, which uses synthesized tests and proxy-based filtering to construct execution-derived feedback for selection and learning.

5.3 Methodology

In this section, we will introduce the overall methodology of AceCoder. We begin with formulations of the problems we are investigating, including reward model training and reinforcement learning for LLMs. Finally, we elaborate on how we synthesize the test cases and construct the AceCode-87K.

5.3.1 Problem Formulation

Reward Model Training Let x denote the coding question and $\mathbf{y} = \{y_1, \cdots, y_t\}$ denote the program solution, where y_i represents the i-th token of the program solution and $(\mathbf{x}, \mathbf{y}) \in D$. Assuming θ represents the parameters of the model, then n responses $(\mathbf{y}^1, ..., \mathbf{y}^n)$ will be sampled from the model π_θ given the input $\mathbf{x}$. Let $(s_1, ..., s_n)$ be the target rewards, i.e. the test case pass rates in our scenario, then we define the Bradley-Terry loss (Bradley and Terry, 1952) for every pair of responses $\mathbf{y}^i$ and $\mathbf{y}^j$ with scores of s_i and s_j when we are training a reward model R_ϕ as follows:

$$\mathcal{L}_\phi(\mathbf{x}, s_i, s_j) = \mathbf{1}[s_i > s_j] \log \sigma(R_\phi(\mathbf{x}, \mathbf{y}^i) - R_\phi(\mathbf{x}, \mathbf{y}^j))$$

where $\mathbf{1}[\cdot] = 1$ if the expression inside the brackets is true, otherwise, it's 0. This is the Bradley–Terry loss of Section 2.2 (Equation (2.6)) applied to every ordered pair among the n sampled programs. The final loss function for the reward training is:

$$\mathcal{L}(\phi) = -\frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j=1}^{n} \mathcal{L}_\phi(\mathbf{x}, s_i, s_j) \tag{5.1}$$

That means the reward model is trained to assign higher values to preferred responses and lower values to non-preferred ones, maximizing the difference between these ratings.

Best-of-N Sampling Best-of-N selection is defined in Section 2.2 (Equation (2.7)). We use it as the first test of a trained reward model R_ϕ, returning the highest-scored of N sampled programs.

Reinforcement Learning We can finally conduct reinforcement learning for the original policy model π_θ after we get a well-trained reward model R_ϕ. Proximal Policy Optimization (PPO) is an actor-critic RL algorithm that is widely used for LLM's RL process. Let $\pi_{\theta_{old}}$ be the reference model and π_θ be the current policy model that is iteratively updated during the RL training. We denote $r_t(\theta)$ as the probability ratio of the current policy model over the old policy model on the t-th generated token:

$$r_t(\theta) = \frac{\pi_\theta(y_t|\mathbf{x}, \mathbf{y}_{<t})}{\pi_{\theta_{old}}(y_t|\mathbf{x}, \mathbf{y}_{<t})} \tag{5.2}$$

Then the PPO algorithms optimize the LLM by the following surrogate objective:

$$\mathcal{L}_{PPO}(\theta) = -\frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|} \min\left[r_t(\theta)A_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)A_t\right]$$

where $\mathbf{y} \sim \pi_{\theta_{old}}(\cdot|x)$, and A_t is the advantage computed through the Generalized Advantage Estimation (GAE) (Schulman et al., 2016) via the rewards generated by R_ϕ and the learned value function V_ψ. The PPO training objective will force the policy model π to increase the probability of generating tokens with higher A_t and decrease the probability ratio of generating tokens with lower A_t until the clipped bounds $1+\epsilon$ and $1-\epsilon$ are reached, respectively.

However, PPO usually requires training an additional value model V_ψ and thus makes the training inefficient. Recently, there have been some other works like Reinforecement++ (Hu, 2025) that eliminate the need for a value model but instead compute advantage only using the rewards generated by R_ϕ and the KL-divergence of the tokens after the t-th token. This makes the RL process more efficient and has also proved to be more stable.

5.3.2 AceCode-87K

To be able to train a reward model specifically designed for code generation, the first step is to synthesize reliable test cases for each coding problem and use them as training signals. In this section, we explain the whole procedure of constructing AceCode-87K step by step. We show the overall statistics in Table 5.1.

Test Case Synthesis from Seed Dataset We start from existing coding datasets with provided question $\mathbf{x}$ and corresponding program $\mathbf{y}$. Specifically, we combine Magicoder-Evol-Instruct[2], Magicoder-OSS-Instruct-75K[3], and StackPyFunction[4] as our seed dataset. We only keep the questions written in Python that contain either a function or a class, resulting in a total of 124K entries. We find that these datasets contain highly noisy questions that cannot be easily evaluated using test cases. Therefore, we feed every question-solution pair ($\mathbf{x}$, $\mathbf{y}$) into a GPT-4o-mini (Hurst et al., 2024) to propose a refined LeetCode-style question $\mathbf{x_r}$ with highly structured instructions. Meanwhile, we also prompt it to 'imagine' around 20 test cases $(t_1, ..., t_m)$ for each refined coding question $\mathbf{x_r}$ based on its understanding of the expected behavior of the desired program. See prompt template used in subsection B.3.2. Please note that we do not use the program solution $\mathbf{y}$ from the existing datasets at all in our final curated AceCode-87K. These datasets are purely used as seeds to help LLM formulate well-structured coding problems.

Test Case Filtering These 'imagined' test cases generated from the LLM contain severe hallucinations. To filter out those hallucinated test cases, we facilitated a stronger coder model Qwen2.5-Coder-32B-Instruct (Hui et al., 2024) as a proxy to perform quality control. Specifically, we prompt it for each $\mathbf{x_r}$ to generate a program $\mathbf{y}'$ and then run these programs over the test cases to approximate their quality. We removed all test cases t_i where the generated solution program $\mathbf{y}'$ could not pass. Furthermore, we removed questions with fewer than 5 tests after filtering, as these questions might be overly ambiguous. With the above filtering, we constructed the AceCode-87K with 87.1K distinct coding questions and 1.38M cleaned test cases, as represented by $(\mathbf{x_r}, (t_1, ..., t_{m_c}))$, where m_c represents the number of test cases after filtering.

Human Case Study Finally, as a last verification step, we conducted a human case study by randomly selecting 40 questions, each with 5 randomly sampled test cases, resulting in 200 manually annotated test cases. Encouragingly, only 3 out of these 200 test cases were found invalid upon review, indicating that our filtering methodology effectively mitigates hallucinations and preserves test-case quality. Moreover, Table 5.2 presents various pass-rate statistics for AceCode-87K, illustrating the overall difficulty and performance trends.

[2] ise-uiuc/Magicoder-Evol-Instruct-110K
[3] ise-uiuc/Magicoder-OSS-Instruct-75K
[4] bigcode/stack-dedup-python-fns

Subset	Evol	OSS	Stack Python	Overall
Before Filtering				
# Examples	36,256	37,750	50,000	124,006
# Avg Test Cases	19.33	17.21	18.27	18.26
After Filtering				
# Examples	26,920	25,862	34,367	87,149
# Avg Test Cases	15.14	16.33	16.08	15.87
# Pairs	89,089	91,636	126,784	307,509

Table 5.1: Dataset statistics of AceCode-87K before and after test-case filtering.

Metric	Value
Pass@1	34.90%
Pass@4	38.50%
Pass@8	39.20%
Pass@16	39.60%
Avg. test pass rate	70.30%
Questions where all 16 responses pass all tests	22.80%

Table 5.2: Pass rates on AceCode-87K after filtering (Qwen2.5-Coder-7B-Instruct; 16 responses per question; temperature 1.0).

Preference Pairs Construction We propose to use the Bradley-Terry loss to train the reward model as defined in Equation 5.1. Therefore, we need to construct (question, [positive program, negative program]) data from AceCode-87K. Specifically, we sample programs $(\mathbf{y}^1, ..., \mathbf{y}^n)$ from existing models (e.g. Llama-3.1 (Grattafiori et al., 2024)) w.r.t $\mathbf{x}_r$ and utilize the test-case pass rate to distinguish positive and negative programs.[5] Since the pass rate s_i for the sampled program $\mathbf{y}^i$ can be any number between $[0, 1]$, a minor difference in pass rate may not represent that one program is more accurate than another. Therefore, instead of using $\mathbf{1}[s_i > s_j]$ to select the preference pairs, we have thus modified the selection rules to be:

$$\mathbf{1}[s_i > s_j + 0.4, s_i > 0.8, s_j > 0] \tag{5.3}$$

This is to ensure the preferred program has at least a 0.8 pass rate to make sure it represents a more correct program. Also, we find that many sampled programs with 0 pass rates can be caused by some small syntax errors or some Python packaging errors during evaluation. We chose not to include them as the preference pair to make sure our constructed datasets represent only the preference based on the valid pass rate. We also ensure that the sampled programs all come from the backbone of R_ϕ, so the reward model is trained in an on-policy way. After that, we train our reward model R_ϕ by fully fine-tuning an instruction coding model. Specifically, we extract the last token's final hidden representation and pass it through a linear model head that generates a single scalar output, which is optimized via the loss function defined in Equation 5.1.

[5]Section B.3.1 contains a detailed model breakdown for how different models are used throughout the training and evaluation process of AceCode-RM for more clarifications.

5.4 Experiments

Mehod	# N	HumanEval		MBPP		BigCodeBench-C		BigCodeBench-I		LiveCodeBench	**Average**
		-	Plus	-	Plus	Full	Hard	Full	Hard	V4	
GPT-4o (0806)	1	92.7	87.2	87.6	72.2	58.9	36.5	48.0	25.0	43.6	61.3
DeepSeek-V2.5	1	90.2	83.5	87.6	74.1	53.2	29.1	48.9	27.0	41.8	59.5
DeepSeek-V3	1	91.5	86.6	87.6	73.0	62.2	39.9	50.0	27.7	63.5	64.6
Qwen2.5-Coder-32B	1	92.1	87.2	90.5	77.0	58.0	33.8	49.0	27.7	48.3	62.6
Inference Model = Mistral-7B-Instruct-V0.3											
Greedy	1	36.6	31.1	49.5	41.3	25.9	6.1	20.1	5.4	7.3	24.8
Average	64	37.1	30.8	45.1	38.0	21.7	4.2	17.6	3.0	4.0	22.4
Oracle	64	87.2	78.0	83.9	73.5	68.4	37.8	58.5	31.1	24.3	60.3
	16	65.9	56.7	59.3	**52.4**	35.1	10.1	29.3	8.8	11.9	36.6
AceCodeRM-7B	32	68.3	58.5	59.8	51.6	37.4	8.8	30.7	10.8	14.6	37.8
	64	71.3	61.6	59.8	51.6	39.4	6.8	31.8	9.5	15.4	38.6
Δ (RM-greedy)	-	+34.8	+30.5	+10.3	+11.1	+13.5	+4.1	+11.7	+5.4	+8.1	+13.8
	16	68.3	61.0	58.7	49.5	37.7	11.5	30.9	10.1	12.9	37.8
AceCodeRM-32B	32	72.6	**65.9**	**61.6**	51.6	40.5	9.5	33.9	13.5	16.1	40.6
	64	**75.0**	64.6	60.6	50.0	**42.7**	**15.5**	**35.6**	**13.5**	**17.4**	**41.7**
Δ (RM-greedy)	-	+38.4	+34.8	+12.2	+10.3	+16.8	+9.5	+15.5	+8.1	+10.1	+16.9

Table 5.3: AceCode-RM's best-of-n results on several benchmarks. Specifically, -C means completion split and -I means instruct split of BigCodeBench. Each Δ is that reward model's best score across N in the column minus the greedy score, and might be off by 0.1 due to rounding.

Mehod	# N	HumanEval		MBPP		BigCodeBench-C		BigCodeBench-I		LiveCodeBench	**Average**
		-	Plus	-	Plus	Full	Hard	Full	Hard	V4	
Inference Model = Llama-3.1-8B-Instruct											
Greedy	1	68.9	62.2	67.2	54.8	38.5	12.8	31.8	13.5	18.0	40.9
Average	64	61.7	54.9	64.5	54.5	32.8	10.1	26.6	9.0	13.8	36.4
Oracle	64	93.9	90.2	92.1	82.3	80.0	54.7	67.9	48.6	40.8	72.3
	16	77.4	70.7	**76.5**	**64.3**	45.8	20.3	36.4	12.2	26.1	47.7
AceCodeRM-7B	32	79.9	72.6	76.2	62.4	47.6	23.0	37.3	13.5	27.3	48.9
	64	81.7	74.4	74.6	61.9	47.8	**23.6**	38.1	13.5	27.6	49.3
Δ (RM-greedy)	-	+12.8	+12.2	+9.3	+9.5	+9.3	+10.8	+6.2	0.0	+9.6	+8.4
	16	82.3	74.4	72.8	60.6	49.8	20.3	38.4	13.5	27.5	48.8
AceCodeRM-32B	32	81.7	76.2	72.8	60.6	**50.4**	22.3	39.1	13.5	30.3	49.6
	64	**85.4**	**79.3**	72.0	59.0	48.5	19.6	**40.0**	**13.5**	**31.0**	**49.8**
Δ (RM-greedy)	-	+16.5	+17.1	+5.6	+5.8	+11.8	+9.5	+8.2	+0.0	+13.0	+9.0

Table 5.3: AceCode-RM's best-of-n results (continued): Llama-3.1-8B-Instruct.

Mehod	# N	HumanEval -	HumanEval Plus	MBPP -	MBPP Plus	BigCodeBench-C Full	BigCodeBench-C Hard	BigCodeBench-I Full	BigCodeBench-I Hard	LiveCodeBench V4	**Average**
Inference Model = Qwen2.5-Coder-7B-Instruct											
Greedy	1	**91.5**	86.0	82.8	71.4	49.5	19.6	41.8	20.3	34.2	55.2
Average	64	86.0	80.1	77.9	65.6	45.3	18.6	37.3	16.2	31.8	51.0
Oracle	64	98.2	95.7	97.4	90.7	80.9	62.8	73.5	53.4	57.4	78.9
AceCodeRM-7B	16	90.2	82.9	**88.6**	74.9	53.8	20.9	45.0	21.6	40.1	57.6
	32	90.9	86.0	87.8	74.1	53.4	25.0	43.9	19.6	39.8	57.8
	64	90.9	85.4	87.6	73.8	52.9	24.3	43.5	21.6	40.1	57.8
Δ (RM-greedy)	-	-0.6	0.0	+5.8	+3.4	+4.3	+5.4	+3.2	+1.4	+5.9	+2.6
AceCodeRM-32B	16	90.2	86.6	88.4	74.9	53.9	25.0	45.4	19.6	44.0	58.7
	32	90.2	**86.6**	88.4	**75.4**	**55.4**	**29.7**	45.6	21.6	43.5	**59.6**
	64	89.6	86.0	87.8	75.1	55.0	26.4	**46.1**	**22.3**	**44.5**	59.2
Δ (RM-greedy)	-	-1.3	+0.6	+5.6	+4.0	+6.0	+10.1	+4.3	+2.0	+10.3	+4.4

Table 5.3: AceCode-RM's best-of-n results (continued): Qwen2.5-Coder-7B-Instruct.

5.4.1 Reward Model Training Setup

We mainly use Qwen2.5-Coder-7B-Instruct [6] as the backbone of the reward model and sample 16 responses from it for each question in AceCode-87K. Finally, following the rule defined in Equation 5.3, around 300K preference pairs were created out of 46,618 distinct questions (37.34% of the total questions) that have at least one pair satisfying the condition, and other questions are not used.

Our reward model is trained using LlamaFactory (Zheng et al., 2024a). We apply full fine-tuning with DeepSpeed stage 3. We train for 1 epoch using a cosine learning rate schedule, starting at 1e-5 with a warmup ratio of 0.1 to gradually increase the learning rate in the initial training phase. The training batch size is set to 128. We enable BF16 precision to reduce memory overhead without compromising model fidelity. The training takes 24 hours on 8 x A100 GPUs.

5.4.2 Reinforcement Learning Setup

We perform RL training from three policy models: Qwen2.5-7B-Instruct [7], Qwen2.5-Coder-7B-Base [8], and Qwen2.5-Coder-7B-Instruct. Two types of reward can be used, i.e. the

[6] Qwen/Qwen2.5-Coder-7B-Instruct
[7] Qwen/Qwen2.5-7B-Instruct
[8] Qwen/Qwen2.5-Coder-7B

Model	HumanEval		MBPP		BigCodeBench (C)		BigCodeBench (I)		LiveCodeBench	Average
	-	Plus	-	Plus	Full	Hard	Full	Hard	V4	
RLEF-8B	-	67.5	-	57.0	-	-	-	-	-	-
RLEF-70B	-	78.5	-	67.6	-	-	-	-	-	-
PPOCoder-7B	78.7	-	67.0	-	-	-	-	-	-	-
StepCoder-7B	76.8	-	63.8	-	-	-	-	-	-	-
CodeGemma-7B	60.5	-	55.2	-	-	-	-	-	-	-
DSTC-33B	79.9	72.0	82.5	70.4	51.6	22.3	41.0	18.2	-	-
Baseline = Qwen2.5-7B-Instruct										
Baseline	81.7	73.2	79.4	67.7	45.6	16.9	38.4	14.2	29.0	49.6
AceCoder$_{RM}$	83.5	**77.4**	**83.1**	**71.2**	**46.8**	**16.9**	39.0	14.9	**30.3**	**51.5**
AceCoder$_{Rule}$	**84.1**	77.4	80.2	68.3	46.8	15.5	**40.2**	**15.5**	30.1	50.9
Δ (RL-baseline)	+2.4	+4.3	+3.7	+3.4	+1.2	0.0	+1.8	+1.4	+1.3	+2.0
Baseline = Qwen2.5-Coder-7B-Base										
Baseline	61.6	53.0	76.9	62.9	45.8	16.2	40.2	14.2	**28.7**	44.4
AceCoder$_{RM}$	83.5	75.6	80.2	67.2	41.9	14.9	36.8	16.2	25.7	49.1
AceCoder$_{Rule}$	**84.1**	**78.0**	**82.3**	**69.3**	**48.6**	**18.2**	**43.2**	**18.2**	28.5	**52.3**
Δ (RL-baseline)	+22.5	+25.0	+5.4	+6.4	+2.8	+2.0	+3.1	+4.1	-0.2	+7.9
Baseline = Qwen2.5-Coder-7B-Instruct										
Baseline	**91.5**	**86.0**	82.8	71.4	49.5	19.6	41.8	**20.3**	34.2	55.2
AceCoder$_{RM}$	89.0	84.1	**86.0**	**72.8**	50.4	18.9	42.0	19.6	**35.0**	55.3
AceCoder$_{Rule}$	90.9	84.8	84.1	71.7	**50.9**	**23.0**	**43.3**	19.6	34.9	**55.9**
Δ (RL-baseline)	-0.6	-1.2	+3.2	+1.3	+1.4	+3.4	+1.5	-0.7	+0.8	+0.7

Table 5.4: AceCoder's Performance after RL tuning using Reinforcement++ algorithm. We start with 3 different initial policy models and 2 kinds of reward types, where *RM* means using our trained AceCode-RM and *Rule* means using the binary pass rate. Results show consistent improvement across various benchmarks.

trained reward model AceCode-RM-7B and the rule-based reward, i.e. pass rate over the test cases in AceCode-87K. During training, we set the pass rate to be a binary reward, which is 1.0 when all test cases pass, otherwise 0. This is similar to the verifiable reward used in Tulu3 (Lambert et al., 2025a) and DeepSeek-R1 (Guo et al., 2025). Similar to DeepSeek-R1 (Guo et al., 2025), we also experiment with RL from the base model because SFT may cause the search space of the model to be stuck in a local minimum. Since coding is also a highly verifiable task like math, we include the Qwen2.5-Coder-7B-Base in our experiments.

We have trained different policy model backbones with different rewards, resulting in 6 RL models in total. All the RL-tuning is based on `OpenRLHF` (Hu et al., 2024a). We adopt the Reinforcement++ (Hu, 2025) algorithm instead of PPO to improve the training

efficiency without training the value model. It's also proved to be more stable than PPO and GRPO. We train our model on a subsampled hard version of AceCode-87K, where we keep the 25% of the questions with lower average pass rates and higher variance. This is to ensure the question is hard and that the sampled programs are diverse enough. For the training hyperparameters, we set the rollout batch size to 256, and 8 programs are sampled per question. The training batch size is 128 with a learning rate of 5e-7. All the models are trained for 1 episode and finished in 6 hours on 8 x H100 GPUs.

5.4.3 Evaluation Setup

We evaluate our method on four established code-focused benchmarks: HumanEval(+) (Chen et al., 2021; Liu et al., 2023a), MBPP(+) (Austin et al., 2021; Liu et al., 2023a), BigCodeBench (Zhuo et al., 2025) and LiveCodeBench (V4) (Jain et al., 2025a).

These benchmarks collectively cover a diverse array of coding tasks, enabling us to assess both the correctness and quality of generated code. For Best-of-N sampling, we adopt top-p sampling with a temperature of 1.0 to generate multiple (16/32/64) candidate solutions per question and then select the response with the highest reward for evaluation. For RL experiments, we use each benchmark's default setting, which is greedy sampling most of the time.

5.4.4 Main Results

Here we show the experimental results of the reward models and RL-trained models.

RM Results We conduct Best-of-N experiments on 3 inference models, specifically Mistral-Instruct-V0.3-7B(AI, 2023), Llama-3.1-Instruct-8B (Grattafiori et al., 2024), and Qwen2.5-Coder-7B-Insutrct (Hui et al., 2024; Yang et al., 2024a). We additionally report the average score across all generated samples and also the oracle score (pass@N) for better comparison.

According to Table 5.3, AceCode-RM can consistently boost the performance of inference models by a large margin compared to the greedy decoding results. On weaker models like Mistral (AI, 2023) and Llama-3.1 (Grattafiori et al., 2024), the overall improvements range from 8.4 to 16.9 points. These improvements can be attributed to our reward model's ability to identify high-quality completions among multiple candidates, thereby

reducing the impact of suboptimal sampling on the final output. Notably, these gains become more pronounced on benchmarks where the gap between greedy decoding and oracle performance (i.e., the best possible completion among all samples) is larger. In such cases, the variance among sampled completions is relatively high, providing greater opportunities for the reward model to pinpoint and elevate top-tier responses.

Greedy decoding systematically outperforms the average sampled performance, reflecting the strong code generation capability of these inference models. Consequently, while most reward models achieve best-of-N results above the average, we consider a reward model effective only if it surpasses the performance of greedy decoding.

RL Results We perform RL training over 3 different initial policy models in Table 5.4 with model-based and rule-based rewards. When starting from Qwen2.5-Instruct-7B, we can see that RL tuning can consistently improve performance, especially for HumanEval and MBPP. Even for the Plus version with more and harder test cases, the RL-tuned model also has more than 3 points of improvement.

When starting from the Qwen2.5-Coder-Instruct-7B itself, we can still observe improvements, especially when using the rule-based reward. For example, we get a 3.4-point improvement on BigCodeBench-Full-Hard. Using the reward model for RL can also bring a 3.2 improvement on MBPP. This highlights the charm of self-improvement given the reward model backbone is the same as the initial policy model. We compare our method with other RL-based models like RLEF (Gehring et al., 2025), PPOCoder (Shojaee et al., 2023), StepCoder (Dou et al., 2024), DSTC (Liu et al., 2024e), etc. We show that our 7B model is able to beat these competitors across the evaluation benchmarks.

Another experiment we conduct is to perform RL training directly from the base model Qwen2.5-Coder-7B-base. We show significant improvement, especially through test-case pass rewards on HumanEval, MBPP, and BigCodeBench-I. These results are achieved by only training for 80 steps. We believe further scaling up the training will lead to much larger gains.

Comparison with Other RMs We compare our AceCode-RM with 3 top-ranked RM on the RewardBench, including InternLM2-RM-8B (Cai et al., 2024), Skywork-Llama-3.1-8B, and Skywork-Gemma-27B (Liu et al., 2024a), where results are reported in Table 5.5. We can see that these general-purpose RM can hardly improve and sometimes decrease the performance through Best-of-N sampling compared to greedy sampling, showcasing the incapability in identifying the correct generated programs. On the other hand, our

Method & RM	HumanEval		MBPP		BigCodeBench-C		BigCodeBench-I		LiveCodeBench	Average
	-	Plus	-	Plus	Full	Hard	Full	Hard	V4	
Greedy	68.9	62.2	67.2	54.8	38.5	12.8	31.8	**13.5**	18.0	40.9
Average	50.1	42.2	57.9	47.2	22.0	10.6	18.2	12.0	14.9	30.6
InternLM2-RM-8B	57.9	55.5	66.7	54.0	38.7	8.8	29.8	8.8	15.1	37.3
Skywork-Gemma-27B	73.8	67.1	64.3	53.4	40.1	14.9	32.5	12.8	23.6	42.5
Skywork-Llama-3.1-8B	67.7	61.6	69.6	56.9	40.6	10.8	31.8	12.2	18.8	41.1
Δ (max(other RM)-greedy)	+4.9	+4.9	+2.4	+2.1	+2.1	+2.0	+0.6	-0.7	+5.6	+2.6
AceCode-RM-7B	**77.4**	**70.7**	**76.5**	**64.3**	**45.8**	**20.3**	**36.4**	12.2	**26.1**	**47.7**
Δ (RM-greedy)	+8.5	+8.5	+9.3	+9.5	+7.3	+7.4	+4.6	-1.4	+8.1	+6.8

Table 5.5: AceCode-RM's performance against other open-sourced reward models in terms of Best-of-16 sampling for Llama-3.1-8B-Inst. We can see the top-ranked RM on Reward Bench get little improvements compared to ours.

AceCode-RM surpasses all other publicly released reward models in our evaluation and consistently gets positive gains. These findings further underscore our assumption that previous RM training lacks reliable signals for codes and prove that our RMs can generate reliable and state-of-the-art reward signals in code generation tasks.

Model	Code	Chat	Math	Safety	Easy	Normal	Hard	Avg
Skywork/Skywork-Reward-Llama-3.1-8B	54.5	69.5	60.6	**95.7**	**89**	74.7	46.6	70.1
NVIDIA/Nemotron-340B-Reward	59.4	71.2	59.8	87.5	81	71.4	56.1	69.5
internlm/internlm2-20b-reward	56.7	63.1	66.8	86.5	82.6	71.6	50.7	68.3
internlm/internlm2-7b-reward	49.7	61.7	**71.4**	85.5	85.4	70.7	45.1	67.1
Skywork-Reward-Llama-3.1-8B-v0.2[1]	53.4	69.2	62.1	96	88.5	74	47.9	70.1
Skywork-Reward-Gemma-2-27B-v0.2[1]	45.8	49.4	50.7	48.2	50.3	48.2	47	48.5
AceCodeRM-7B	66.9	66.7	65.3	89.9	79.9	74.4	62.2	72.2
AceCodeRM-32B	**72.1**	**73.7**	70.5	88	84.5	**78.3**	**65.5**	**76.1**
Δ (AceCodeRM-7B - max(other RM))	7.5	-4.5	-6.1	-6.1	-9.1	-0.3	6.1	2.1
Δ (AceCodeRM-32B - max(other RM))	12.7	2.5	-0.9	-8	-4.5	3.6	9.4	6

[1] There is no official result for this model; however, the authors made best efforts to extend the original RM Bench code base to adapt to this new model.

Table 5.6: AceCode-RM's Performance on RM Bench against various other reward models. We can see that AceCode-RM-32B model performed best in Coding, Chat, Normal, Hard, and Average scores against all other reward models.

Moreover, we have also evaluated our AceCode-RM against other reward models on RM-Bench in Table 5.6, a benchmark designed to assess the reward model's capabilities across code synthesis, mathematical reasoning, and other tasks (Liu et al., 2025b). In this head-to-head comparison, AceCode-RM emerges as the clear state-of-the-art in

coding, hard tasks, and overall average. Remarkably, our 7 billion-parameter variant, AceCode-RM-7B, outperforms NVIDIA-Nemotron-340B-Reward(NVIDIA et al., 2024) by 7.50 points on the coding benchmark, proving that a more compact model can deliver superior reward estimates for code generation. Beyond code, AceCode-RM also generalizes well: it beats all other reward models on the average score, underscoring its robust reasoning and dialogue capabilities.

5.4.5 Ablation Studies

Method	HumanEval -	HumanEval Plus	MBPP -	MBPP Plus	BigCodeBench-C Full	BigCodeBench-C Hard	BigCodeBench-I Full	BigCodeBench-I Hard	LiveCodeBench V4	Average
Inference Model = Llama-3.1-8B-Instruct										
RM w/o Test Case Filter	73.8	65.9	73.3	61.4	44.6	17.6	35.5	9.5	25.1	45.2
RM w/ Test Filter	**77.4**	**70.7**	**76.5**	**64.3**	**45.8**	**20.3**	**36.4**	**12.2**	**26.1**	**47.7**
Δ (w/ Filter - w/o Filter)	+3.7	+4.9	+3.2	+2.9	+1.2	+2.7	+0.9	+2.7	+1.0	+2.5
Inference Model = Qwen2.5-Coder-7B-Instruct										
RM w/o Test Case Filter	**91.5**	**86.0**	86.0	72.2	52.5	**21.6**	43.4	19.6	36.9	56.6
RM w/ Test Filter	90.2	82.9	**88.6**	**74.9**	**53.8**	20.9	**45.0**	**21.6**	**40.1**	**57.6**
Δ (w/ Filter - w/o Filter)	-1.2	-3.0	+2.6	+2.6	+1.3	-0.7	+1.6	+2.0	+3.2	+1.0

Table 5.7: Ablation study on test-case filtering. Results are Best-of-16 sampling performance.

Method	HumanEval -	HumanEval Plus	MBPP -	MBPP Plus	BigCodeBench-C Full	BigCodeBench-C Hard	BigCodeBench-I Full	BigCodeBench-I Hard	LiveCodeBench V4	Average
Inference Model = Llama-3.1-8B-Instruct										
AceCode-RM (LLama)	65.9	59.1	69.6	57.9	42.7	12.8	32.9	**13.5**	19.9	41.6
AceCode-RM (Qwen)	**77.4**	**70.7**	**76.5**	**64.3**	**45.8**	**20.3**	**36.4**	12.2	**26.1**	**47.7**
Δ (Qwen-Llama)	+11.6	+11.6	+6.9	+6.3	+3.1	+7.4	+3.5	-1.4	+6.2	+6.1
Inference Model = Qwen2.5-Coder-7B-Instruct										
AceCode-RM (LLama)	87.8	81.7	82.0	67.7	50.5	**25.0**	39.0	19.6	32.4	54.0
AceCode-RM (Qwen)	**90.2**	**82.9**	**88.6**	**74.9**	**53.8**	20.9	**45.0**	**21.6**	**40.1**	**57.6**
Δ (Qwen-Llama)	+2.4	+1.2	+6.6	+7.1	+3.2	-4.1	+6.0	+2.0	+7.7	+3.6

Table 5.8: Comparison of AceCode-RM's performance trained on different base model, where AceCode-RM (Llama) is based on Llama-3.1-Inst-8B and AceCode-RM (Qwen) is based on Qwen-Coder-2.5-7B-Inst. Results are Best-of-16 sampling performance.

Test Case Quality Matters We also conduct experiments to investigate how filtering the test cases with a proxy model can affect the results. As shown in Table 5.7, training

RM on data after the filtering improves the performance significantly, especially for those hard code questions like MBPP-Plus and BigCodeBench-Hard (C/I). We believe this is because the test case filtering can ensure the remaining ones are consistent with each other and thus point to the same implicit program, which improves the quality of the rewards.

RM Backbone Matters Our results in Table 5.8 clearly show that changing the backbone of the reward model from Llama-3.1 to Qwen2.5 can significantly improve the Best-of-16 performance. This is because the Qwen2.5-Coder models have been pre-trained on way more code-related data compared to the Llama-3.1 models, and thus more knowledgeable when tuning it into a reward model.

Does R1-style Tuning Work? Inspired by the recent DeepSeek-R1 (Guo et al., 2025), we also conduct the RL directly from the base model without any SFT. It turns out we get huge improvements when using rule-based rewards. For example, we get 25.0 points of improvements on HumanEval-Plus after training only 6 hours from the Base Model, which is way more efficient than large-scale SFT. What's more, the AceCoder $_{Rule}$ improve the BigCodeBench-Instruct-Full's performance from 40.2 to 43.2, nearly the same performance with DeepSeek-R1-Distill-Qwen-32B (43.9) which was directly distilled from the DeepSeek-R1 Model. This further consolidates the finding of DeepSeek-Zero. However, we do find that using reward models for RL tuning can lead to worse results. We attribute this to the potential reward hacking during the tuning process.

5.5 Summary

This chapter developed AceCoder to turn automatically synthesized tests into feedback for code generation. I constructed AceCode-87K with 87.1K coding questions and 1.38M filtered tests, then used sampled programs' test pass rates to construct approximately 300K preference pairs. Reward models trained on these pairs improve average Best-of-N performance over greedy decoding across the evaluated generators (Table 5.3). Direct test-pass rewards also support reinforcement learning, under which Qwen2.5-Coder-7B-Base improves from 53.0 to 78.0 on HumanEval+ after 80 optimization steps (Table 5.4). The dataset, preference pairs, reward models, and training code are publicly available.

The feedback remains limited by the tests used to produce it. Proxy-based filtering does not guarantee that every expected output is correct or that the tests cover all relevant cases, and the manual review found three invalid tests among 200 sampled from 40 questions

(Section 5.3.2). The relative performance of learned and test-pass rewards also varies with the initial policy. In the base-model setting, the learned reward improves the overall average but produces declines on some benchmark splits. I discuss potential reward hacking as one possible explanation for these declines (Section 5.4.5).

This chapter uses execution results to evaluate completed programs and provide feedback for selection and learning. Chapter 6 extends this setting to multi-turn tool interaction, where observations can guide the policy's next action and task checks provide rewards for the resulting trajectory.

Chapter 6

Environment Feedback for Agent Trajectories

6.1 Introduction

"We shape our tools, and thereafter our tools shape us." — Marshall McLuhan

Large language models (LLMs) such as OpenAI's O-series (Contributors et al., 2024) and DeepSeek-R1 (Guo et al., 2025) have recently achieved striking advances, surpassing top human performers in challenging domains like mathematics (AIME) and programming (LiveCodeBench (Jain et al., 2025a), Codeforces (Quan et al., 2025)). A central driver of this progress is the paradigm of *reinforcement learning with verifiable rewards* (RLVR), which strengthens long-context reasoning during training. Through RLVR, LLMs exhibit emergent cognitive behaviors such as reflection, backtracking, and multi-step reasoning.

Yet these systems remain constrained in a fundamental way: they are unable to interact with the external world. Current LLM reasoning unfolds in a closed, single-turn setting without environmental feedback, often leading to brittle behaviors such as overthinking (Chen et al., 2024d) or hallucination (Yao et al., 2025). Conceptually, these models resemble a "brain in a vat", locked into self-contained simulations without grounding in interactive or physical reality.

To overcome this isolation, a parallel line of work has explored augmenting LLMs with the ability to use tools. Systems such as ToolFormer (Schick et al., 2023) and OpenHands (Wang et al., 2025c) extend models with supervised training on synthetic

tool-use data, enabling practical interaction with code interpreters, search engines, or APIs. However, these approaches primarily rely on imitation learning. They lack the agentic autonomy needed to learn directly from feedback and refine their behavior adaptively in open-ended environments.

Recent research begins to bridge this gap by combining tool use with RLVR, giving rise to a new paradigm we term **ARLT**—**A**gentic **R**einforcement **L**earning with **T**ool use. In ARLT, LLMs can actively engage with external tools such as code execution environments (Li et al., 2025c), search engines (Jin et al., 2025), image manipulators (Su et al., 2025), and domain-specific APIs (Feng et al., 2026). This interaction transforms training into a multi-turn, feedback-rich process that not only improves efficiency and reduces token usage but also fosters more robust agentic behaviors.

However, enabling ARLT poses significant challenges from a systems perspective. First, *rollout efficiency* becomes critical: multi-tool trajectories unfold asynchronously, with different tools producing results at varying speeds, demanding scalable asynchronous execution. Second, *tool management* remains fragmented: existing ARLT codebases are often tailored to specific tools, making it difficult to extend or reproduce results. Finally, *multi-modal support* is still underdeveloped: while most RL frameworks focus narrowly on text, emerging multimodal reasoning agents (e.g., Pixel-Reasoner (Su et al., 2025)) require handling tool outputs that include images, videos, or other structured modalities in a unified design.

These barriers have slowed community progress, limiting reproducibility, extensibility, and algorithmic innovation. To address them, we introduce VerlTool: an open-source, user-friendly, and efficient framework built on top of VeRL (Sheng et al., 2025), designed explicitly for ARLT that supports both text and multimodal training. Unlike prior systems, VerlTool enables multi-turn, stateful agentic training with tool use through four key contributions:

- **Upstream Alignment.** VerlTool inherits VeRL as a submodule, ensuring compatibility with upstream updates. This modular separation between RL training and agentic interaction simplifies maintenance and accelerates framework evolution.
- **Unified Tool Management.** We introduce a dedicated tool server with standardized interaction APIs, supporting diverse tools such as code execution, search, SQL/tabular reasoning, and vision utilities. Adding a new tool requires only a lightweight Python definition file, streamlining extensibility for both training and evaluation.
- **Asynchronous Rollouts.** By interacting with tool servers on a trajectory-by-trajectory basis rather than synchronously batch by batch, VerlTool eliminates

idle waiting time. This design yields up to 1.97× speedup during rollout execution.
- **Diverse ARLT Tasks.** We have implemented and tested VerlTool on six ARLT tasks, including Math, Search Retrieval, SQL, Visual Reasoning, Web Browsing, and SWE-Bench, achieving competitive performance with previous baselines while trained in a unified framework. We also present common findings in the agentic RL setting across these tasks.

We also use VerlTool to investigate how feedback design affects learning. We ask whether auxiliary rewards for tool use improve task performance, and how the length of tool observations affects results (Section 6.5). In visual reasoning, the model trained with auxiliary reward terms achieves higher accuracy than the accuracy-only model, whereas in the small-scale mathematics comparison, the tested auxiliary terms increase tool calls without improving final accuracy. In retrieval, increasing the observation budget improves performance at short lengths, while further increases give mixed results. These experiments examine how rewards and tool observations affect task performance within the framework.

In summary, VerlTool provides a principled, extensible, and efficient framework for ARLT, bridging the gap between isolated LLM reasoning and interactive agentic intelligence. By combining upstream-aligned RL infrastructure, unified tool integration, asynchronous execution, and diverse tasks, it paves the way for scalable research and practical deployment of LLMs as tool-using agents.

6.2 Related Work

6.2.1 Reinforcement Learning for Agentic Tool Use

The integration of reinforcement learning with tool use has emerged as a powerful paradigm for developing adaptive LLM agents. Early tool-calling approaches relied on prompt-based orchestration (Yao et al., 2023; Lu et al., 2023a; Shen et al., 2023), building on Chain-of-Thought reasoning (Wei et al., 2022b) and multi-agent frameworks for training-free tool invocation. While instruction-tuned models (Schick et al., 2023; Kong et al., 2023; Gou et al., 2024) learned structured calling patterns through supervised learning, they remained largely static and limited to single-turn interactions.

Recent work has demonstrated the advantages of reinforcement learning for tool use, enabling models to optimize their tool-calling policies based on execution outcomes and environmental feedback (Li et al., 2025c; Feng et al., 2026; Moshkov et al., 2025; Wang et al.,

2025a). This paradigm, which we refer to as *Agentic Reinforcement Learning with Tool use (ARLT)*, extends beyond single-turn verification to support long-horizon, multi-turn interactions. Key characteristics of ARLT include: (1) credit assignment across sequential tool calls, (2) explicit handling of observation tokens from tool responses, and (3) integration with robust, failure-aware execution environments (Plaat et al., 2025; Ke et al., 2025).

This shift from static instruction-following to dynamic, feedback-driven learning has shown effectiveness across diverse domains, including mathematical reasoning with code execution, information retrieval, natural language to SQL generation, and visual reasoning tasks. These applications require agents to probe environments iteratively, adapt to tool feedback, and refine their strategies—capabilities that are difficult to achieve through purely supervised approaches.

6.2.2 Agentic RL Training Frameworks

Table 6.1: Tool support comparison across different frameworks (up to update until August 23, 2025). RAGEN and ROLL focus on the puzzle environments like bandit, which we did not list here.

Framework	FAISS Search	Python Executor	Web Search	Bash Terminal	SQL Executor	Image Processing
OpenRLHF (Hu et al., 2024a)	✓	✓	×	×	×	×
VeRL (Sheng et al., 2025)	✓	✓	×	×	×	×
ROLL (Wang et al., 2025b)	×	×	×	×	×	×
RAGEN (Wang et al., 2025f)	×	×	×	×	×	×
Slime (THUDM, 2024)	✓	×	×	×	×	×
AReal (Fu et al., 2025)	✓	×	✓	×	×	×
SkyRL (Cao et al., 2025)	×	✓	×	✓	✓	×
VerlTool(ours)	✓	✓	✓	✓	✓	✓

The success of Reinforcement Learning from Verifier Rewards (RLVR) has motivated the development of various frameworks to support scalable RL training for language models. Established synchronous frameworks include OpenRLHF (Hu et al., 2024a) and VeRL (Sheng et al., 2025), which employ Ray-based distributed computing to manage training workflows. Additionally, fully asynchronous frameworks such as AReal (Fu et al., 2025), ROLL (Wang et al., 2025b), and Slime (THUDM, 2024) have emerged to address scalability challenges.

As shown in Table 6.1, existing frameworks exhibit varying degrees of tool support. Traditional RL frameworks such as OpenRLHF and VeRL provide basic support for

search and code execution tools but lack comprehensive multi-modal capabilities. ROLL focuses primarily on core RL training without extensive tool integration, while AReal supports search functionality but has limited executor capabilities. SkyRL (Cao et al., 2025) offers broader tool support, including bash terminals and SQL executors, but requires complex containerized environments that introduce deployment overhead. The limited tool coverage in existing frameworks has led to the development of domain-specific systems (e.g., Search-R1, PixelReasoner, and ToolRL) as task-specific extensions. However, these implementations typically feature hard-coded tool integrations that limit their extensibility and adaptability to new domains. As evident from Table 6.1, there remains a need for frameworks that provide comprehensive, extensible support for diverse tool types while maintaining ease of deployment and development.

6.3 VerlTool Framework

In this section, we formulate the conceptual foundation of the ARLT paradigm starting from the original RLVR setting. We then elaborate on how VerlTool serves as a practical implementation on the server side for **A**gentic **R**einforcement **L**earning with **T**ool use (**ARLT**).

The single-turn reward-optimization objective and GRPO are introduced in Section 2.2 (Equations (2.8) and (2.9)). The domain-specific rewards and training settings are given in Section B.4.1.

6.3.1 Agentic Reinforcement Learning with Tool Use

ARLT. In the agentic RL setting, rollouts are *multi-turn* instead of single-turn, and the agent can *interact with tools* to receive external observations during the reasoning process. Thus, the trajectory can be written as $\tau = \{a_0, o_0, \ldots, a_{n-1}, o_{n-1}, a_n\}$, where a_i denotes the LLM-generated action tokens and o_i denotes the observation tokens returned by a tool call. Here, n is the total number of interaction steps.

To determine whether an action a_i invokes a specific tool, we assume that each a_i (for $0 \leq i < n$) ends with a stop token $s \in \mathbb{S}_k$, where $\mathbb{S}_k$ is the predefined set of stop tokens for tool $T_k \in \mathbb{T}$. For example, $\mathbb{S}_{\text{CI}} = \{\texttt{```output}, \texttt{</python>}\}$ for a code interpreter tool, or $\mathbb{S}_{\text{search}} = \{\texttt{</search>}\}$ for a search tool. The complete set of stop tokens is the union over all invoked tools: $\mathbb{S} = \bigcup_{k=1}^{|\mathbb{T}|} \mathbb{S}_k$.

The introduction of observation tokens o_i makes ARLT fundamentally different from the agentic RL defined in RAGEN (Wang et al., 2025f), where the agent only receives scalar rewards through environmental interaction. Moreover, the observation tokens are off-policy with respect to the current LLM π_θ being optimized, which can destabilize training (Jin et al., 2025). Therefore, these tokens are typically masked out during policy optimization. Let $T_{i,j}$ be the token index of the first token in action segment $a_{i,j}$ of trajectory τ_i, which has n_i interaction steps, then the GRPO loss for ARLT becomes:

$$J_{\text{GRPO-ARLT}}(\theta) = \frac{1}{G}\sum_{i=1}^{G}\frac{1}{\sum_{j=0}^{n_i}|a_{i,j}|}\sum_{j=0}^{n_i}\sum_{t=T_{i,j}}^{T_{i,j}+|a_{i,j}|-1} \times \min\left[r_{i,t}(\theta)\cdot\hat{A}_{i,t},\ \text{clip}\left(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon\right)\cdot\hat{A}_{i,t}\right], \tag{6.1}$$

Here, G is the number of trajectories sampled for the same input x, and ϵ controls the clipping range. The token-level importance ratio is

$$r_{i,t}(\theta) = \frac{\pi_\theta(\tau_{i,(t)} \mid x, \tau_{i,<t})}{\pi_{\text{old}}(\tau_{i,(t)} \mid x, \tau_{i,<t})}, \tag{6.2}$$

where π_{old} is the policy used to sample the trajectories, $\tau_{i,(t)}$ is the current policy-generated token, and $\tau_{i,<t}$ includes preceding actions and tool observations. Each policy-generated token receives the trajectory-level advantage $\hat{A}_{i,t} = \hat{A}_i$, computed by Equation (2.9) with $R_i = R(x, \tau_i)$ denoting the total reward for trajectory τ_i.

6.3.2 Framework Design

Challenges. Building a general RL training framework that supports various tools is inherently challenging due to the additional overhead introduced by tool interactions. First, prior ARLT works are designed around a single tool and tightly couple the tool interaction logic with the core RL training loop, making it difficult for developers to extend or substitute tools (Li et al., 2025c). This fragmentation increases the development burden for researchers seeking to experiment with novel tools or multi-tool scenarios. Second, these systems often rely on synchronous rollout mechanisms that process trajectories batch by batch (Jin et al., 2025). In such settings, the tool interaction phase is triggered only after all actions a_i in a batch have been generated, resulting in idle bubbles and inefficient utilization of computational resources, especially on GPUs.

To address these issues, we propose VERLTOOL, a general-purpose ARLT framework designed to support various tools as modular plugins via a unified API. Our goal is to

minimize the integration overhead for community developers and provide a more efficient and extensible infrastructure for training LLMs with tool-use capabilities.

Overview. As shown in Figure 6.1, VERLTOOL adopts a modular and decoupled architecture consisting of two main components: the **VeRL Workflow** and the **Tool Server**, connected via a unified API. This separation enables independent management of RL training and tool execution while preserving efficient communication between them.

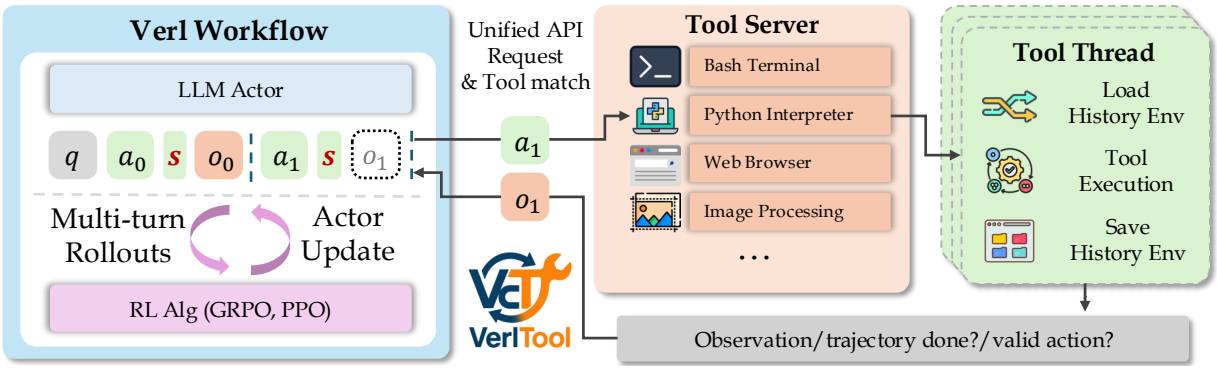


Figure 6.1: Overview of the VERLTOOL, a modularized and efficient framework for the **A**gentic **R**einforcement **L**earning with **T**ool Use (ARLT) training paradigm, where the RL workflow and tool execution are fully disaggregated for both efficiency and extensibility.

The **VeRL Workflow** handles all reinforcement learning activities, including multi-turn rollouts and actor updates. The LLM actor interacts with the external environment by generating a sequence of actions $\{a_0, a_1, \dots\}$, each potentially triggering a tool interaction. Once an action is identified as tool-invoking (via matching a predefined stop token), it is sent to the **Tool Server** along with auxiliary metadata. The observation o_i returned by the tool is then appended to the rollout, enabling observation-aware agent behavior and reward computation.

Asynchronous Rollout Design. A key feature of VERLTOOL is its support for fully *asynchronous rollouts*, which avoids the inefficiency of traditional synchronous batch-based frameworks. In such a setting, tool calls are processed only after the entire batch has completed generating their respective actions a_i, resulting in idle "bubbles" in GPU and CPU utilization. In contrast, VERLTOOL enables each trajectory to interact with the tool server independently and immediately upon finishing its action generation, as shown in Figure 6.2. This design ensures that tool execution latency does not block the entire

batch, significantly improving throughput and system utilization in large-scale distributed settings. As shown in Table 6.2, the actor and environment evolve concurrently, achieving near 2 times speedup for the rollout stage.

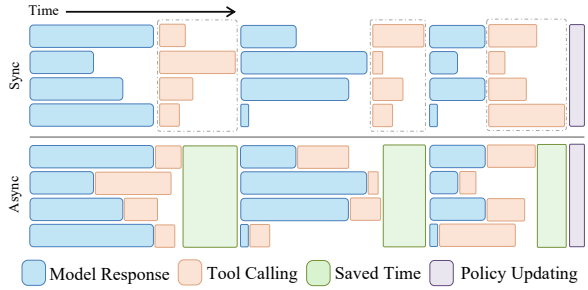


Figure 6.2: Visualization of the Async Rollout pipeline design and its effect in saving time.

Table 6.2: Performance comparison of Synchronous vs Asynchronous approaches. Experiments conducted on 8 H100 GPUs.

Metrics	Math-TIR	SQL	DeepSearch
Turns	4	5	5
Sync (s)	87	111	193
Async (s)	66	91	98
Speed Up (×)	**1.32**	**1.22**	**1.97**

Modular Tool-as-Plugin Design. As illustrated in Figure 6.3, VERLTOOL adopts a modular plugin system that cleanly abstracts tool interaction as an interface between the LLM actor and its external environment. Each tool is implemented as a subclass of a unified `BaseTool`, enabling seamless registration and extensibility. During rollouts, the actor's action a_i is parsed by `parse_action` to determine whether it invokes a tool; valid calls are routed to the appropriate module, which retrieves the trajectory state via `load_env`. The tool then executes its `conduct_action`, returning the observation o_i, a validity flag, and a termination flag for next action generation. We also maintain per-trajectory environments through lightweight state dictionaries, updated via `update_env` and cleared at the end of an episode with `delete_env`. By decoupling tool logic from the training workflow, developers can add new tools with minimal overhead, while the framework dynamically manages their execution across threads or distributed workers.

Tokenization. A practical challenge in multi-turn agentic RL is how to tokenize tool observations and concatenate them with preceding LLM actions. Two strategies exist: (i) tokenize the action and observation strings separately, then append their token sequences; or (ii) concatenate the raw strings first and tokenize jointly. Although they yield the same sequence most of the time, discrepancies may arise for some specific combinations, as illustrated in Figure 6.4. The action string "`</python>`" and observation "`\n<result>`" produce consistent tokens under the first strategy (*On-Policy*), whereas the second one



Figure 6.3: Example of code design for adding a new tool in VERLTOOL via the plugin interface.

merges boundary symbols into a different token id (e.g., `29,198` vs. `397`), which changed the LLM-generated contents (*Off-Policy*). To avoid such inconsistencies, we adopt the first approach and always keep a consistent token list prefix during rollout, ensuring stable alignment in multiple rollouts turns.

Parallel Tool Server Backend. To support high-throughput and scalable execution of tool interactions, VERLTOOL offers two types of parallel execution backends within the Tool Server:

- *Multi-threading*: For small-scale or lightweight tool calls, VERLTOOL employs Python's `ThreadPoolExecutor` to parallelize calls across multiple worker threads.

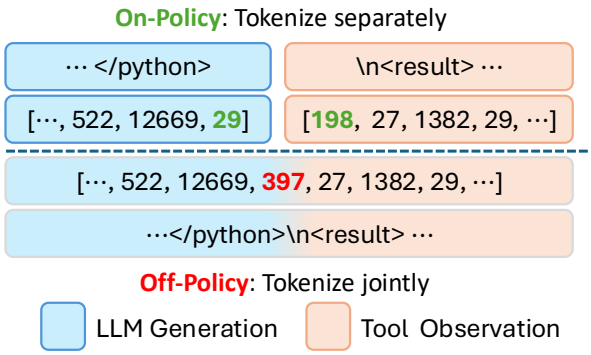


Figure 6.4: Tokenization of LLM generated content "...</python>" and tool observation "\n<result>..." can produce different token lists using Qwen2.5 tokenizer under different strategies.

- *Ray-based Asynchronous Execution*: To deal with resource-intensive tools for better resource management, VERLTOOL optionally supports integration with Ray (Moritz et al., 2018), enabling distributed and fault-tolerant tool execution across machines or GPU nodes. This design provides robust scalability for long-horizon or computationally intensive tools.

6.4 Experiments

6.4.1 Experiment Setup

With a modular, plug-and-play design, VERLTOOL equips an agent with tools spanning multiple domains and modalities as shown in Table B.20. In this section, we show the experiment results in six agentic RL with tool use (ARLT) tasks, including VT-Math (Table 6.3), VT-Search (Table 6.4), VT-SQL (Table 6.5), VT-VisualReasoner (Table 6.6), VT-DeepSearch (Table 6.6), and VT-SWE (Table 6.5), demonstrating the compatibility of VERLTOOL with various tools. Please see details of training, evaluation in subsection B.4.1.

6.4.2 Results

Training on VERLTOOL achieves competitive results. Models trained using VERLTOOL consistently match or exceed existing baselines across all six tasks. VT-Math

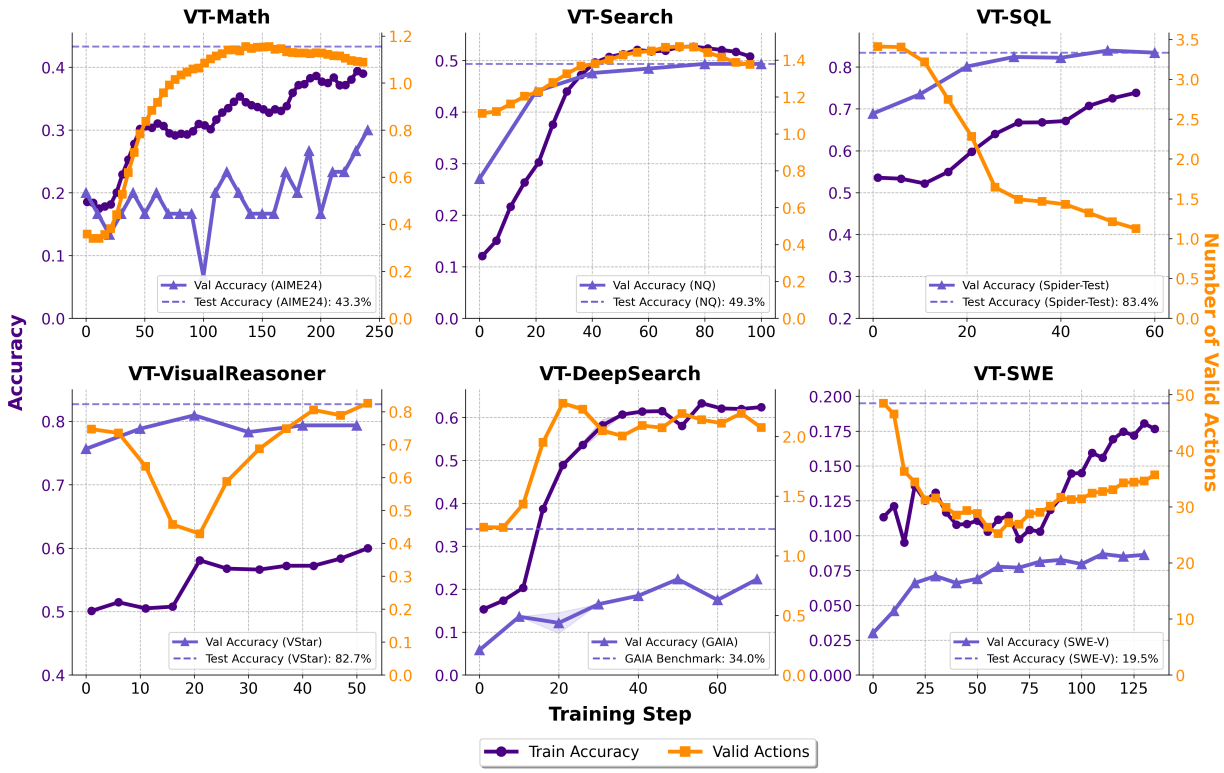


Figure 6.5: Training dynamics using VERLTOOL on all 6 tasks. For each task, the corresponding test benchmarks are AIME24, NQ, Spider-Test, VStar, GAIA, and SWE-Verified. All models are trained and evaluated based on VERLTOOL framework. The actual evaluation performance (purple dash) can be higher due to the train-eval settings difference. The number of actions is averaged over all sampled responses in each batch.

Table 6.3: Results on Math-related benchmarks with Python interpreter tool. The best results are indicated in **bold** and the second-best results are underlined.

Model	GSM8K	MATH 500	Minerva Math	Olympiad Bench	AIME24	AMC23	Avg.
Qwen2.5-Math-7B-Base/Instruct							
Qwen2.5-Math-7B-Instruct	**95.2**	83.0	**37.1**	41.6	16.7	70.0	57.3
Qwen-2.5-Math-7B-Instruct-TIR	88.8	80.2	26.8	41.6	30.0	52.5	53.3
SimpleRL-Zoo-7B	94.6	82.4	29.0	50.5	30.0	62.5	58.2
ToRL-7B	92.7	82.2	33.5	49.9	43.3	65.0	61.1
VT-Math-zero (GRPO)	91.8	**83.2**	31.6	50.5	**43.3**	70.0	61.7
VT-Math-zero (DAPO)	92.1	82.8	34.9	**51.6**	36.7	**75.0**	**62.2**

Table 6.4: Results of on knowledge-QA benchmarks. †/⋆ represents in-domain/out-domain datasets.

Model	General QA			Multi-Hop QA				Avg.
	NQ†	TriviaQA⋆	PopQA⋆	HotpotQA†	2wiki⋆	Musique⋆	Bamboogle⋆	Avg.
Qwen2.5-3b-Base/Instruct								
Direct Inference	10.6	28.8	10.8	14.9	24.4	2.0	2.4	13.4
Search-R1-base (GRPO)	42.1	58.3	41.3	29.7	27.4	6.6	12.8	31.2
Search-R1-base (PPO)	40.6	58.7	43.5	28.4	27.3	4.9	8.8	30.3
VT-Search-zero (GRPO)	45.4	61.6	**48.1**	32.4	30.8	7.6	15.2	34.4
VT-Search-zero (DAPO)	**45.8**	**62.3**	46.5	**33.0**	**31.1**	**8.2**	15.2	**34.6**
Qwen2.5-7b-Base/Instruct								
Direct Inference	13.4	40.8	14.0	18.3	25.0	3.1	12.0	18.1
Search-R1-base (GRPO)	39.5	56.0	38.8	32.6	29.7	12.5	36.0	35.0
Search-R1-base (PPO)	48.0	63.8	45.7	43.3	38.2	**19.6**	43.2	43.1
VT-Search-zero (GRPO)	**49.3**	**66.2**	**50.2**	**44.8**	**45.3**	19.3	**46.4**	**45.9**
VT-Search-zero (DAPO)	48.3	63.4	48.2	42.6	39.2	18.0	38.4	41.2

Table 6.5: Results on NL2SQL (left) and SWE-Verified (right) benchmarks in terms of pass rates.

Model	Spider	
Split	Dev	Realistic
Reasoning without Tool		
GPT-4o	70.9	-
DeepSeekCoder-6.7B-Instruct	63.2	-
OpenCoder-8B-Instruct	59.5	-
Qwen2.5-Coder-7B-Instruct	73.4	-
Tool Integrated Reasoning		
OmniSQL-7B	81.2	63.9
SkyRL-SQL-7B (GRPO)	**83.9**	81.1
VT-SQL (Qwen-2.5-Coder-7B-Instruct based)		
+ GRPO	**83.9**	**81.3**

Model	SWEBench
OpenHands Scaffold	
Qwen3-8B	3.6
OpenHands-7B-Agent	11.0
SkyRL-v0 (Cao et al., 2025)	
Qwen3-8B Based	9.4
OpenHands-7B-Agent Based	14.6
R2E Gym Scaffold	
Qwen3-8B	10.4
VT-SWE (Qwen3-8B Based)	
+GRPO	**19.5**

Table 6.6: Results on visual reasoning (left) and agentic search benchmarks (right).

Model	V* Bench
Reasoning without Tool	
GPT-4o	62.8
Gemini-2.5-Pro	79.2
Qwen2.5-VL-7B-Instruct	70.4
Video-R1-7B	51.2
Tool Integrated Reasoning	
Visual Sketchpad (GPT-4o)	80.4
IVM-Enhanced (GPT-4V)	81.2
Pixel-Reasoner-7B	**84.3**
VT-VisualReasoner (Qwen2.5-VL-7B-Instruct Based)	
+ GRPO-Acc	78.8
+ GRPO-Complex	<u>82.7</u>

Model	GAIA	HLE
Reasoning without Tool		
DeepSeek-R1-671B	<u>25.2</u>	**8.6**
GPT-4o	17.5	2.6
Qwen3-8B	20.4	4.6
Tool Integrated Reasoning (Qwen3-8B)		
Vanilla RAG	20.4	5.8
Search-o1	21.4	6.4
WebThinker	22.3	6.6
ReAct	23.3	4.6
VT-DeepSearch (Qwen3-8B Based)		
+ Snippets-Only	32.0	7.8
+ QwQ-32B	**34.0**	<u>8.4</u>

achieves a 62.2% average performance on mathematical benchmarks, surpassing expert models on multiple benchmarks such as AIME24, AMC23, and Olympiad Bench. VT-Search reaches 45.9% accuracy on knowledge QA, surpassing Search-R1 by 10.9%. In terms of the NL2SQL task, VT-SQL matches specialized systems such as SkyRL-SQL. VT-VisualReasoner achieves 82.7% on V* Bench while VT-DeepSearch reaches 34.0% on GAIA. These figures demonstrate that trained under our unified framework, agents could achieve competitive task-specific performance compared to separate divergent code bases.

Multi-modal tools are well-supported. VerlTool's modular design enables the seamless integration of a wide range of types of tools within a unified API interface. We demonstrated this capability through the implementation of text-based tools support including Python and SQL Interpreter, Local Retriever and Web Search Tools, as well as visual processing tools that operate on image and video modality (image operations and video frame selections). Moreover, our framework is equipped with system-level tools such as the bash terminal and file operation tools. Visual reasoning experiments demonstrate that our agents could dynamically manipulate images and process visual information iteratively, enabling complex multi-modal workflows that were unsupported in existing single-modality RL-training frameworks, which only work on the text level.

Dynamics of Tool Usage across Tasks. Tool usage patterns exhibit substantial variation across different domains, with mathematical tasks typically requiring $1 \sim 4$ interactions while software engineering tasks may extend to over 100 interactions. Importantly, models do not spontaneously develop effective tool-use capabilities without appropriate reward design and initialization strategies. For instance, without VT-VisualReasoner's sophisticated reward mechanism, the frequency of tool actions gradually diminishes to zero within a few reinforcement learning (RL) steps.

The evolution of tool usage during training demonstrates task-specific characteristics that reflect the underlying utility of tool interactions. In VT-SQL settings, the number of actions decreases rapidly after several dozen RL steps, as the model learns that SQL executors are non-essential for most straightforward queries. Through training, the model develops a preference for responses requiring fewer tool calls by gradually memorizing expected execution results, thereby reducing the need for verification through SQL executors. Conversely, in VT-DeepSearch settings, tool usage increases dramatically during training because problem-solving fundamentally depends on search capabilities. Unlike simpler SQL results, the extensive and information-rich content returned by search tools cannot be easily memorized, necessitating increased tool invocations for effective task completion. This divergent behavior underscores how the intrinsic value of tool assistance shapes learning dynamics across different computational domains.

Emerging behaviours of Agentic RL. Agentic models trained using VerlTool emerged sophisticated behaviors, including self-correction, iterative refinement, and strategic tool selection in the multi-round tool-calling environment provided by our framework. For instance, mathematical agents verify computations and backtrack from errors, search agents refine queries based on retrieved information, and software agents develop debugging strategies combining code analysis and incremental fixes. These capabilities represent genuine agentic problem-solving that extends beyond simple function-calling invocations. Case studies are given in the published version of this work (Jiang et al., 2026).

6.5 Feedback Design

The experiments above establish that VerlTool supports training across different tool-use tasks. I now examine two choices within this setting, whether auxiliary rewards improve task performance as well as tool use, and how the amount of information returned by a tool affects performance. The reward definitions and main task results are given in Section B.4.1 and Section 6.4, and I focus here on the additional comparisons.

6.5.1 Auxiliary Rewards for Tool Use

The main visual-reasoning experiment provides a positive example, in which GRPO-Complex, which includes auxiliary reward terms, achieves 82.7% on V* Bench, compared with 78.8% for GRPO-Acc (Table 6.6).

I compare three reward settings in a small-scale VT-Math experiment with Qwen2.5-Math-1.5B over 100 GRPO steps. All use the same answer-match reward of +1 for a correct answer and −1 otherwise. The first setting uses this reward alone. The second subtracts 0.25 from incorrect responses that make no valid interpreter call or encounter an execution error. The third adds 0.1 for each valid tool call. These are additional reward variants that do not replace the reward definitions of the main VT-Math and VT-DeepSearch experiments.

Figure 6.6 compares tool calls, execution errors, and held-out MATH-500 accuracy. Both auxiliary terms increase tool calls relative to the outcome reward alone, but neither gives higher accuracy at the final checkpoint. The per-call bonus run drops sharply at its last evaluation. With one run per setting, this comparison does not establish a consistent accuracy effect or identify the cause of that drop. It illustrates why more tool calls should be evaluated alongside task performance before being treated as an improvement in tool use.

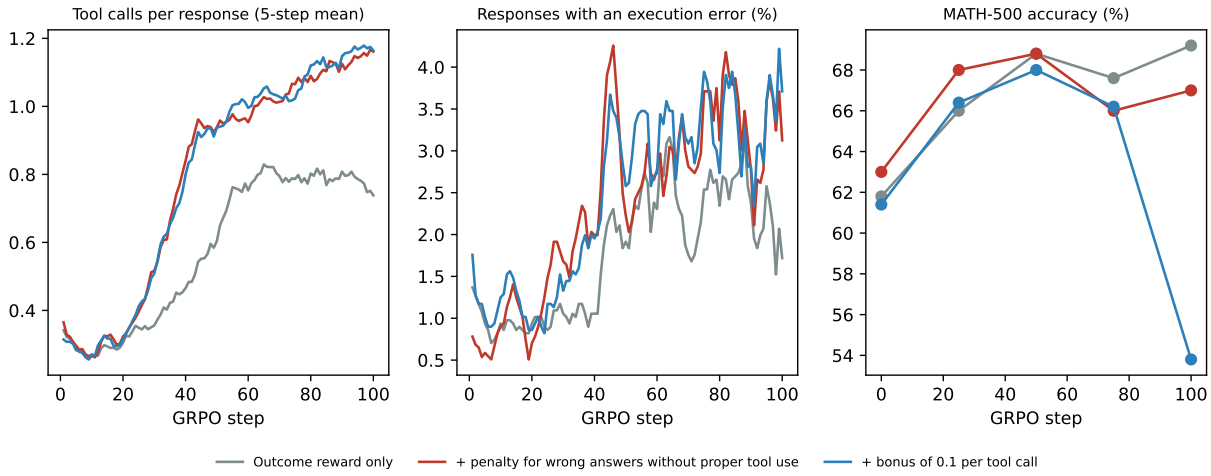


Figure 6.6: Auxiliary reward comparison in VT-Math with Qwen2.5-Math-1.5B over 100 GRPO steps (one run per setting). Left: tool calls per response, using a five-step mean. Middle: responses with an execution error. Right: held-out MATH-500 accuracy.

6.5.2 Observation Length

The observation cap limits how many tokens a tool response contributes to the policy's context. In a supplementary VT-Search experiment conducted during review of VerlTool, we compare caps of 512, 1024, 2048, and 4096 tokens (Table 6.7). This experiment uses a configuration separate from the main retrieval results in Table 6.4.

Increasing the cap from 512 to 1024 tokens improves exact match on all seven datasets, and NQ rises from 23.4% to 47.5%, and PopQA from 17.9% to 46.9%. Increasing it further gives mixed changes, with no cap above 1024 performing best on every dataset. This comparison shows that the observation budget can materially affect performance. It does not identify which passages account for the changes or establish a preferred cap for other tools and tasks.

Table 6.7: Exact match (%) at four maximum tool-response lengths, in tokens. Results are from the review experiments of VerlTool, with a configuration separate from Table 6.4.

Cap	NQ	TriviaQA	PopQA	HotpotQA	2Wiki	Musique	Bamboogle
512	23.4	44.5	17.9	20.6	25.9	4.3	8.8
1024	47.5	63.4	46.9	35.1	32.4	9.0	23.2
2048	47.3	63.7	48.5	34.8	33.1	8.7	20.0
4096	46.9	62.8	48.8	35.2	33.4	9.3	19.2

Supplementary analyses of tokenization and trajectory masking from the review experiments are collected in Appendix C.1 and Appendix C.2, respectively.

6.6 Summary

This chapter developed VerlTool, a framework for reinforcement learning with multi-turn tool use. It separates the reinforcement learning workflow from a tool server and supports stateful interactions with text and multimodal tools. Experiments across six domains show that agents trained within this framework achieve competitive results against specialized systems, for example 62.2 average accuracy on the mathematics benchmarks and 19.5 on SWE-Bench Verified. This extends Chapter 5 from using execution results to assess completed programs to learning through sequences of actions and tool observations.

The additional feedback-design experiments examine auxiliary rewards and the amount of information returned by tools. In the main visual-reasoning run the curiosity bonus kept

image operations and raised accuracy, whereas in the small-scale mathematics comparison two other auxiliary terms increased tool calls without improving final MATH-500 accuracy in the runs shown. In the retrieval experiment, increasing the observation cap from 512 to 1024 tokens improved exact match across the evaluated datasets, while larger caps gave mixed results. These observations distinguish the behavior encouraged by a reward from task performance, and show that how much of a tool's output the policy reads can change exact match by more than twenty points.

These comparisons cover specific tasks and configurations, so they do not establish a general rule for reward shaping or observation length. The training objective also assigns a trajectory-level advantage to policy-generated tokens, leaving finer credit assignment across tool calls open. Together, Chapters 5 and 6 show how execution and interaction can provide feedback for model improvement, while illustrating why the effects of that feedback need to be checked against task performance.

Chapter 7

Discussion

The preceding chapters show how feedback can support evaluation and model improvement, while also revealing limits in what it measures. A learned evaluator may score a refusal as free of errors, and a reward for tool use may increase calls without improving task performance. This chapter discusses how these limitations affect the use of feedback during training. I then examine how subsequent work builds on these studies and identify directions for future research.

7.1 Reward Hacking

Reward hacking, or specification gaming, occurs when a policy learns to obtain reward through behavior that does not achieve the intended task objective (Amodei et al., 2016; Skalse et al., 2022). I discuss this risk in learned rewards and environment rewards, focusing on how the behavior encouraged by a reward relates to performance on independent task evaluations.

Learned rewards can favor responses that avoid the task. In the small-scale experiment of Section 3.5.2, I use TIGERScore as a reward for GRPO training. A response receives the highest score when the evaluator detects no errors. During training, the policy increasingly produces refusals and disclaimers, its penalties approach zero, and its GSM8K accuracy and independent judge preference decline. These responses contain few substantive claims for the evaluator to mark as wrong, allowing the policy to avoid deductions without answering the question. In this setting, optimizing the evaluator exposes a gap between avoiding detected errors and completing the task.

Environment rewards can miss the purpose of tool use. A reward can also be computed as defined while encouraging behavior that falls short of the intended use of tools. In VT-VisualReasoner, the accuracy reward credits a correct answer whether or not the policy uses image operations. Under this reward alone, the policy stops cropping and reaches 78.8% on V* Bench, compared with 82.7% for GRPO-Complex, which includes auxiliary reward terms (Table 6.6). Directly rewarding tool use can create a different mismatch. In the small-scale VT-Math comparison, a per-call bonus and a no-call-or-error penalty increase tool calls, but neither improves final MATH-500 accuracy over the outcome reward alone (Section 6.5.1). These patterns suggest possible reward hacking, where satisfying an answer check or collecting a tool-use reward does not ensure that the policy learns more effective tool use. Examining tool-use behavior alongside held-out task accuracy helps identify this divergence.

7.2 Subsequent Work

Nine papers by my collaborators and me build on Chapters 3–6 (Table 7.1). Their results are not contributions of this thesis, and the last column gives my role in each.

Learned evaluators. Chapter 4 found that prompted multimodal judges reproduce human votes poorly. VideoScore (He et al., 2024b) tests whether the recipe of Chapter 3 fixes this for video. Trained on human ratings of 37.6K videos, it reached 78.5% pairwise accuracy on the video portion of GenAI-Bench, 11.4 points above the best prompted baseline, under a protocol that differs from the four-way exact match of Chapter 4. VideoScore2 (He et al., 2026a) adds an explanation before the score, with rationales elicited from Claude-4-Sonnet, so their faithfulness is open. Neither evaluator was used as a reward for training a generator. StructEval (Yang et al., 2026) evaluates structured outputs across 18 formats using parsing, path, and visual checks. Its human evaluation also identifies limits of visual assessment. Some annotated requirements, such as precise numeric values or interactive elements, cannot be judged from the rendered image, accounting for about 11% of sampled checks.

Execution feedback. EvolveCoder (Ruan et al., 2026b) refines synthesized tests using candidate programs' pass and fail patterns and removes tests that do not distinguish among candidates. With Qwen3-4B, data from rounds 0 to 3 gave averages of 46.6, 47.2, 48.5, and 49.0 over four code benchmarks. Data size also changed with each round, so the comparison

Table 7.1: Subsequent work by my collaborators and me, grouped by the chapter it extends (status as of October 2026). Results of arXiv preprints should be read with that caveat.

Work	Extends	Status	My role
VideoScore (He et al., 2024b)	Ch. 3, 4	EMNLP 2024	Co-first author
VideoScore2 (He et al., 2026a)	Ch. 3, 4	TMLR 2026	Co-first author
StructEval (Yang et al., 2026)	Ch. 3	TMLR 2026	Co-first author
EvolveCoder (Ruan et al., 2026b)	Ch. 5	arXiv preprint	Second author
General-Reasoner (Ma et al., 2025)	Ch. 5	NeurIPS 2025	Third author
Critique-Coder (Ruan et al., 2026a)	Ch. 5	ICLR 2026	Second author
OpenResearcher (Li et al., 2026a)	Ch. 6	EMNLP 2026	Co-lead, second author
DCI (Li et al., 2026b)	Ch. 6	NeurIPS 2026	Idea, writing, corresponding author
Dr-DCI (Lu et al., 2026b)	Ch. 6	arXiv preprint	Advising, corresponding author

does not isolate the effect of test generation. General-Reasoner (Ma et al., 2025) carries verification past code to short answers that rule-based checkers often misjudge, using a 1.5B generative verifier. On 50K answers that Gemini-2.0-Flash judged correct, it agreed on 78.7% against 22.2% for a rule-based verifier. Early policies exploited it by putting long content in the answer box, which a length penalty now counters. Critique-Coder (Ruan et al., 2026a) lets learned feedback borrow the reliability of execution, since it rewards critiques whose verdict matches an execution label, and replacing 20% of standard RL data with these tasks raised the 8B average from 49.8 to 51.5, although self-critique gave no gain at test time.

Environments and interfaces for agents. OpenResearcher (Li et al., 2026a) addresses the reproducibility limit of Chapter 6, live web search (Section 6.6), with an offline corpus that merges bootstrapped gold documents for 6K questions with 15M FineWeb documents. A fixed corpus makes search rewards repeatable across runs. A model fine-tuned on about 55K correct teacher trajectories reached 54.8% on BrowseComp-Plus, and removing the gold documents cut accuracy to 6.35%. DCI (Li et al., 2026b) and Dr-DCI (Lu et al., 2026b) study the tool interface, which VerlTool exposes as a plugin and which decides what observations the agent receives. Replacing a top-k retriever with terminal tools such as `grep` over the raw corpus raised Claude Sonnet 4.6 on BrowseComp-Plus from 69.0% to 80.0%, and Dr-DCI remained effective from 100K to 10M documents, and both change the interface at inference time only.

Adoption and follow-up by others. I report only uses verified in the cited papers or public records on October 14, 2026. UnifiedReward (Wang et al., 2025e) evaluates on the image and video subsets of GenAI-Bench, the reference-data role that Chapter 4 proposed. AceCode-87K has been used as reinforcement learning data by DiffuCoder (Gong et al., 2026), wd1 (Tang et al., 2026), and Cui et al. (2025). Olmo 3 (Team Olmo, 2025) sources the code prompts of its Dolci Think post-training data from AceCoder, generating completions with QwQ-32B and filtering them with synthesized tests, the same recipe as Chapter 5 with a different test generator. HardTestGen (He et al., 2026c) measured how often synthesized tests accept wrong programs, finding that tests whose inputs are all written by an LLM, as in our recipe, reached an average precision of 42.42% on programs from Qwen2.5-Coder-7B-Instruct over 1,253 AtCoder and Codeforces problems, against 79.21% once inputs from generator programs and hacking tests were added, with the gap largest on the hardest problems. CodeScaler (Zhu et al., 2026) reported 37.01 with execution-based rewards against 33.19 with AceCodeRM-7B, the same direction as Section 5.4.5. AdaTooler-V (Wang et al., 2026), Lu et al. (2026a), and Zhang et al. (2026) used VerlTool as a training framework, the last to train a tool-using verifier, which combines learned and environment feedback.

7.3 Future Directions

Aligning rubric-based judges with human judgments. A future direction is to improve rubric-based LLM judges using human judgments before using their scores as training rewards. Human comparisons could be used to train the judge, with its agreement with people measured on held-out examples. The resulting judge could then provide rewards for policy training, while human evaluations assess the outputs produced during optimization. This would test whether improving the judge's agreement with people leads to better training outcomes, and whether that agreement holds as the policy changes.

Correct and readable code. The test-pass reward of Chapter 5 asks only whether a program is correct, and a policy optimized against tests alone has no reason to keep its code readable. Human preference is the natural source for that property, and it is affordable here because the comparisons can be restricted to programs that already pass the tests. One design trains a preference model on human comparisons of correct programs, judged on readability, naming, and structure, and adds it to the pass-rate reward. The question is whether the resulting policy keeps the pass rates of the test-only policy while producing programs that people rate as more readable.

Tests that follow the policy's failures. The tests of Chapter 5 were generated once and fixed, and for a reward the quality that matters is how often they accept a wrong program the policy actually writes, which HardTestGen (He et al., 2026c) measures as precision over a policy's own programs. The direction is to make the tests adaptive, finding the wrong programs the current policy produces and adding tests that reject them, by mutation testing (Jia and Harman, 2011) or by writing tests from the pass and fail patterns of sampled programs as EvolveCoder (Ruan et al., 2026b) does. The experiment is a matched-compute comparison of fixed and adaptive tests as the RL reward, with the precision of each suite over the policy's samples tracked during training.

Chapter 8

Conclusion

This thesis studied how feedback can be obtained for evaluating generated outputs and used to improve generative models. It examined learned evaluators, human judgments, program execution, and interaction with environments through two research questions: how to obtain assessments that agree with human judgments and provide understandable explanations, and how to design feedback that remains reliable while a model is optimized against it.

8.1 Evaluating Generated Outputs

The first research question asked how feedback on generated outputs can be obtained that agrees with human judgment and provides explanations for its assessments.

For text, Chapter 3 developed TIGERScore, a reference-free evaluator trained on error analyses to identify and explain errors across text generation tasks. Its scores achieved higher average Kendall correlation with human ratings than the compared metrics other than GPT-4 used as a judge. Human annotators also assessed its analyses, rating 70.9% of them 3 or higher out of 5. These results show that scores and explanations can be learned across the tasks studied and evaluated against human judgments.

For images and videos, Chapter 4 developed GenAI-Arena, an open platform that collects anonymous pairwise comparisons for text-to-image generation, image editing, and text-to-video generation. The collected preferences support model rankings, and the cleaned votes are released as GenAI-Bench to evaluate automatic judges. The best multimodal judge tested agrees with human votes about half the time. Human preferences

therefore remain important in these settings, both for ranking generators and for assessing automatic judges.

8.2 Improving Generative Models

The second research question asked how feedback can be designed to remain reliable while a model is optimized against it.

Chapter 5 developed AceCoder to obtain feedback for code generation through program execution. It synthesizes tests for coding questions, turns sampled programs' pass rates into preference pairs, and trains reward models to select among candidate programs. Best-of-64 selection with the 7B reward model raises Llama-3.1-8B-Instruct's average score from 40.9 to 49.3 across the evaluated benchmarks. Test results also provide rewards directly for reinforcement learning: Qwen2.5-Coder-7B-Base improves from 53.0 to 78.0 on HumanEval+ after 80 optimization steps.

Chapter 6 developed VerlTool, a framework that supports reinforcement learning over multi-turn tool interactions through a unified tool server and asynchronous rollouts. Agents trained in six domains achieve competitive results against specialized systems. The additional feedback-design experiments examine auxiliary rewards for tool use and the amount of information returned by tools. In the small-scale mathematics comparison, auxiliary rewards increase tool calls without improving final task accuracy in the runs shown. The retrieval comparison shows that the amount of a tool's output available to the policy can change task performance. These comparisons concern the tasks and configurations tested and do not establish a general rule for auxiliary rewards or observation length.

Together, these chapters show how execution results and interaction with environments can provide task-specific feedback for selection and learning. Their experiments assess this feedback through its effects on candidate selection and task performance during learning.

8.3 How the Two Parts Connect

Part I studies feedback for assessing generated outputs, using human judgments to evaluate learned scores and explanations and to rank visual generators. Part II studies feedback for selecting outputs and training models, using execution results and environment interactions. The additional TIGERScore experiments connect these uses. As a Best-of-8 selector, it gives only a small improvement over random selection on GSM8K. In the small-scale

training experiment, the policy increasingly produces refusals that receive favorable scores while independent task evaluations decline (Section 3.5.2). These observations identify limits of this evaluator in the selection and training settings tested, alongside its agreement with human ratings in evaluation.

8.4 Closing Remarks

The broader aim of this thesis is to make feedback useful both for understanding generated outputs and for improving the models that produce them. The qualities that matter vary across tasks, from correctness in code to human preferences in visual generation. Learned assessments, human judgments, execution results, and tool observations offer different ways to obtain information about these qualities. Studying them together provides a basis for choosing and evaluating feedback according to the task and its intended use.

Looking ahead, the directions in Section 7.3 explore how these sources might work together during training. Aligning rubric-based judges with human judgments, combining execution feedback with preferences for readable code, and adapting tests to the policy's emerging failures could help connect the feedback used for learning more closely to the quality of the resulting outputs. I hope this thesis helps guide the development of feedback that people can understand and assess, and that enables generative models to improve on the tasks people want them to perform.

References

Mistral AI. Mistral-7b-instruct-v0.3. https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3, 2023.

Nadia Alshahwan et al. Automated unit test improvement using large language models at Meta. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion)*, pages 185–196. ACM, 2024.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. arXiv preprint arXiv:1606.06565, 2016.

Jacob Austin et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Jinze Bai et al. Qwen-vl: A frontier large vision-language model with versatile abilities. *ArXiv*, abs/2308.12966, 2023.

Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5): 507–525, 2015.

Rohan Bavishi et al. Introducing our multimodal models, 2023. URL https://www.adept.ai/blog/fuyu-8b.

Black Forest Labs. FLUX, 2024. URL https://github.com/black-forest-labs/flux.

Andreas Blattmann et al. Stable Video Diffusion: Scaling latent video diffusion models to large datasets. arXiv preprint arXiv:2311.15127, 2023.

Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Tim Brooks, Aleksander Holynski, and Alexei A. Efros. InstructPix2Pix: Learning to follow image editing instructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18392–18402. IEEE, 2023.

Tom Brown et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

Zheng Cai et al. InternLM2 technical report. arXiv preprint arXiv:2403.17297, 2024.

Chris Callison-Burch, Miles Osborne, and Philipp Koehn. Re-evaluating the role of Bleu in ma-

chine translation research. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 249–256, Trento, Italy, April 2006. Association for Computational Linguistics.

Shiyi Cao et al. Skyrl-v0: Train real-world long-horizon agents via reinforcement learning, 2025.

Bei Chen et al. CodeT: Code generation with generated tests. In *International Conference on Learning Representations*, 2023.

Haoxin Chen et al. VideoCrafter2: Overcoming data limitations for high-quality video diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7310–7320, 2024a.

Junsong Chen et al. PixArt-σ: Weak-to-strong training of diffusion transformer for 4K text-to-image generation. In *Computer Vision – ECCV 2024*, Lecture Notes in Computer Science, pages 74–91. Springer, 2024b.

Junsong Chen et al. PixArt-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. In *International Conference on Learning Representations*, 2024c.

Mark Chen et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021.

Xingyu Chen et al. Do not think that much for 2+3=? on the overthinking of o1-like llms. *ArXiv*, abs/2412.21187, 2024d.

Wei-Lin Chiang et al. Chatbot Arena: An open platform for evaluating LLMs by human preference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, 2024.

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*, volume 30. Curran Associates, Inc., 2017.

Hyung Won Chung et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

Karl Cobbe et al. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021.

Foundational Contributors et al. Openai o1 system card. *ArXiv*, abs/2412.16720, 2024.

Ganqu Cui et al. UltraFeedback: Boosting language models with scaled AI feedback. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 9722–9744, 2024.

Ganqu Cui et al. The entropy mechanism of reinforcement learning for reasoning language models, 2025. arXiv preprint arXiv:2505.22617.

Wenliang Dai et al. InstructBLIP: Towards general-purpose vision-language models with instruction tuning. In *Advances in Neural Information Processing Systems*, volume 36, pages 49250–49267, 2023.

Martin D. Davis and Elaine J. Weyuker. Pseudo-oracles for non-testable programs. In *Proceedings of the ACM '81 Conference*, pages 254–257, 1981.

Guanting Dong et al. Agentic reinforced policy optimization. In *International Conference on Learning Representations*, 2026.

Hanze Dong et al. RAFT: Reward rAnked FineTuning for generative foundation model alignment. *Transactions on Machine Learning Research*, 2023.

Shihan Dou et al. StepCoder: Improving code generation with reinforcement learning from compiler feedback. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4571–4585, 2024.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. Length-controlled AlpacaEval: A simple debiasing of automatic evaluators. In *First Conference on Language Modeling*, 2024.

Desmond Elliott and Frank Keller. Comparing automatic evaluation measures for image description. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 452–457, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

Arpad E. Elo. *The Rating of Chessplayers, Past and Present*. Arco Publishing, New York, 1978. ISBN 0-668-04721-6.

Alexander R Fabbri, Wojciech Kryściński, Bryan McCann, Caiming Xiong, Richard Socher, and Dragomir Radev. SummEval: Re-evaluating summarization evaluation. *Transactions of the Association for Computational Linguistics*, 9:391–409, 2021.

fal.ai. AuraFlow, 2024. URL https://huggingface.co/fal/AuraFlow.

Angela Fan, Yacine Jernite, Ethan Perez, David Grangier, Jason Weston, and Michael Auli. Eli5: Long form question answering. *arXiv preprint arXiv:1907.09190*, 2019.

Jiazhan Feng et al. ReTool: Reinforcement learning for strategic tool use in LLMs. In *International Conference on Learning Representations*, 2026.

Gordon Fraser and Andrea Arcuri. EvoSuite: Automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE)*, pages 416–419, 2011.

Markus Freitag, George Foster, David Grangier, Viresh Ratnakar, Qijun Tan, and Wolfgang Macherey. Experts, errors, and context: A large-scale study of human evaluation for machine translation. *Transactions of the Association for Computational Linguistics*, 9:1460–1474, 2021.

Markus Freitag et al. Results of WMT22 metrics shared task: Stop using BLEU – neural metrics are better and more robust. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 46–68, Abu Dhabi, United Arab Emirates (Hybrid), December 2022. Association for Computational Linguistics.

Jinlan Fu, See-Kiong Ng, Zhengbao Jiang, and Pengfei Liu. Gptscore: Evaluate as you desire, 2023.

Stephanie Fu et al. Dreamsim: Learning new dimensions of human visual similarity using synthetic data. *Advances in Neural Information Processing Systems*, 36, 2024.

Wei Fu et al. AReaL: A large-scale asynchronous reinforcement learning system for language reasoning. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Yujian Gan et al. Towards robustness of text-to-sql models against synonym substitution. *ArXiv*, abs/2106.01065, 2021a.

Yujian Gan, Xinyun Chen, and Matthew Purver. Exploring underexplored limitations of cross-domain text-to-sql generalization, 2021b.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10835–10866, 2023.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. RLEF: Grounding code LLMs in execution feedback with reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 19034–19055, 2025.

Sara Ghazanfari, Alexandre Araujo, Prashanth Krishnamurthy, Farshad Khorrami, and Siddharth Garg. Lipsim: A provably robust perceptual similarity metric. In *The Twelfth International Conference on Learning Representations*, 2023.

Shansan Gong et al. DiffuCoder: Understanding and improving masked diffusion models for code generation. In *International Conference on Learning Representations*, 2026.

Zhibin Gou et al. ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *International Conference on Learning Representations*, 2024.

Aaron Grattafiori et al. The Llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024.

Jian Guan et al. Openmeva: A benchmark for evaluating open-ended story generation metrics. *arXiv preprint arXiv:2105.08920*, 2021.

Daya Guo et al. DeepSeek-Coder: When the large language model meets programming – the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024a.

Daya Guo et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Yuwei Guo et al. AnimateDiff: Animate your personalized text-to-image diffusion models without specific tuning. In *International Conference on Learning Representations*, 2024b.

Chaoqun He et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Annual Meeting of the Association for Computational Linguistics*, 2024a.

Xuan He et al. VideoScore: Building automatic metrics to simulate fine-grained human feedback for video generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2105–2123, 2024b.

Xuan He et al. VideoScore2: Think before you score in generative video evaluation. *Transactions on Machine Learning Research*, 2026a.

Zhiwei He et al. DeepMath-103K: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. In *International Conference on Learning Representations*, 2026b.

Zhongmou He et al. HardTestGen: A high-quality RL verifier generation pipeline for LLM algorithmic coding. In *International Conference on Learning Representations*, 2026c.

Dan Hendrycks et al. Measuring coding challenge competence with APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021a.

Dan Hendrycks et al. Measuring mathematical problem solving with the math dataset. *ArXiv*, abs/2103.03874, 2021b.

Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Prompt-to-prompt image editing with cross-attention control. In *International Conference on Learning Representations*, 2023.

Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. CLIPScore: A reference-free evaluation metric for image captioning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7514–7528, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David Fleet. Video diffusion models. In *Advances in Neural Information Processing Systems*, volume 35, pages 8633–8646. Curran Associates, Inc., 2022.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*, 2020.

Jian Hu. REINFORCE++: A simple and efficient approach for aligning large language models. *arXiv preprint arXiv:2501.03262*, 2025.

Jian Hu et al. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *ArXiv*, abs/2405.11143, 2024a.

Shengding Hu et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *ArXiv*, abs/2404.06395, 2024b.

Yushi Hu et al. TIFA: Accurate and interpretable text-to-image faithfulness evaluation with question answering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 20406–20417, 2023.

Baizhou Huang, Shuai Lu, Xiaojun Wan, and Nan Duan. Enhancing large language models in coding through multi-perspective self-consistency. In *Proceedings of the 62nd Annual Meeting*

of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1429–1450, 2024a.

Kaiyi Huang, Kaiyue Sun, Enze Xie, Zhenguo Li, and Xihui Liu. T2I-CompBench: A comprehensive benchmark for open-world compositional text-to-image generation. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 36, pages 78723–78747, 2023.

Lifu Huang, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Cosmos qa: Machine reading comprehension with contextual commonsense reasoning. *arXiv preprint arXiv:1909.00277*, 2019.

Ziqi Huang et al. VBench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21807–21818. IEEE, 2024b.

Peter J. Huber. The behavior of maximum likelihood estimates under nonstandard conditions. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 221–233, Berkeley, CA, 1967. University of California Press.

Hugging Face. Spaces ZeroGPU: Dynamic GPU allocation for spaces, 2024. URL https://huggingface.co/docs/hub/spaces-zerogpu.

Hugging Face Spaces. Tokenizer arena. https://huggingface.co/spaces/eson/tokenizer-arena, 2024a. Accessed: 2024-06-05.

Hugging Face Spaces. Text to speech arena. https://huggingface.co/spaces/TTS-AGI/TTS-Arena, 2024b.

Binyuan Hui et al. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

David R. Hunter. MM algorithms for generalized Bradley-Terry models. *The Annals of Statistics*, 32(1):384–406, 2004.

Aaron Hurst et al. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Hakan Inan et al. Llama Guard: LLM-based input-output safeguard for human-AI conversations. *arXiv preprint arXiv:2312.06674*, 2023.

Hideki Isozaki, Tsutomu Hirao, Kevin Duh, Katsuhito Sudoh, and Hajime Tsukada. Automatic evaluation of translation quality for distant language pairs. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 944–952, Cambridge, MA, October 2010. Association for Computational Linguistics.

Naman Jain et al. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025a.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environment generation and hybrid verifiers for scaling open-weights SWE agents. In *Second Conference on Language Modeling*, 2025b.

Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-Blender: Ensembling large language models with pairwise ranking and generative fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14165–14178, Toronto, Canada, July 2023. Association for Computational Linguistics.

Dongfu Jiang et al. MANTIS: Interleaved multi-image instruction tuning. *Transactions on Machine Learning Research*, 2024a. Outstanding Paper Award.

Dongfu Jiang et al. GenAI Arena: An open evaluation platform for generative models. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 37, 2024b.

Dongfu Jiang, Yishan Li, Ge Zhang, Wenhao Huang, Bill Yuchen Lin, and Wenhu Chen. TIGERScore: Towards building explainable metric for all text generation tasks. *Transactions on Machine Learning Research*, 2024c.

Dongfu Jiang et al. VerlTool: Towards holistic agentic reinforcement learning with tool use. *Transactions on Machine Learning Research*, 2026.

Fangkai Jiao, Geyang Guo, Xingxing Zhang, Nancy F. Chen, Shafiq Joty, and Furu Wei. Preference optimization for reasoning with pseudo feedback. *ArXiv*, abs/2411.16345, 2024.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*, 2025.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

Vladimir Karpukhin et al. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, 2020.

Zixuan Ke et al. A survey of frontiers in llm reasoning: Inference scaling, learning to reason, and agentic systems. *Trans. Mach. Learn. Res.*, 2025, 2025.

Seungone Kim et al. Prometheus: Inducing fine-grained evaluation capability in language models. In *International Conference on Learning Representations*, 2024a.

Seungone Kim et al. Prometheus 2: An open source language model specialized in evaluating other language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024b.

Svetlana Kiritchenko and Saif Mohammad. Best-worst scaling more reliable than rating scales: A case study on sentiment intensity annotation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 465–470, Vancouver, Canada, July 2017. Association for Computational Linguistics.

Yuval Kirstain, Adam Polyak, Uriel Singer, Shahbuland Matiana, Joe Penna, and Omer Levy. Pick-a-Pic: An open dataset of user preferences for text-to-image generation. In *Advances in Neural Information Processing Systems*, volume 36, pages 36652–36663, 2023.

Yilun Kong et al. Tptu-v2: Boosting task planning and tool usage of large language model-based agents in real-world systems. *ArXiv*, abs/2311.11315, 2023.

Max Ku, Dongfu Jiang, Cong Wei, Xiang Yue, and Wenhu Chen. VIEScore: Towards explainable metrics for conditional image synthesis evaluation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12268–12290, 2024a.

Max Ku et al. ImagenHub: Standardizing the evaluation of conditional image generation models. In *International Conference on Learning Representations*, 2024b.

Sumith Kulal et al. SPoC: Search-based pseudocode to code. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Kwai-Kolors. Kolors, 2024. URL https://github.com/Kwai-Kolors/Kolors. GitHub repository.

Tom Kwiatkowski et al. Natural Questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

Woosuk Kwon et al. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Nathan Lambert et al. Tülu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025a.

Nathan Lambert et al. RewardBench: Evaluating reward models for language modeling. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1755–1797, 2025b.

Hugo Laurençon et al. OBELICS: An open web-scale filtered dataset of interleaved image-text documents. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 36, pages 71683–71702, 2023.

Hugo Laurençon, Léo Tronchon, Matthieu Cord, and Victor Sanh. What matters when building vision-language models? In *Advances in Neural Information Processing Systems*, volume 37, pages 87874–87907, 2024.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Tony Lee et al. Holistic evaluation of text-to-image models. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 36, pages 69981–70011, 2023.

Christoph Leiter, Piyawat Lertvittayakumjorn, M. Fomicheva, Wei Zhao, Yang Gao, and Steffen Eger. Towards explainable evaluation metrics for natural language generation. *ArXiv*, abs/2203.11131, 2022.

Patrick Lewis et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Aitor Lewkowycz et al. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems*, 2022.

Bo Li et al. Otter: A multi-modal model with in-context instruction tuning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(9):7543–7557, 2025a.

Daiqing Li, Aleks Kamko, Ehsan Akhgari, Ali Sabet, Linmiao Xu, and Suhail Doshi. Playground v2.5: Three insights towards enhancing aesthetic quality in text-to-image generation. *arXiv preprint arXiv:2402.17245*, 2024a.

Daiqing Li, Aleks Kamko, Ali Sabet, Ehsan Akhgari, Linmiao Xu, and Suhail Doshi. Playground v2, 2024b. URL https://huggingface.co/playgroundai/playground-v2-1024px-aesthetic.

Jiachen Li et al. T2V-Turbo: Breaking the quality bottleneck of video consistency model with mixed reward feedback. In *Advances in Neural Information Processing Systems*, volume 37, pages 75692–75726, 2024c.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19730–19742. PMLR, 2023a.

Kuan Li et al. WebSailor: Navigating super-human reasoning for web agent. *arXiv preprint arXiv:2507.02592*, 2025b.

Rongao Li et al. TACO: Topics in algorithmic COde generation dataset. *arXiv preprint arXiv:2312.14852*, 2023b.

Xuechen Li et al. AlpacaEval: An automatic evaluator of instruction-following models. GitHub repository, https://github.com/tatsu-lab/alpaca_eval, 2023c.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. ToRL: Scaling tool-integrated RL. arXiv preprint arXiv:2503.23383, 2025c.

Zhimin Li et al. Hunyuan-DiT: A powerful multi-resolution diffusion transformer with fine-grained Chinese understanding. *arXiv preprint arXiv:2405.08748*, 2024d.

Zhuofeng Li et al. OpenResearcher: A fully open pipeline for long-horizon deep research trajectory synthesis. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, 2026a.

Zhuofeng Li et al. Beyond semantic similarity: Rethinking retrieval for agentic search via direct corpus interaction, 2026b. arXiv preprint arXiv:2605.05242.

Hunter Lightman et al. Let's verify step by step. In *International Conference on Learning Representations*, 2024.

Bin Lin et al. Video-LLaVA: Learning united visual representation by alignment before projection. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5971–5984. Association for Computational Linguistics, 2024a.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, 2004.

Shanchuan Lin, Anran Wang, and Xiao Yang. SDXL-Lightning: Progressive adversarial diffusion distillation. *arXiv preprint arXiv:2402.13929*, 2024b.

Chris Yuhao Liu et al. Skywork-Reward: Bag of tricks for reward modeling in LLMs. *arXiv preprint arXiv:2410.18451*, 2024a.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26296–26306, 2024b.

Haotian Liu et al. LLaVA-NeXT: Improved reasoning, OCR, and world knowledge, January 2024c. URL https://llava-vl.github.io/blog/2024-01-30-llava-next/.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, 2023a.

Shu Liu et al. SkyRL-SQL: Matching GPT-4o and o4-mini on Text2SQL with multi-turn RL, 2025a. Notion Blog.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore, December 2023b. Association for Computational Linguistics.

Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. RM-Bench: Benchmarking reward models of language models with subtlety and style. In *International Conference on Learning Representations*, 2025b.

Yaofang Liu et al. EvalCrafter: Benchmarking and evaluating large video generation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22139–22149, 2024d.

Yixin Liu, Pengfei Liu, Dragomir Radev, and Graham Neubig. Brio: Bringing order to abstractive summarization. *arXiv preprint arXiv:2203.16804*, 2022.

Zhihan Liu, Shenao Zhang, Yongfei Liu, Boyi Liu, Yingxiang Yang, and Zhaoran Wang. DSTC: Direct preference learning with only self-generated tests and code to improve code LMs. *arXiv preprint arXiv:2411.13611*, 2024e.

Meng Lu et al. Scaling agentic reinforcement learning for tool-integrated reasoning in VLMs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026a.

Pan Lu et al. Chameleon: Plug-and-play compositional reasoning with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023a.

Qingyu Lu, Baopu Qiu, Liang Ding, Liping Xie, and Dacheng Tao. Error analysis prompting enables human-like translation evaluation in large language models: A case study on chatgpt. *ArXiv*, abs/2303.13809, 2023b.

Yi Lu et al. Dr-DCI: Scaling direct corpus interaction via dynamic workspace expansion, 2026b. arXiv preprint arXiv:2606.14885.

Stephan Lukasczyk and Gordon Fraser. Pynguin: Automated unit test generation for Python. In *Proceedings of the 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 168–172, 2022.

Haipeng Luo et al. WizardMath: Empowering mathematical reasoning for large language models via reinforced Evol-Instruct. In *International Conference on Learning Representations*, 2025.

Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. An empirical analysis of flaky tests. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 643–653, 2014.

Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv preprint arXiv:2310.04378*, 2023a.

Ziyang Luo et al. Wizardcoder: Empowering code large language models with evol-instruct, 2023b.

Xueguang Ma, Qian Liu, Dongfu Jiang, Ge Zhang, Zejun Ma, and Wenhu Chen. General-Reasoner: Advancing LLM reasoning across all domains. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Hannaneh Hajishirzi, and Daniel Khashabi. When not to trust language models: Investigating effectiveness and limitations of parametric and non-parametric memories. *arXiv preprint arXiv:2212.10511*, 7, 2022.

David Manheim and Scott Garrabrant. Categorizing variants of Goodhart's law. arXiv preprint arXiv:1803.04585, 2018.

Chenlin Meng et al. SDEdit: Guided image synthesis and editing with stochastic differential equations. In *International Conference on Learning Representations*, 2022.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*, 2024.

Philipp Moritz et al. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 561–577, Carlsbad, CA, 2018. USENIX Association.

Ivan Moshkov et al. Aimo-2 winning solution: Building state-of-the-art mathematical reasoning models with openmathreasoning dataset, 2025. URL https://arxiv.org/abs/2504.16891.

Linyong Nan et al. Fetaqa: Free-form table question answering. *Transactions of the Association for Computational Linguistics*, 10:35–49, 2022.

Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 1999.

Jekaterina Novikova, Ondřej Dušek, Amanda Cercas Curry, and Verena Rieser. Why we need

new evaluation metrics for NLG. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2241–2252, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

NVIDIA et al. Nemotron-4 340B technical report. arXiv preprint arXiv:2406.11704, 2024.

OpenAI. Chat generative pre-trained transformer (chatgpt). https://www.openai.com/, 2023.

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

openjourney.ai. Openjourney is an open source stable diffusion fine tuned model on midjourney images, 2023. URL https://huggingface.co/prompthero/openjourney.

Mayu Otani et al. Toward verifiable and reproducible human evaluation for text-to-image generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14277–14286, 2023.

Long Ouyang et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball. Feedback-directed random test generation. In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, pages 75–84, 2007.

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*, pages 311–318, 2002.

Gaurav Parmar, Krishna Kumar Singh, Richard Zhang, Yijun Li, Jingwan Lu, and Jun-Yan Zhu. Zero-shot image-to-image translation. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–11, 2023.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4172–4182. IEEE, 2023.

Zhiliang Peng et al. Kosmos-2: Grounding multimodal large language models to the world. *ArXiv*, abs/2306.14824, 2023.

Pablo Pernías, Dominic Rampas, Mats L. Richter, Christopher Pal, and Marc Aubreville. Würstchen: An efficient architecture for large-scale text-to-image diffusion models. In *International Conference on Learning Representations*, 2024.

Long Phan et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Aske Plaat, Max J. van Duijn, Niki van Stein, Mike Preuss, Peter van der Putten, and Kees Joost Batenburg. Agentic large language models, a survey. *ArXiv*, abs/2503.23037, 2025.

Dustin Podell et al. SDXL: Improving latent diffusion models for high-resolution image synthesis. In *International Conference on Learning Representations*, 2024.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*, 2022.

Shanghaoran Quan et al. Codeelo: Benchmarking competition-level code generation of llms with human-comparable elo ratings. *ArXiv*, abs/2501.01257, 2025.

Alec Radford et al. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 2021.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *ArXiv*, abs/2204.06125, 2022.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In *International Conference on Learning Representations*, 2016.

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. COMET: A neural framework for MT evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online, November 2020. Association for Computational Linguistics.

Ricardo Rei et al. COMET-22: Unbabel-IST 2022 submission for the metrics shared task. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 578–585, Abu Dhabi, United Arab Emirates (Hybrid), December 2022a. Association for Computational Linguistics.

Ricardo Rei et al. CometKiwi: IST-Unbabel 2022 submission for the quality estimation shared task. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 634–645, Abu Dhabi, United Arab Emirates (Hybrid), December 2022b. Association for Computational Linguistics.

Ehud Reiter. A structured review of the validity of BLEU. *Computational Linguistics*, 44(3): 393–401, September 2018.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Bjorn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10674–10685. IEEE, 2022.

Chi Ruan, Dongfu Jiang, Yubo Wang, and Wenhu Chen. Critique-Coder: Enhancing coder models by critique reinforcement learning. In *International Conference on Learning Representations*, 2026a.

Chi Ruan, Dongfu Jiang, Huaye Zeng, Ping Nie, and Wenhu Chen. EvolveCoder: Evolving test cases via adversarial verification for code reinforcement learning, 2026b. arXiv preprint arXiv:2603.12698.

Tim Salimans et al. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Axel Sauer, Dominik Lorenz, Andreas Blattmann, and Robin Rombach. Adversarial diffusion distillation. In *Computer Vision – ECCV 2024*, Lecture Notes in Computer Science, pages 87–103. Springer, 2024.

Timo Schick et al. Toolformer: Language models can teach themselves to use tools. In *Advances*

in Neural Information Processing Systems, volume 36, 2023.

Benjamin Schneider, Dongfu Jiang, Chao Du, Tianyu Pang, and Wenhu Chen. QuickVideo: Real-time long video understanding with system algorithm co-design. *Transactions on Machine Learning Research*, 2026.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Thibault Sellam, Dipanjan Das, and Ankur Parikh. BLEURT: Learning robust metrics for text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7881–7892, Online, July 2020. Association for Computational Linguistics.

Zhihong Shao et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yue Ting Zhuang. HuggingGPT: Solving AI tasks with ChatGPT and its friends in Hugging Face. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Guangming Sheng et al. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys)*, pages 1279–1297, 2025.

Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K Reddy. Execution-based code generation using deep reinforcement learning. *Transactions on Machine Learning Research*, 2023.

Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In *Advances in Neural Information Processing Systems*, volume 35, pages 9460–9471. Curran Associates, Inc., 2022.

Huatong Song et al. R1-Searcher: Incentivizing the search capability in LLMs via reinforcement learning. arXiv preprint arXiv:2503.05592, 2025.

Stability AI. CosXL, 2024a. URL https://huggingface.co/stabilityai/cosxl.

Stability AI. Stable diffusion 3 release, 2024b. URL https://stability.ai/news/stable-diffusion-3. News release.

Ivan Stelmakh, Yi Luan, Bhuwan Dhingra, and Ming-Wei Chang. Asqa: Factoid questions meet long-form answers. *arXiv preprint arXiv:2204.06092*, 2022.

Nisan Stiennon et al. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems*, volume 33, pages 3008–3021. Curran Associates, Inc., 2020.

Alex Su, Haozhe Wang, Weiming Ren, Fangzhen Lin, and Wenhu Chen. Pixel Reasoner: Incentivizing pixel space reasoning via curiosity-driven reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Chuyue Sun, Ying Sheng, Oded Padon, and Clark W. Barrett. Clover: Closed-loop verifiable code generation. In *SAIV*, 2023.

Shuang Sun et al. SimpleDeepSearcher: Deep information seeking via web-powered reasoning trajectory synthesis. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 13705–13720, 2025.

Xiaohang Tang, Rares Dolga, Sangwoong Yoon, and Ilija Bogunovic. wd1: Weighted policy optimization for reasoning in diffusion language models. In *International Conference on Learning Representations*, 2026.

Gemini Team et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Team Olmo. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

THUDM. slime: A llm post-training framework aiming at scaling rl. https://github.com/THUDM/slime, 2024. Software framework for LLM post-training with reinforcement learning scaling.

Hugo Touvron et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Narek Tumanyan, Michal Geyer, Shai Bagon, and Tali Dekel. Plug-and-play diffusion features for text-driven image-to-image translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1921–1930, 2023.

Jonathan Uesato et al. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Thomas Unterthiner, Sjoerd van Steenkiste, Karol Kurach, Raphael Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric & challenges. *arXiv preprint arXiv:1812.01717*, 2018.

Ashish Vaswani et al. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Chaoyang Wang et al. AdaTooler-V: Adaptive tool-use for images and videos. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026.

Haozhe Wang et al. To code or not to code? adaptive tool integration for math language models via expectation-maximization. *arXiv preprint arXiv:2502.00691*, 2025a.

Jiuniu Wang, Hangjie Yuan, Dayou Chen, Yingya Zhang, Xiang Wang, and Shiwei Zhang. ModelScope text-to-video technical report. *arXiv preprint arXiv:2308.06571*, 2023a.

Liang Wang et al. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

Peiyi Wang et al. Large language models are not fair evaluators. In *Proceedings of the 62nd*

Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 9440–9450, 2024a.

Weihan Wang et al. CogVLM: Visual expert for pretrained language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 121475–121499, 2024b.

Weixun Wang et al. Reinforcement learning optimization for large-scale learning: An efficient and user-friendly scaling library. *arXiv preprint arXiv:2506.06122*, 2025b.

Xingyao Wang et al. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 50208–50232, 2024c.

Xingyao Wang et al. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations*, 2025c.

Yaohui Wang et al. LaVie: High-quality video generation with cascaded latent diffusion models. *International Journal of Computer Vision*, 133(5):3059–3078, 2025d.

Yibin Wang, Yuhang Zang, Hao Li, Cheng Jin, and Jiaqi Wang. Unified reward model for multimodal understanding and generation, 2025e. arXiv preprint arXiv:2503.05236.

Yidong Wang et al. Pandalm: An automatic evaluation benchmark for llm instruction tuning optimization. *ArXiv*, abs/2306.05087, 2023b.

Yizhong Wang et al. Self-Instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, 2023c.

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004.

Zihan Wang et al. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025f.

Jason Wei et al. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022a.

Jason Wei et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022b.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with OSS-Instruct. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, 2024.

Elaine J. Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–470, 1982.

Chen Henry Wu and Fernando De la Torre. A latent space of stochastic diffusion models for zero-shot image editing and guidance. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7378–7387, 2023.

Penghao Wu and Saining Xie. V*: Guided visual search as a core mechanism in multimodal LLMs.

In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13084–13094, 2024.

Xiaoshi Wu et al. Human preference score v2: A solid benchmark for evaluating human preferences of text-to-image synthesis. arXiv preprint arXiv:2306.09341, 2023.

Can Xu et al. Wizardlm: Empowering large language models to follow complex instructions. *ArXiv*, abs/2304.12244, 2023a.

Jiazheng Xu et al. ImageReward: Learning and evaluating human preferences for text-to-image generation. In *Advances in Neural Information Processing Systems*, volume 36, pages 15903–15935. Curran Associates, Inc., 2023b.

Peng Xu et al. LVLM-eHub: A comprehensive evaluation benchmark for large vision-language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(3):1877–1893, 2025.

Sihan Xu, Yidong Huang, Jiayi Pan, Ziqiao Ma, and Joyce Chai. Inversion-free image editing with natural language. In *Conference on Computer Vision and Pattern Recognition 2024*, 2024.

Wenda Xu, Xian Qian, Mingxuan Wang, Lei Li, and William Yang Wang. SESCORE2: Learning text generation evaluation via synthesizing realistic mistakes. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5166–5183, Toronto, Canada, July 2023c. Association for Computational Linguistics.

Wenda Xu et al. INSTRUCTSCORE: Towards explainable text generation evaluation with automatic feedback. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5967–5994, Singapore, December 2023d. Association for Computational Linguistics.

Zhenghai Xue et al. SimpleTIR: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. In *International Conference on Learning Representations*, 2026.

An Yang et al. Qwen2 technical report, 2024a. URL https://arxiv.org/abs/2407.10671.

Jialin Yang et al. StructEval: Benchmarking LLMs' capabilities to generate structural outputs. *Transactions on Machine Learning Research*, 2026. Journal-to-Conference certification at ICLR 2026.

John Yang et al. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024b.

Zhilin Yang et al. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, 2018.

Zhuoyi Yang et al. CogVideoX: Text-to-video diffusion models with an expert transformer. In *International Conference on Learning Representations*, 2025.

Shunyu Yao et al. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Zijun Yao et al. Are reasoning models more prone to hallucination? *ArXiv*, abs/2505.23646, 2025.

Tao Yu et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and Text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, 2018.

Weizhe Yuan, Graham Neubig, and Pengfei Liu. BARTScore: Evaluating generated text as text generation. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Huaye Zeng, Dongfu Jiang, Haozhe Wang, Ping Nie, Xiaotong Chen, and Wenhu Chen. AceCoder: Acing coder RL via automated test-case synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12023–12040, 2025.

Hang Zhang et al. Scaling medical reasoning verification via tool-integrated reinforcement learning, 2026. arXiv preprint arXiv:2601.20221.

Kai Zhang, Lingbo Mo, Wenhu Chen, Huan Sun, and Yu Su. MagicBrush: A manually annotated dataset for instruction-guided image editing. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 36, pages 31428–31449, 2023a.

Kexun Zhang, Danqing Wang, Jingtao Xia, William Yang Wang, and Lei Li. Algo: Synthesizing algorithmic programs with generated oracle verifiers. *ArXiv*, abs/2305.14591, 2023b.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. BERTScore: Evaluating text generation with BERT. In *International Conference on Learning Representations*, 2020.

Lianmin Zheng et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 36, 2023a.

Lianmin Zheng et al. Efficiently programming large language models using sglang, 2023b.

Yaowei Zheng et al. LlamaFactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024a. Association for Computational Linguistics.

Zangwei Zheng et al. Open-Sora: Democratizing efficient video production for all. *arXiv preprint arXiv:2412.20404*, 2024b.

Ming Zhong et al. Towards a unified multi-dimensional evaluator for text generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2023–2038, 2022.

Chunting Zhou et al. Lima: Less is more for alignment, 2023.

Giulio Zhou and Gerasimos Lampouras. Webnlg challenge 2020: Language agnostic delexicalisation for multilingual rdf-to-text generation. In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 186–191, 2020.

Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. In *The Twelfth International Conference on Learning Representations*, 2023.

Xiao Zhu et al. CodeScaler: Scaling code LLM training and test-time inference via reward models. In *Advances in Neural Information Processing Systems*, 2026.

Terry Yue Zhuo et al. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, 2025.

APPENDICES

Appendix A

Other Work Completed During the Doctorate

The four research chapters of this thesis are adapted from four papers, and the papers that extend them directly (VideoScore, VideoScore2, StructEval, EvolveCoder, General-Reasoner, Critique-Coder, OpenResearcher, DCI, and Dr-DCI) are discussed in Section 7.2. This appendix lists the other papers on which I was a first or co-first author. None of them is claimed as a contribution of this thesis.

LLM-Blender: Ensembling Large Language Models with Pairwise Ranking and Generative Fusion. ACL 2023, first author (Jiang et al., 2023). Different models excel on different inputs, which motivates comparing outputs against each other instead of scoring each one independently. PairRanker encodes input–candidate pairs jointly to determine fine-grained preferences; GenFuser synthesizes the top-ranked outputs into a better one. The idea that a comparative judgment can be easier to obtain reliably than a calibrated absolute score also underlies the pairwise votes of Chapter 4 and the preference pairs of Chapter 5.

MANTIS: Interleaved Multi-Image Instruction Tuning. TMLR 2024, first author; received the TMLR Outstanding Paper Award (Jiang et al., 2024a). We constructed Mantis-Instruct, 721K multi-image instruction examples covering co-reference, comparison, reasoning, and temporal understanding, and used it for instruction tuning at academic scale. With about 200× less multi-image data than the 140M interleaved examples used to pretrain Idefics2-8B, this tuning reached state-of-the-art results on all eight multi-image

benchmarks it was evaluated on, and improved on Idefics2-8B by 13 points on average. It is not a chapter because it concerns training data for perception and does not study feedback.

QuickVideo: Real-Time Long Video Understanding with System–Algorithm Co-Design. TMLR 2026, co-first author (Schneider et al., 2026). Hour-long video presents two bottlenecks: sequential decoding that can take up to a minute, and prefilling millions of visual tokens. QuickVideo addresses both with QuickCodec, a parallelized CPU decoder that splits a video into keyframe-aligned intervals for a 2–3× decoding speedup, together with KV-cache pruning during prefill and a scheme that overlaps decoding with GPU inference. Combined, these components remove about a minute from the time needed to process an hour-long video.

Appendix B

Supplementary Material from the Published Papers

This appendix reproduces the parts of the published papers' supplementary material that the research chapters refer to. Ethics, broader-impact, and data-collection statements of the papers are omitted.

B.1 TIGERScore (Chapter 3)

B.1.1 Prompting Strategies

In our study to extract high-quality error-analysis insights from GPT-4, we employed various intuitive prompting strategies. These strategies are detailed in the prompting templates found in subsection B.1.7. Key strategies are outlined below:

Two-Step Generation and Formatting Process One of the challenges in eliciting structured knowledge from LLMs is directing them to generate content in a specific format. Although GPT-4 often adheres to given formats, there are instances of deviations. We observed that enforcing a strict format can compromise content quality, as indicated by reduced correlation in our analysis. Our approach involves initially allowing GPT-4 to generate responses freely in the first conversation turn. In the subsequent turn, we request GPT-4 to reformat its response according to a predefined format. The initial generation

templates for each task, along with a singular template for the formatting step, are listed in subsection B.1.7.

Incorporation of Task-Specific Words in Initial Queries To leverage GPT-4's task-specific knowledge, we designed varied prompting templates for different tasks with slight modifications. Keywords like 'Source,' 'Instruction,' 'Reference,' 'Output,' 'Solution,' 'Translation,' and 'Summary' are dynamically utilized in various task contexts. In tasks like instruction-following, where the context is self-explanatory, we omitted specific keywords.

Integration of Predefined Aspect Definitions for Focused Evaluation Directly requesting GPT-4 to evaluate task outputs often led to low-quality error identification. It either points out simple discrepancies with the reference or misses crucial evaluation aspects, thus overlooking some errors. To address this, we incorporated predefined evaluation aspects defined in Table B.2 into the templates, guiding GPT-4 to produce more focused responses. Exceptionally, for data2text, we found that directly evaluating errors was good enough, and thus, we did not include our predefined aspects.

Classification of Errors Using Major/Minor Error Guidelines Drawing inspiration from the MQM translation human rating system and InstructScore prompting template (Freitag et al., 2021; Xu et al., 2023d), we classified translation errors as either major or minor. This classification helped GPT-4 in assigning more consistent scores to each error, countering its instability with numerical judgments (Lu et al., 2023b).

Adopting a 0.5 to 5 Scale for Scoring Error Severity Initially, we used an integer scale from 1 to 5 for error penalty scores. While effective in translation tasks, this scale is less effective in tasks like summarization, data2text, and instruction-following. Our experiments demonstrated that a more nuanced scoring scale ranging from 0.5 to 5 yielded better correlation across all tasks

B.1.2 Creation of evaluation aspects

With the assistance of GPT-4, we have carefully curated the evaluation aspects of each task that are mutually exclusive and collectively exhaustive. The steps include:

- Step 1: We prompt GPT-4 to output 20 candidate aspects for each task.

- Step 2: We ask GPT-4 to summarize these aspects into 3 to 5 general aspects for this task.
- Step 3: We ask GPT-4 to generate detailed definition and 5 specific error types under each aspect.

In each step, we check the reasonability of GPT-4's response and make necessary modifications to the responses, including summarized error aspects, error definition, and error types, to make them clear, concise, and typical. Example prompts used in each step are show in Table B.13.

B.1.3 TIGERScore prompt template

You are evaluating errors in a model-generated output for a given instruction. Instruction: ${generation_instruction} ${input_context} Model-generated Output: ${hypothesis_output} For each error you give in the response, please also elaborate the following information: - error location (the words that are wrong in the output) - error aspect it belongs to. - explanation why it's an error, and the correction suggestions. - severity of the error ("Major" or "Minor"). - reduction of score (between 0.5 and 5 given the severity of the error) Your evaluation output:

Table B.1: The prompt template used for the fine-tuning and inference of TIGERScore

B.1.4 MetricInstruct data statistics analysis

In order to find the secret of the success of MetricInstruct, we conducted a deep analysis of its inherent statistics, as shown in Figure B.1. we first count the distribution of data length through the Llama tokenizer. The results show that more than 90% of data has a length lower than 1024. Due to the lack of GPU resources for fine-tuning in the longer context length scenario, this distribution further demonstrates the reasonability of using 1024 as the context length of TIGERScore.

Furthermore, we examined the incidence of errors per data instance across various tasks. The figure illustrates that each task contains most instances with the number of errors being 1 or 2 and fewer instances are considered perfect and very erroneous, reflecting a naturally occurring distribution. We contend that such a balanced distribution is crucial for the model's performance—it helps in reducing fabricated errors in correct outputs and aids in the precise identification of minor and major errors in outputs that are partially correct or entirely incorrect.

Additionally, we categorized errors as 'Major' or 'Minor' and quantified their occurrences. Our analysis reveals that tasks of a more subjective nature, such as translation and summarization, tend to have a higher frequency of minor errors. Contrastingly, in tasks like MathQA, the predominance of major errors is in alignment with the expectation that mathematical inaccuracies are generally more critical."

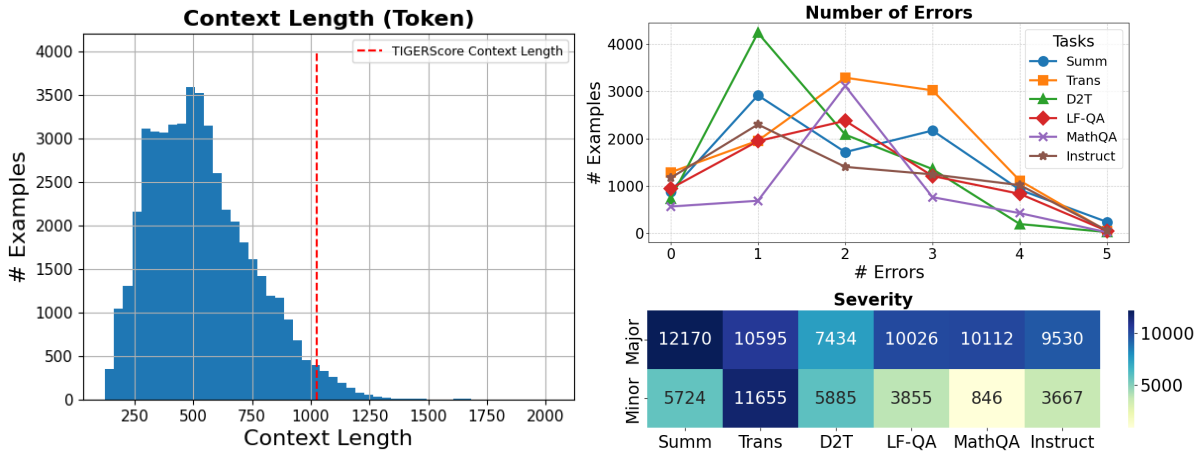


Figure B.1: Distribution analysis of MetricInstruct training data for the context length, number of errors, and error severities. The left figure shows the context length distribution. The right-top plot illustrates the distribution of the per-instance number of errors. The right-bottom figure illustrates the counts of "Major" and "Minor" errors for all six tasks

B.1.5 Gold scores for correlation computation

Human preference scores are necessary to conduct the correlation analysis. Those datasets with official system outputs released are usually accompanied by systematic human preference scores, like the WMT-MQM score for the translation task. However, these scores

are not available for tasks like long-form QA, MathQA, and instruction-following, where we need to create on our own.

Therefore, we here introduce what human preference scores we have used to conduct the evaluation experiments. For summarization, data2text, and story generation tasks, we use their official human ratings from multiple aspects of their released outputs. For translation, we use the official WMT-22 MQM scores as the gold scores. For MathQA, we simply use the accuracy (1 or 0) as the gold preferences. For instruct-following, we use human ratings from the hugging face community of a dataset where 500 instances are sampled from LIMA and AlpacaEvla. For Long-form QA, we use the powerful GPT-4 to perform pairwise comparisons for them and count the winning times as the way to rank them, which is similar to how Jiang et al. (2023) constructs MixInstruct.

B.1.6 Evaluation aspects for all tasks

Task	Aspect	Definition
Summ	Relevance	The degree to which the summarized output accurately reflects the key points of the input text.
	Fact Consistency	If the facts in the summary are consistent with the facts in the original text.
	Coherence	Pertains to the logical and meaningful arrangement of ideas in the summary.
	Fluency	Reviews the model-generated output's use of language, including grammar, punctuation, and vocabulary that affect the quality of the sentences.
Trans	Accuracy	The degree to which the translated text adheres to the original text, maintaining the same meaning, context and cultural nuances.
	Fluency	How naturally the translation reads in the target language.
	Terminology	The appropriate use of specific terms and jargon related to a particular field or industry.
	Style Matching	Translator's ability to maintain the same style, tone, and voice as the original text. Example error types include:
D2T	Accuracy	Deals with the correctness of the information presented by the output.
	Logical Coherence	How well the output transforms structured data into a comprehensible, logical, and engaging text.
	Fluency	Reviews the model-generated output's use of language, including grammar, punctuation, and vocabulary that affect the quality of the sentences.
LF-QA	Accuracy	Evaluates the factual correctness of the answer.
	Completeness	Evaluates if the answer leaves out any critical parts or details that were asked in the question.
	Informativeness	Assesses the quality of the response in terms of how helpful it is for the user to understand the answer.
	Clarity	Assesses the readability and understandability of the response.
MathQA	Problem Understanding	Assesses how well the output accurately comprehend the text-based description of the math problem.
	Problem Formulation	Involves translating the problem from a textual form into a mathematical equation or set of equations that can be solved.
	Computing Accuracy	Assesses the output's ability to perform the mathematical operations accurately to arrive at the correct solution.
	Solution Interpretation	Involves the how well the output correctly interpret the solution of the problem in the context of the original problem statement.
Instruct	Comprehension	Evaluates how well the output understands the given instruction.
	Accuracy	Measures the correctness of the output in relation to the instruction and the paired input context.
	Informativeness	Assesses the relevancy and usefulness of the information provided by the output.
	Coherence	Evaluates how logically the output flows and connects.

Table B.2: Definitions of evaluation aspects of TIGERScore for the 6 text generation task.

B.1.7 Prompting templates

Source: ${input_context} Reference: ${reference_output} Output:${hypothesis_output} Based on the given Source and Reference, please evaluate the quality of summary(Output) written for the input text. Please score the summarization with 0.5 to 5 for aspects below. Then, identify the major and minor errors in this output for the $task task. There may be multiple errors or no error in the output. Here are the aspects you need to focus on: ${aspects_descriptions}

Table B.3: Prompting templates for summarization task

Translation Instruction: ${generation_instruction} Source Text: ${input_context} ${reference_output} Model-generated Translation: ${hypothesis_output} Please identify and categorize the errors in the model-generated translation as either Major or Minor. Major errors significantly impact the task, while Minor errors are subjective and represent minor imperfections. When identifying errors, do not solely rely on the reference translation for comparison. Provide explanations as an expert in the task domain, without explicitly mentioning the reference output.

Table B.4: Prompting templates for translation task

Task instruction:{generation_instruction} Source: $ {input_context} $ {reference_output} Output: $ {hypothesis_output} Based on the given source and reference, identify the major and minor errors in this Output for the data to text task, which is to $generation_instruction. Note that Major errors refer to actual errors that affects the task severely, may change the meaning of the output, and Minor errors refer to smaller imperfections, and purely subjective opinions about the output. There may be multiple errors or no error in the output.

Table B.5: Prompting templates for data2text task

${generation_instruction} ${input_context} The correct solution is: ${reference_output} A model-generated solution is: ${hypothesis_output} Please identify all the errors in this output considering the following aspects: ${aspects_list}

Table B.6: Prompting templates for mathQA task

Source: ${input_context}
${reference_output}
Output: ${hypothesis_output}
Based on the given Source and reference, identify the major and minor errors in this Output for the ${task} task, which is to ${generation_instruction}. Note that Major errors refer to actual errors that affects the task severely, may change the meaning of the output, and Minor errors refer to smaller imperfections, and purely subjective opinions about the output. You should check about ${aspects_descriptions}.There may be multiple errors or no error in the output.

Table B.7: Prompting templates for long-form QA task

${generation_instruction_and_source}
${reference_output}
Output: ${hypothesis_output}
Based on the given Source and reference, identify the major and minor errors in this Output for the ${task} task. Note that Major errors refer to actual errors that affects the task severely, may change the meaning of the output, and Minor errors refer to smaller imperfections, and purely subjective opinions about the output. You should check about ${aspects_descriptions}.There may be multiple errors or no error in the output.

Table B.8: Prompting templates for instruction-following task

For each error identified in your response, please provide the following information in a specific JSON format:
- Error Location: The substring in the Output that contains the error.
- Error Aspect: Choose only one from ${aspects_list}.
- Explanation: Describe why the identified issue is an error, and offer suggestions for correction. Explain as an expert in the task domain, without explicitly mentioning the reference output.
- Severity: Classify the error as "Major" or "Minor".
- Score Reduction: Assign a reduction score between 0.5 and 5, considering the severity of the error.
JSON Format for Output:
- If there are no errors:
{"errors": {}}
- If there are errors:
{"errors": {
"error_1": {
"error_location": "...",
"error_aspect": "...",
"explanation": "...",
"severity": "...",
"score_reduction": ...
},
...
}}

Table B.9: Prompt template to format the response into json format

You are generating an output given the following instruction and context:
${generation_instruction}
${input_context}

The correct output is: ${reference_output}

Please generate an incorrect output for the given instruction and context by modifying the
correct output. The incorrect output should contain the following errors:
${error_requirements}
For each error, give me the
- error location (the substring that is wrong in the generated incorrect output)
- error aspect
- explanation (the generic error type description, why it's an error, and the correction suggestions)
- severity ("major" or "minor")
- score reduction (an integer between 1 to 5 given the severity of the error)

Output format:
Generated incorrect output:

Error location 1:
Error aspect 1:
Explanation 1:
Severity 1:
Score reduction 1:
...

Table B.10: Prompt template to generate data in the synthetic channel for all 6 tasks.

Instruction:
${instruction}
${input}

A ground-truth response:
${output}

A model will be asked to respond to this instruction. However, that response might contain errors in various aspects.

Please first output 5 possible error aspects if a model is asked to generate a response for the above instruction. The error aspects don't have to be one of the above aspects and can be any aspect that you think is reasonable for this instruction.

Then generate an incorrect response contains up to ${num_errors} errors of these aspects. Each error corresponds to one of the aspect.
The incorrect response should mimic style the real-generation of a model.

Then give an analysis of these errors. For each error, give me the
- error location (the substring that is wrong in the generated incorrect output)
- error aspect
- explanation (the generic error type description, why it's an error, and the correction suggestions)
- severity ("major" or "minor")
- score reduction (an integer between 1 to 5 given the severity of the error)

Output format:
Generated incorrect output:

Error location 1:
Error aspect 1:
Explanation 1:
Severity 1:
Score reduction 1:
...

Table B.11: Prompt template to generate data with free-form aspects in the synthetic channel for instruction-following task. Data generated from this template aims to improve TIGERScore's generalization ability on the unseen error cases and enhance the robustness.

${instruction}
${input}

Model-generated output:
${output}

An error analysis provided:
${error_analysis}

Is the error analysis reasonable? Answer me "yes" or "no" only.

Table B.12: Prompt template used to check whether the error analysis synthesized is reasonable or not.

You are evaluating a model-generated output for the mathematical word problem.
There might be multiple errors in the output and they can focus on different evaluation aspects.
Please list 20 evaluation aspects for mathematical word problems and their definitions.

Please summarize the above-provided aspects into 3 to 5
general aspects that are mutually exclusive and collectively exhaustive.

Mathematical word problems are a type of math task that requires model
to read and understand a text-based description of a math situation and
then formulate and solve a mathematical equation to solve the problem.
Now you are evaluating a model-generated output for a mathematical word problem.
you might want to design a few aspects where errors are identified in the outputs
could be attributed to these aspects.
These aspects could be
"Problem Understanding", "Problem Formulation",
"Computing Accuracy", "Solution Interpretation",
provide the definition of each aspect above and 5 specific error types for each aspect.
Note that these aspects are used to evaluate a single output instead of a system.

Table B.13: An example prompt when creating the template of mathQA

Task instruction: ${instruction} Source: ${input} Reference output: ${reference_output} Model-generated output: ${output} An error analysis provided: ${error_analysis} Does error analysis include factual errors or hallucinations of the model-generated output?

Table B.14: The prompt template we used to identify whether there are hallucinated contents or factor errors in TIGERScore's response

B.1.8 Correlation results on Pearson and Spearman

Tasks→	Summ	Trans	D2T	LF-QA	MathQA	Instrct	Story-Gen	Average
GPT-based Metrics								
GPT-3.5-turbo (zero-shot)	**45.53**	**43.77**	**47.76**	29.84	**61.26**	15.36	7.80	35.90
GPT-4 (zero-shot)	40.75	33.92	46.83	**49.30**	54.98	**60.45**	**37.74**	**46.28**
Reference-based Metrics								
BLEU	11.66	17.47	34.29	18.21	18.12	29.47	-0.64	18.37
ROUGE-2f	16.03	16.26	35.85	19.66	20.69	33.49	2.88	20.69
InstructScore	27.40	51.55	47.28	20.59	0.36	20.98	12.81	25.85
GPTScore-ref	13.47	21.05	48.70	33.40	18.22	29.66	18.94	26.20
BARTScore-cnn (hypo-ref)	16.67	23.56	45.08	32.78	**23.09**	26.57	27.61	27.91
BARTScore-para (hypo-ref)	19.73	29.04	47.89	32.70	17.33	30.20	17.76	27.81
BERTScore	26.26	37.65	48.22	26.39	11.19	45.58	4.08	28.48
BLEURT	17.27	43.00	**54.32**	34.26	3.98	39.15	27.89	31.41
UniEval (summ)	**53.22**	23.11	51.14	**36.95**	17.69	30.87	**44.88**	36.84
COMET-22	35.32	**58.46**	43.82	36.79	-5.58	**49.68**	40.12	**36.94**
Reference-free Metrics								
BARTScore-para (src-hypo)	43.11	6.96	37.82	29.86	-0.41	19.37	19.99	22.38
BARTScore-cnn (src-hypo)	39.72	9.53	45.43	41.48	3.28	34.97	33.51	29.70
Llama-2-13b-chat-0-shot	29.59	9.09	41.32	21.67	2.80	22.71	21.13	21.19
COMETKiwi	14.22	**50.91**	23.63	22.59	-13.35	34.46	19.12	21.65
GPTScore-src	41.71	6.82	41.19	39.79	13.99	27.59	23.22	27.76
TigerScore-7B	43.95	37.70	49.13	**46.10**	21.77	38.26	39.90	39.54
TigerScore-13B	**44.21**	41.54	**52.87**	44.76	**24.41**	**47.52**	**47.66**	**43.28**
Δ (ours - best reference-free)	+1	-9	+7	+5	+10	+20	+14	+13
Δ (ours - best reference-based)	-9	-17	-2	+9	+1	-2	+3	+6

Table B.15: The **Pearson** correlation results of all the baseline metrics and TIGERSCORE on the evaluation datasets shown in Table 3.3. For each task, the metric with the highest correlation to average performance is highlighted in bold.

Tasks→	Summ	Trans	D2T	LF-QA	MathQA	Instrct	Story-Gen	Average
GPT-based Metrics								
GPT-3.5-turbo (zero-shot)	**38.50**	40.53	40.20	29.33	**66.46**	23.20	4.77	34.71
GPT-4 (zero-shot)	36.46	**43.87**	**44.04**	**48.95**	51.71	**58.53**	**32.48**	**45.15**
Reference-based Metrics								
BLEU	11.98	19.73	33.29	11.38	21.12	**46.61**	-1.17	20.42
ROUGE-2f	14.53	17.83	35.49	16.83	22.12	44.56	2.34	21.96
InstructScore	26.33	50.43	38.54	21.62	-4.15	16.19	16.13	23.58
GPTScore-ref	14.73	24.95	39.42	31.60	18.20	33.14	18.24	25.75
BARTScore-cnn(hypo-ref)	13.64	28.53	36.12	29.57	**23.35**	32.49	26.64	27.19
BARTScore-para (hypo-ref)	17.18	33.72	40.79	28.94	17.27	34.47	17.43	27.11
BERTScore	23.67	42.41	43.75	25.60	11.53	45.77	2.88	27.95
BLEURT	17.30	48.41	**48.76**	33.26	3.53	36.46	27.52	30.75
UniEval(summ)	**47.52**	21.90	38.38	**41.83**	19.78	16.02	**44.46**	32.84
COMET-22	33.75	**56.35**	33.92	35.28	-5.53	46.13	39.20	**34.16**
Reference-free Metrics								
BARTScore-para (src-hypo)	**38.68**	9.60	32.26	26.86	-2.70	5.92	20.55	18.74
BARTScore-cnn (src-hypo)	35.50	12.83	34.33	40.96	1.50	25.43	33.48	26.29
Llama-2-13b-chat-0-shot	28.53	14.38	29.24	19.91	1.08	21.37	26.78	20.18
COMETKiwi	16.27	**48.48**	27.90	18.05	-11.48	34.86	18.47	21.79
GPTScore-src	37.41	8.90	28.82	39.48	14.25	26.46	23.91	25.61
TIGERScore-7B (ours)	35.11	41.50	42.39	**47.11**	21.23	43.57	39.26	38.60
TIGERScore-13B (ours)	36.81	44.99	**45.88**	46.22	**23.32**	**47.03**	**46.36**	**41.52**
Δ (ours - best reference-free)	-2	-3	+12	+5	+9	+14	+13	+16
Δ (ours - best reference-based)	-9	-11	-3	+5	-0	+0	+2	+7

Table B.16: The **Spearman** correlation results of all the baseline metrics and TIGERScore on the evaluation datasets shown in Table 3.3. For each task, the metric with the highest correlation to average performance is highlighted in bold.

B.2 GenAI-Arena (Chapter 4)

B.2.1 Extra Visualization on GenAI-Arena

We included more analysis in Figure B.2 and 4.5 to show the reliability of GenAI-Arena. Specifically, Figure B.2 shows the error bar of the Elo rating to prove the reliability. For Figure 4.5, it predicts the average win rate if the model is played against other models.

(a) Text-to-Image (b) Image Editing (c) Text-to-Video

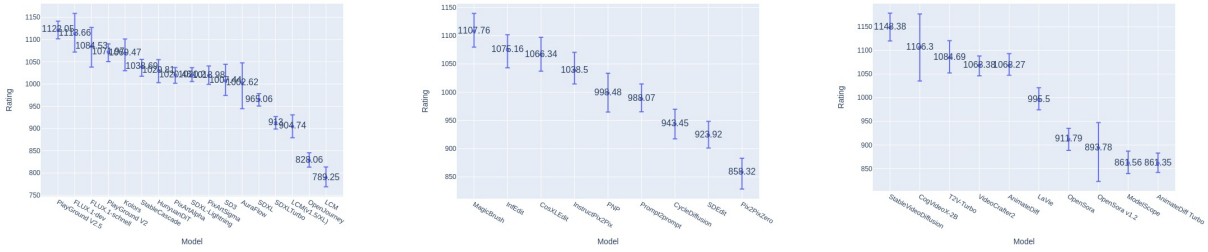


Figure B.2: Bootstrap of Elo Estimates (1000 Rounds of Random Sampling)

B.2.2 VideoGenHub

VideoGenHub is an open-source library to standardize the inference and evaluation of all the conditional video generation models, similar to ImagenHub Ku et al. (2024b) in the image domain. In the library, all models are implemented with the literature standard, and the seeds are set as 42 for a fair comparison, which is the same standard as ImagenHub Ku et al. (2024b) implementation.

B.2.3 Prompt Templates for GenAI-Bench

We provide the prompt templates used to prompt MLLM to output their preferences for the genai bench data in the followings. MLLMs are required to output 4 labels including [[A>B]], [[B>A]], [[A=B=Good]], and [[A=B=Bad]]. Videos are extracted into image frames and fed into them as an image sequence, or directly fed into the model if the model have a specific video processing unit. We then compare their output labels with the

real-world users preferences collected from out GenAI-Arena to judge a MLLM's ability in judging the quality of AI generative contents.

For text-to-image generation task, the prompt is as follows:

> Please act as an impartial judge and a professional digital artist to evaluate the quality of the responses provided by two AI image generation models to the user inputs displayed below. You will be given model A's generated image and model B's generated image. Your job is to evaluate which assistant's generated image is better.
>
> Text prompt: <prompt>
> Model A Generated Image: <left_image>
> Model B Generated Image: <right_image>
>
> When evaluating the quality of the generated images, you must identify the any inappropriateness in the edited images by considering the following criteria:
> 1. Whether the text prompt has been followed successfully in the generated image.
> 2. Whether the generated image looks natural, such as the sense of distance, shadow, and lighting.
> 3. Whether the generated image contains any artifacts, such as distortion, watermark, scratches, blurred faces, unusual body parts, or subjects not harmonized.
> 4. Whether the generated image is visually appealing and esthetically pleasing.
>
> After providing your explanation, you must output only one of the following choices as your final verdict with a label:
>
> 1. Model A is better: [[A>B]]
> 2. Model B is better: [[B>A]]
> 3. Tie, relatively the same acceptable quality: [[A=B=Good]]
> 4. Both are bad: [[A=B=Bad]]

For image-edition task, the prompt is as follows:

Please act as an impartial judge and a professional digital artist to evaluate the quality of the responses provided by two AI image edition models to the user inputs displayed below. You will be given model A's edited image and model B's edited image. Your job is to evaluate which assistant's edited image is better.

Source Image prompt: <source_prompt>
Target Image prompt after editing: <target_prompt>
Editing instruction: <instruct_prompt>
Source Image: <source_image>

Model A Edited Image: <left_output_image>
Model B Edited Image: <right_output_image>

When evaluating the quality of the edited images, you must identify the any inappropriateness in the edited images by considering the following criteria:
1. Whether the editing instruction has been followed successfully in the edited image.
2. Whether the edited image is overedited, such as the scene in the edited image is completely different from the original.
3. Whether the edited image looks natural, such as the sense of distance, shadow, and lighting.
4. Whether the edited image contains any artifacts, such as distortion, watermark, scratches, blurred faces, unusual body parts, or subjects not harmonized.
5. Whether the edited image is visually appealing and esthetically pleasing.

After providing your explanation, you must output only one of the following choices as your final verdict with a label:

1. Model A is better: [[A>B]]
2. Model B is better: [[B>A]]
3. Tie, relatively the same acceptable quality: [[A=B=Good]]
4. Both are bad: [[A=B=Bad]]

For video-generation tasks, the prompt is as follows:

Please act as an impartial judge and a professional digital artist to evaluate the quality of the responses provided by two AI video generation models to the user inputs displayed below. You will be given model A's generated video and model B's generated video. Your job is to evaluate which assistant's generated video is better.

Text prompt: <prompt>
Model A Generated Video: <left_video>
Model B Generated Video: <right_video>

When evaluating the quality of the generated videos, you must identify the any inappropriateness in the edited videos by considering the following criteria:
1. Whether the text prompt has been followed successfully in the generated video.
2. Whether the generated video looks natural, such as the sense of distance, shadow, and lighting.
3. Whether the generated video is good visual quality, such as clearness, resolution, brightness, and color.
4. Whether the generated video is consistent and coherent in terms of the scene, objects, and characters.
5. Whether the generated video is dynamic and not static like a single image.
6. Whether the generated video is visually appealing and esthetically pleasing.

After providing your explanation, you must output only one of the following choices as your final verdict with a label:

1. Model A is better: [[A>B]]
2. Model B is better: [[B>A]]
3. Tie, relatively the same acceptable quality: [[A=B=Good]]
4. Both are bad: [[A=B=Bad]]

B.3 AceCoder (Chapter 5)

B.3.1 AceCode-RM Model Breakdown

In this section, we present the different models involved in the training and evaluation process of AceCode-RM in a concise table for further clarification.

AceCodeRM-7B	
Backbone	Qwen2.5-Coder-7B-Instruct
Training Responses (Preference Pairs) Generation	Qwen2.5-Coder-7B-Instruct
Inference Model	Any Model (Qwen2.5-Coder-7B-Instruct, Llama-3.1-8B-Instruct, etc.)
AceCodeRM-32B	
Backbone	Qwen2.5-Coder-32B-Instruct
Training Responses (Preference Pairs) Generation	Qwen2.5-Coder-32B-Instruct
Inference Model	Any Model (Qwen2.5-Coder-7B-Instruct, Llama-3.1-8B-Instruct, etc.)

Table B.17: Models used during the training and evaluation process of AceCode-RM.

B.3.2 Prompt Template

system:
You are an AI assistant that helps people with python coding tasks.

user:
You are the latest and best bot aimed at transforming some code snippet into a leetcode style question. You will be provided with a prompt for writing code, along with a reference program that answers the question. Please complete the following for me:
1. Come up with a leetcode style question which consists of a well-defined problem. The generated question should meet the following criteria:
   a. The question is clear and understandable, with enough details to describe what the input and output are.
   b. The question should be solvable by only implementing 1 function instead of multiple functions or a class. Therefore, please avoid questions which require complicated pipelines.
   c. The question itself should not require any access to external resource or database.
   d. Feel free to use part of the original question if necessary. Moreover, please do not ask for runtime and space complexity analysis or any test cases in your response.
2. Based on the modified question that you generated in part 1, you need to create around 20 test cases for this modified question. Each test case should be independent assert clauses. The parameters and expected output of each test case should all be constants, **without accessing any external resources**.

Here is the original question:
{instruction}

Here is the reference program that answers the question:
```python
{program}
```

Now give your modified question and generated test cases in the following json format:
{"question": ..., "tests":["assert ...", "assert ..."]}.

Table B.18: Prompt Used for Converting Seed Code Dataset into LeetCode-style Questions and Test Cases

system: You are an AI assistant that helps people with python coding tasks.
user: You are the latest and best bot aimed at transforming some code snippet into a leetcode style question. You will be provided with a reference program. Please complete the following for me: 1. Come up with a leetcode style question which consists of a well-defined problem. The generated question should meet the following criteria: a. The question is clear and understandable, with enough details to describe what the input and output are. b. The question should be solvable by only implementing 1 function instead of multiple functions or a class. Therefore, please avoid questions which require complicated pipelines. c. The question itself should not require any access to external resource or database. d. Feel free to use part of the original question if necessary. Moreover, please do not ask for runtime and space complexity analysis or any test cases in your response. 2. Based on the modified question that you generated in part 1, you need to create around 20 test cases for this modified question. Each test case should be independent assert clauses. The parameters and expected output of each test case should all be constants, **without accessing any external resources**. Here is the reference program: ‘‘‘python {program} ‘‘‘

Table B.19: Prompt Used for Converting Seed Code Dataset using only the reference program without instruction into LeetCode-style Questions and Test Cases

B.4 VerlTool (Chapter 6)

B.4.1 Detailed Experiment Setup

We evaluate VerlTool across six diverse domains to demonstrate its effectiveness in tool-augmented reasoning. Each task domain presents unique challenges and requires different tool integration strategies, allowing us to comprehensively assess the framework's adaptability and performance.

B.4.2 Mathematical Reasoning with Python Executor (VT-Math)

Mathematical reasoning tasks often involve complex computations that are prone to numerical errors when performed purely through natural language reasoning. To address this limitation, we integrate a Python code interpreter tool that enables agents to execute math-

ematical calculations reliably and verify intermediate results. We train a mathematical-coding agent that issues Python snippets to a sandboxed interpreter and processes execution traces.

We use DeepMath (He et al., 2026b) as our training dataset. The reward function combines answer accuracy with tool usage incentives:

$$R_{\text{acc}}(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if match}(\mathbf{y}, \mathbf{y}_g) \\ -1 & \text{otherwise} \end{cases}, \quad R_{\text{tool}}(\mathbf{x}, \mathbf{y}) = \begin{cases} 0 & \text{if use_tool}(\mathbf{y}) \\ -0.25 & \text{otherwise} \end{cases} \tag{B.1}$$

where the final reward is $R_{\text{math}} = R_{\text{acc}}(\mathbf{x}, \mathbf{y}) + R_{\text{tool}}(\mathbf{x}, \mathbf{y})$. This design encourages the model to explore Python executor usage for problem-solving while maintaining an accuracy focus.

We evaluate on multiple mathematical benchmarks: MATH-500 (Hendrycks et al., 2021b), OLYMPIAD (He et al., 2024a), MINERVA (Lewkowycz et al., 2022), GSM8K (Cobbe et al., 2021), AMC, AIME24, and AIME25, using MATH-EVALUATION-HARNESS[1] for standardized eval.

B.4.3 Knowledge QA with Search Retriever (VT-Search)

Question answering tasks often require access to external knowledge beyond the model's parametric memory, particularly for factual queries and multi-hop reasoning. We integrate a FAISS-based search retriever tool that enables agents to query a local knowledge base and extract relevant information for answering complex questions.

Following prior work (Jin et al., 2025; Song et al., 2025), we integrate an E5 retriever (Wang et al., 2022) and index the 2018 Wikipedia dump (Karpukhin et al., 2020). The agent alternates between search operations and reasoning steps to construct comprehensive answers.

For this task, we apply accuracy as the primary reward:

$$R_{\text{search}}(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if match}(\mathbf{y}, \mathbf{y}_g) \\ -1 & \text{otherwise} \end{cases} \tag{B.2}$$

We evaluate using Exact Match scores on General Q&A benchmarks (NQ (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017), PopQA (Mallen et al., 2022)) and Multi-hop Q&A benchmarks (HotpotQA (Yang et al., 2018), 2Wiki (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), Bamboogle (Press et al., 2022)).

[1] `https://github.com/ZubinGou/math-evaluation-harness`

B.4.4 Multi-Turn SQL Query Generation (VT-SQL)

Natural language-to-SQL (NL2SQL) conversion requires understanding database schemas and translating natural language queries into executable SQL commands. This task benefits from tool integration as it allows iterative query refinement based on execution feedback and error correction.

We assess SQL Executor adaptability using the SkyRL-SQL training set (Liu et al., 2025a) with Qwen2.5-Coder-7B-Instruct as the base model. The agent translates natural language questions into executable SQL, given schema hints and tool-calling instructions.

The reward function focuses solely on execution accuracy:

$$R_{\text{sql}}(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if match}(\mathbf{y}, \mathbf{y}_g) \\ -1 & \text{otherwise} \end{cases} \tag{B.3}$$

Following standard conventions, we evaluate execution accuracy (EX) on SPIDER-1.0 (Yu et al., 2018) (Dev and Test splits), SPIDER-DK (Gan et al., 2021b), and SPIDER-SYN (Gan et al., 2021a).

B.4.5 Visual Reasoning with Image Operations (VT-VisualReasoner)

Traditional visual reasoning tasks are conducted primarily in the text modality, where models cannot dynamically process images as actions. To address this limitation, we implement image operation tools that enable agents to zoom into specific image regions, select key frames, and perform other visual manipulations to enhance reasoning over dense visual information, following the Pixel-Reasoner (Su et al., 2025) approach.

The reward design incorporates both accuracy-oriented and compositional complexity measures:

$$R_{\text{visualreasoner}}(\mathbf{x}, \mathbf{y}) = r(\mathbf{x}, \mathbf{y}) + \alpha \cdot r_{\text{curiosity}}(\mathbf{x}, \mathbf{y}) + \beta \cdot r_{\text{penalty}}(\mathbf{y}), \tag{B.4}$$

$$\text{where} \quad r_{\text{curiosity}}(\mathbf{x}, \mathbf{y}) = \max(H - \text{RaPR}(\mathbf{x}), 0) \cdot \mathbf{1}_{\text{PR}}(\mathbf{y}) \tag{B.5}$$

$$r_{\text{penalty}}(\mathbf{y}) = \min(N - \mathbf{n}_{\text{vo}}(\mathbf{y}), 0) \tag{B.6}$$

where $\text{RaPR}(\mathbf{x})$ denotes the ratio of responses that invoke tool calls and $\mathbf{n}_{\text{vo}}(\mathbf{y})$ denotes the number of actions that response $\mathbf{y}$ invokes. Hyperparameters are set as $H = 0.3$, $N = 1$,

$\alpha = 0.5$ and $\beta = 0.05$. We train two variants using accuracy-only reward and the original complexity-driven reward, denoted as "GRPO-acc" and "GRPO-complex" respectively.

We use the official training dataset from Pixel-Reasoner and evaluate primarily on V-Star (Wu and Xie, 2024), which assesses MLLM visual search capabilities.

B.4.6 Agentic Web Search (VT-DeeepSearch)

Open-web question answering requires real-time information retrieval and multi-step reasoning over diverse web sources. GAIA (Mialon et al., 2024) and HLE (Phan et al., 2025) are representative benchmarks testing these capabilities. We implement a Web Search tool using Google Search API through SERPER with caching, enabling agents to perform dynamic information gathering and synthesis from online sources.

We apply both accuracy and tool-usage rewards to encourage effective search behavior:

$$\begin{aligned} R_{\text{deepsearch}}(\mathbf{x}, \mathbf{y}) &= R_{\text{acc}}(\mathbf{x}, \mathbf{y}) + R_{\text{tool}}(\mathbf{x}, \mathbf{y}), \\ \text{where } R_{\text{tool}}(\mathbf{x}, \mathbf{y}) &= \begin{cases} 0.1, & \text{if tool is called} \\ 0, & \text{if no tool call} \end{cases} \end{aligned} \tag{B.7}$$

We use 1K mixed training examples from SimpleDeepSearcher (Sun et al., 2025) and Web-Sailor (Li et al., 2025b), following the setting in Dong et al. (2026). Starting from Qwen3-8B, we evaluate on GAIA and HLE (text-only) benchmarks. We retrieve top-k URLs for each query and use the returned snippets as content during RL training. For evaluation, we employ two settings: "Snippet-Only" aligns with training conditions using only snippet content, while "QwQ-32B" uses a browser agent to summarize raw content from retrieved URLs.

B.4.7 Software Engineering Benchmark (VT-SWE)

Software engineering tasks require code understanding, localization, debugging, and modification capabilities that benefit from iterative execution and testing. We integrate bash terminal and code execution tools to enable agents to interact with software development environments effectively.

We build on the R2E-Gym scaffold (Jain et al., 2025b) and its training dataset R2E-Lite, using Qwen3-8B in no-think mode as the base model. The reward function is defined

strictly by task completion accuracy: an agent must terminate normally and pass all verification tests to receive a reward of 1; otherwise, the reward is 0:

$$R_{\text{swe}}(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if execution terminates successfully and all tests pass} \\ 0 & \text{otherwise} \end{cases} \tag{B.8}$$

We evaluate performance on the SWE-Verified benchmark, measuring the agent's ability to resolve software engineering tasks and pass verification tests.

For training and evaluation, we maintain a cluster of eight servers (each with 64 CPU cores and 200 GB memory), orchestrating sandbox services via Kubernetes. Each task runs inside the official Docker image provided by R2E-Lite, initialized with 1 CPU and 2 GB memory, elastically scalable up to 2 CPUs and 4 GB memory. We observed that the main bottleneck lies in disk I/O during Docker initialization. To stabilize training, we therefore allocate more CPU and memory resources than are minimally required.

The modular architecture of VerlTool, which separates training from environment services, allows us to scale sandbox environments efficiently. Each environment interaction is given a 90-second timeout, reward computation has a 300-second timeout, and the maximum time per trajectory is capped at 20 minutes. Any trajectory that times out, encounters an exception, or exceeds the length limit is assigned a reward of 0, and its gradients are masked during updates.

B.4.8 Supported Tools

Table B.20: Currently supported tools of VerlTool framework.

Tools	Description	Related works
Code Interpreter	Execute Python code	ToRL (Li et al., 2025c)
Faiss Search	Vector similarity search for documents	Search-R1 (Jin et al., 2025)
Web Search API	Real-time web search and retrieval	SimpleDeepSearch (Sun et al., 2025)
Image Processing	Image resize, video frame selection	PixelReasoner (Su et al., 2025)
Bash Terminal	Execute shell commands	R2E-Gym (Jain et al., 2025b)
SQL Executor	Database queries and data management	SkySQL (Liu et al., 2025a)
MCP Interface	Model Context Protocol for external tool	ToolRL (Feng et al., 2026)

Appendix C

Supplementary Analyses of Multi-Turn Training

This appendix collects tokenization and trajectory-masking experiments conducted during review of VerlTool. They supplement the framework and feedback-design experiments in Chapter 6.

C.1 Tokenization Stability

Section 6.3.2 describes why the framework preserves policy-generated token IDs when appending tool observations. A supplementary SQL experiment compares this strategy with retokenizing the concatenated text (Figure C.1). The review experiments reported a token-ID mismatch of 98.6% under retokenization and a 132.9% increase in average gradient norm. These results support preserving emitted token IDs to maintain alignment between sampled actions and the policy objective.

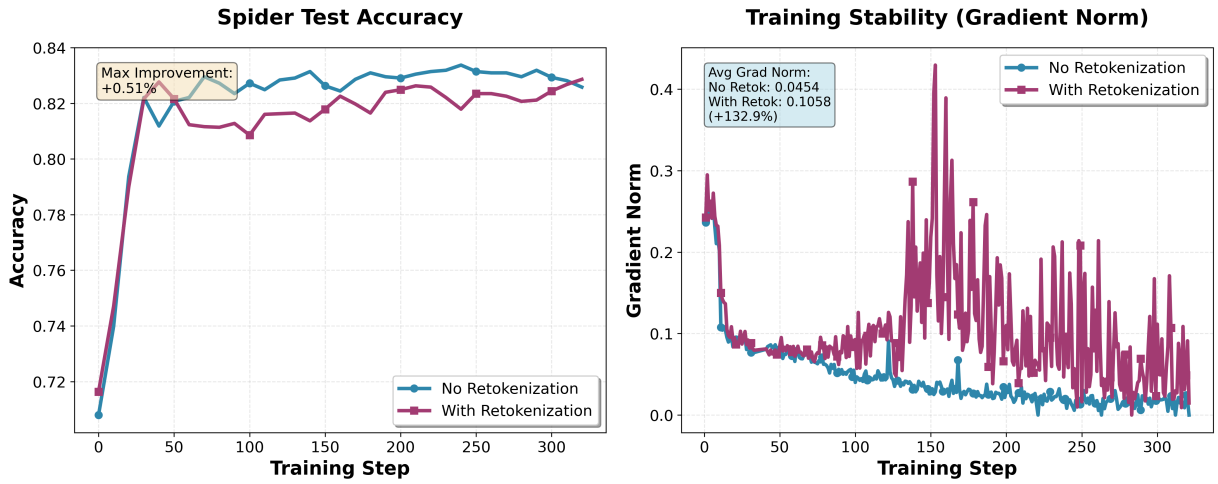


Figure C.1: Tokenization comparison from the review experiments of VERLTOOL. Left: Spider test accuracy. Right: gradient norm. Retokenizing the concatenated text increases the average gradient norm by 132.9% relative to preserving policy-generated token IDs.

C.2 Trajectory Masking

We also compare Math-TIR training with and without masking erroneous trajectories, following SimpleTIR (Xue et al., 2026). Masking excludes these trajectories from the policy loss. Figure C.2 shows the AIME25 accuracy curves and the fraction of masked trajectories from the review experiments.

The masked run ends at a higher accuracy in the plotted comparison, although the curves overlap during training. The masked fraction initially falls and later rises to around 20%. This comparison illustrates the effect of changing which trajectories contribute to an update, but it does not isolate the effects of policy errors and infrastructure failures. The VT-SWE timeout and masking settings are described separately in Section B.4.7.

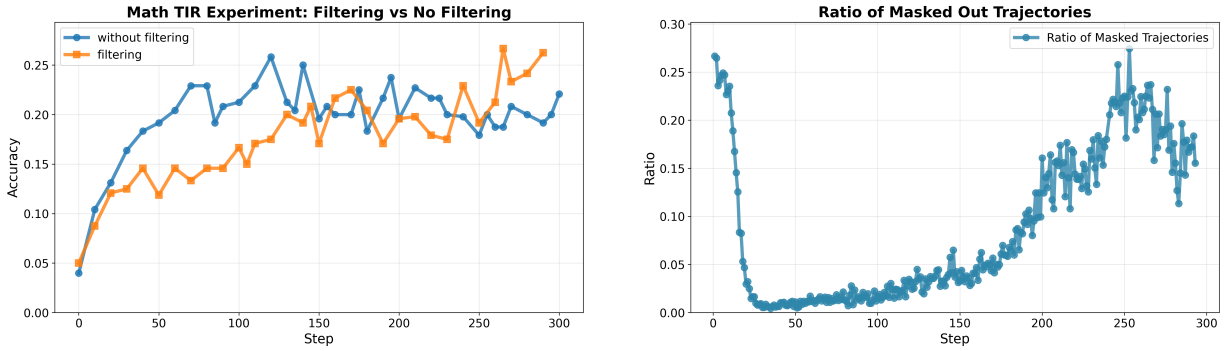


Figure C.2: Trajectory masking in Math-TIR, reproduced from the review experiments of VerlTool. Left: AIME25 accuracy (avg@8) with and without masking failed trajectories. Right: share of trajectories masked at each step.