# Distilling Legal Language Models with Limited Data through Synthesis and Attribution

by

Alireza Arbabi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2026

© Alireza Arbabi 2026

## Author's Declaration

I hereby declare that I am the sole author of this 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.

## Abstract

Large language models are increasingly capable on specialized legal NLP tasks, but their direct use incurs recurring inference costs and may require proprietary or confidential data to be shared with external providers, creating practical barriers for sensitive legal applications. In this setting, knowledge distillation provides a natural approach for building small and specialized language models. However, conventional distillation methods are limited when only a sparse set of real task examples is available, a common challenge in the legal domain.

To address this challenge, this thesis presents **LegalSynth**, a framework that uses **synthetic data generation** followed by **attribution-based data selection** as an alternative channel for transferring knowledge from the teacher to the student. Given a task description and a small set of real examples, LegalSynth first prompts a teacher model to infer the task structure from the available task information and then leverages this understanding to generate a large and diverse pool of synthetic training examples beyond the initial seed set. Because synthetic examples vary in quality and utility, LegalSynth then applies data attribution to filter the synthetic pool and retain the examples that are most useful for training, improving the performance of the student model. Experiments across four low-resource, multi-way legal classification tasks from LegalBench demonstrate the effectiveness of this framework, enabling Qwen3.5 models with up to 4B parameters to compete with, and in several settings outperform, frontier language models including GPT-5.6-Terra, Gemini-3.7-Flash, and Qwen3.5-397B.

**Acknowledgements**

I would like to express my sincere gratitude to my supervisor, Dr. Xi He, for her constant support, care, thoughtful mentorship, and for giving me the opportunity to work on insightful projects under her supervision. Her guidance, encouragement, and the opportunities she has provided have had a significant impact on my academic and professional development, and I will always be grateful for everything she has done for me and for the trust she placed in me throughout my master's studies.

I would also like to thank Dr. Ihab Ilyas for his support and supervision, for introducing the original idea that led to this thesis, and for teaching me the importance of paying attention to every detail. I am also grateful to Dr. Florian Kerschbaum for supporting me during my first research paper in my master's studies on AI proprietary alignment. I would further like to thank Dr. Kerschbaum and Dr. Charles Clarke for dedicating their time and attention to reading and evaluating this thesis.

Finally, I would like to thank Hedi Driss, Wei Pang, Dr. George Beskales, and all my friends and collaborators for their support and encouragement throughout my master's studies.

## Dedication

I dedicate this work to my little sister, Mehrnoosh, and my parents, Nooshin and Ahmad.

# Table of Contents

**Author's Declaration** ii

**Abstract** iii

**Acknowledgements** iv

**Dedication** v

**List of Figures** ix

**List of Tables** xii

**1 Introduction** 1

**2 Background and Related Work** 4

2.1 Legal Natural Language Processing . . . . . 4

2.1.1 Legal NLP and its Challenges . . . . . 4

2.1.2 Legal NLP Tasks and Legal Text Classification . . . . . 5

2.1.3 Legal Language Models and Benchmarks . . . . . 5

2.2 Knowledge Distillation . . . . . 6

2.2.1 General Form of Knowledge Distillation . . . . . 6

2.2.2 Synthetic Data as a Distillation Channel in Data-Scarce Settings . 8

2.3 Data Attribution and Selection . . . 9
2.3.1 Early Approaches to Data Attribution: Influence Functions, Shapley Values, TracIn, and Datamodels . . . 10
2.3.2 TRAK: Attributing Model Behavior at Scale . . . 11
2.4 Supervised Fine-Tuning and LoRA . . . 13

**3 Framework and Methodology** **15**
3.1 Problem Setup and Data Organization . . . 15
3.1.1 LegalBench Task Structure . . . 17
3.1.2 Train, Validation, and Test Splits . . . 17
3.2 Synthetic Data Generation . . . 18
3.2.1 Multi-Step Task-Aware Synthesis . . . 18
3.2.2 Synthetic Augmentation . . . 20
3.3 Data Selection via Attribution . . . 21
3.3.1 Attribution Setup . . . 21
3.3.2 Checkpoint and Projection Ensembling . . . 21
3.3.3 Selection Policies . . . 22
3.3.4 Adapting TRAK to Qwen 3.5 . . . 22
3.3.5 Data Selection Procedure . . . 23
3.4 Student Model Fine-Tuning . . . 23
3.4.1 Question-Answer Format and Training Objective . . . 23
3.4.2 Adaptation and Early Stopping . . . 25

**4 Evaluation** **26**
4.1 Experimental Setup . . . 26
4.1.1 Tasks . . . 26
4.1.2 Settings . . . 32
4.2 Results . . . 35
4.2.1 Main Results . . . 35
4.2.2 Benchmarking Synthesis Prompt Strategies via Attribution Analysis 57

**5 Discussion** **68**

5.1 Student Models Outperforming Their Teachers . . . . . 68

5.2 High-Recall, Low-Precision Nature of Weakly-Guided Synthetic Data and the Motivation for Attribution Analysis . . . . . 69

5.3 Cost, Efficiency, and Privacy . . . . . 70

5.4 Implications for Legal NLP . . . . . 71

**6 Conclusion** **72**

6.1 Summary of Findings . . . . . 72

6.2 Limitations and Future Work . . . . . 73

**References** **75**

**Appendix** **79**

**A Synthetic Data Generation Prompts** **80**

A.1 Static Prompt Templates . . . . . 80

A.1.1 Paraphrasing . . . . . 80

A.1.2 Multi-Step Phase 1: per-class analysis . . . . . 82

A.1.3 Multi-Step Phase 2a: class-boundary analysis . . . . . 83

A.1.4 Multi-Step Phase 2b: content-domain coverage . . . . . 84

A.1.5 Multi-Step Phase 2c: rhetorical-register analysis . . . . . 85

A.1.6 Multi-Step Phase 3: generation-prompt design . . . . . 87

A.1.7 Multi-Step Phase 4: generation assembly fragments . . . . . 90

# List of Figures


3.1 End-to-end overview of the LegalSynth framework. Task-aware synthesis produces a broad candidate pool, TRAK attribution ranks candidates by their estimated effect on a held-out validation set, and the selected subset is used to fine-tune compact student language models. . . . . 16

4.1 Abercrombie: test macro-F1 by student size. . . . . 38

4.2 Abercrombie, Qwen3.5-0.8B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies. . . . . 39

4.3 Abercrombie, Qwen3.5-2B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies. . . . . 40

4.4 Abercrombie, Qwen3.5-4B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies. . . . . 41

4.5 Insurance Policy Interpretation: test macro-F1 by student size. . . . . 43

4.6 Insurance Policy Interpretation, Qwen3.5-0.8B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies. . . 44

4.7 Insurance Policy Interpretation, Qwen3.5-2B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies. . . . . 45

4.8 Insurance Policy Interpretation, Qwen3.5-4B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies. . . . . 46

4.9 Oral Argument Question Purpose: test macro-F1 by student size. . . . . 48

4.10 Oral Argument Question Purpose, Qwen3.5-0.8B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies. . . 49

4.11 Oral Argument Question Purpose, Qwen3.5-2B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies. . . 50

4.12 Oral Argument Question Purpose, Qwen3.5-4B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies. . . 51

4.13 Function of Decision Section: test macro-F1 by student size. . . . . 53

4.14 Function of Decision Section, Qwen3.5-0.8B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies. . . . . 54

4.15 Function of Decision Section, Qwen3.5-2B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies. . . . . 55

4.16 Function of Decision Section, Qwen3.5-4B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies. . . . . 56

4.17 Composition of each synthetic training pool by the sign of its mean normalized attribution across validation examples. Zero denotes an exactly zero mean; no near-zero threshold is applied. Percentages and sample counts are shown within each segment. . . . . 58

4.18 Distribution of average normalized TRAK scores for each synthesis prompt strategy. Boxes show the interquartile range, centre lines show the median, diamonds show the mean, and whiskers span the 10th to 90th percentiles. Annotations report the negative, exactly zero, and positive shares. . . . . 61

4.19 Relative ranking of positive-attribution examples by synthesis prompt strategy. Within each task, examples with positive raw mean attribution are pooled and ranked from highest to lowest. At each top-$N$ cutoff, each curve reports the fraction of that strategy's positive examples captured. The dotted line is the expected curve under a uniform ranking. . . . . 62

4.20 Attribution by prompt strategy and class for Abercrombie. The left panel shows mean positive support, while the right panel shows net signed support. 63

4.21 Attribution by prompt strategy and class for Insurance Policy Interpretation. The left panel shows mean positive support, while the right panel shows net signed support. . . . . 64

4.22 Attribution by prompt strategy and class for Oral Argument Question Purpose. The left panel shows mean positive support, while the right panel shows net signed support. . . . . 65

4.23 Attribution by prompt strategy and class for Function of Decision Section. The left panel shows mean positive support, while the right panel shows net signed support. . . . . 65

4.24 Class–class attribution sign composition across the four tasks. Rows denote the labels of synthetic candidates and columns denote the labels of real validation examples. Cell annotations report the negative, exactly zero, and positive attribution rates, while colour represents the positive rate minus the negative rate. . . . . 67

# List of Tables


4.1 Abercrombie: label distribution across the full real dataset ($n = 100$). . . . 28

4.2 Insurance Policy Interpretation: label distribution across the full real dataset ($n = 138$). . . . 29

4.3 Oral Argument Question Purpose: label distribution across the full real dataset ($n = 319$). . . . 31

4.4 Function of Decision Section: label distribution across the full real dataset ($n = 374$). . . . 33

4.5 Number of synthetic examples generated per method for each task. Contrastive counts are summed over the three hardest label pairs identified during cross-class analysis. . . . 33

4.6 Teacher and student results on the *Abercrombie* task. . . . 37

4.7 Teacher and student results on the *Insurance Policy Interpretation* task. . . . 42

4.8 Teacher and student results on the *Oral Argument Question Purpose* task. 47

4.9 Teacher and student results on the *Function of Decision Section* task. . . . 52

A.1 Literal values substituted for `{{difficulty}}` and `{{guidance}}` in the paraphrase user prompt, by tier. . . . 81

# Chapter 1

# Introduction

Large Language Models (LLMs) have demonstrated strong capabilities across a wide range of natural language processing tasks [28, 2, 8, 7] and are increasingly being applied to specialized domains such as law [13, 4, 1]. Legal applications include question answering, document analysis, information extraction, summarization, and text classification, where models must interpret complex domain-specific language and distinguish between closely related legal concepts [1, 13]. These capabilities make LLMs attractive for legal NLP, particularly in settings where accurate understanding of specialized text is required.

However, relying directly on large proprietary LLMs is not always practical for legal applications. Many legal workloads involve confidential or proprietary documents that organizations may be unwilling or unable to send to an external model provider [25]. At the same time, repeatedly querying large models introduces a recurring inference cost that can become substantial at scale. These considerations motivate the use of smaller, locally deployable language models that can process sensitive data in-house at substantially lower inference cost. The main difficulty is that small models generally have much weaker task performance than large frontier models without task-specific adaptation.

Knowledge distillation therefore provides a natural mechanism for transferring capability from a stronger teacher model to a smaller student [15, 12, 33]. In a conventional distillation setting, a teacher provides supervision over a collection of task-relevant training inputs, and the resulting labelled examples are used to train the student. This setup, however, assumes that a sufficiently informative collection of training inputs already exists. In low-resource legal tasks, this assumption may not hold [1]. Even when the teacher can provide the labels, a small set of real examples may cover only a limited portion of the target task distribution. The resulting problem is therefore not only a shortage of labels,

but a shortage of task-relevant training inputs over which knowledge can be transferred.

Synthetic data generation provides an alternative distillation channel for this setting [35, 36, 34]. Rather than using the teacher only to label existing inputs, a capable LLM can be conditioned on the task description and a small set of real seed examples and asked to generate entirely new input–label pairs. In this way, the teacher's broad pretrained knowledge can be used to expand a limited set of real observations into a substantially larger task-specific training pool. The goal is to move beyond the limited coverage of the available examples by exposing the student to additional class boundaries, subject-matter domains, linguistic forms, and other variations that may be absent from the scarce real dataset.

The resulting synthetic pool, however, should not be assumed to form a reliable training set by itself. When generation is guided by only limited information about the true task distribution, synthetic examples can vary substantially in usefulness: some may provide strong supervision, while others may be ambiguous, redundant, incorrectly labelled, poorly aligned with the real data, or even harmful to the student. Synthetic generation can therefore be viewed as a **high-recall, low-precision** process that prioritizes broad coverage before filtering [20, 9]. This creates a second problem: determining which generated examples should actually be used for training. Many filtering approaches assess synthetic samples in isolation, for example through similarity measures, embeddings, or a separate LLM judge [30, 17]. However, such standalone filtering methods have shown limited utility [9], as for student training the optimal filtering procedure should ultimately be model-dependent: whether a particular synthetic example helps the intended student model behave correctly on real target examples, rather than whether the sample appears useful when analyzed in isolation. This motivates the use of **data attribution** [19, 10, 18, 23], which estimates the contribution of individual training examples to model behavior on a target set and can therefore provide a direct signal for synthetic data selection. Taken together, synthesis and attribution provide complementary mechanisms for the low-data setting: synthesis expands the available task-specific supervision, while attribution identifies which parts of that expanded pool are most useful for training the student.

Building on this idea, in this thesis we introduce **LegalSynth**, a framework for language-model distillation under data scarcity that combines task-aware synthetic data generation, model-aware data attribution, and supervised fine-tuning of small language models on legal text classification. LegalSynth first constructs a broad synthetic candidate pool using multiple complementary generation strategies. The candidates are then scored using TRAK attribution [23] with respect to held-out real examples, and the highest-ranked subsets are used to fine-tune small Qwen3.5 (0.8B, 2B, and 4B) [26] student models. We evaluate the framework on four low-resource, multi-way legal text-classification tasks from LegalBench

[13]. The results show that attribution-based selection generally produces stronger students than training on the full synthetic pool, and that compact 0.8B–4B parameter models can become competitive with, and in several settings outperform, substantially larger frontier language models.

The main contributions of this thesis are:

- We present a systematic framework for **knowledge distillation under task-data scarcity**, where synthetic data generation is used to expand the available training inputs beyond a limited set of real seed examples.
- We develop a **multi-step, task-aware synthesis procedure** that analyzes class characteristics, difficult class boundaries, missing content domains, and underrepresented rhetorical forms to automatically craft varied prompts for large-scale generation, with the goal of constructing a broad candidate training pool.
- We demonstrate the use of **TRAK data attribution for model-aware synthetic-data selection and analysis**, allowing generated examples to be ranked according to their estimated usefulness for the target student model on held-out real data.
- We demonstrate across four LegalBench classification tasks that **small specialized language models can match or outperform much larger general-purpose models** in several evaluation settings. Across the twelve teacher–task combinations per prompting regime, the best student with validation-set attribution and subset size $N$ chosen on the validation set surpasses seven zero-shot baselines, five few-shot baselines, and one many-shot baseline. Choosing $N$ on the test set instead raises these counts to nine, seven, and two; using test-set attribution as well yields ten, eight, and five.

# Chapter 2

# Background and Related Work

## 2.1 Legal Natural Language Processing

### 2.1.1 Legal NLP and its Challenges

A legal system is the framework through which a society creates, interprets, and enforces rules governing the rights, obligations, and conduct of individuals, organizations, and public institutions. These rules are generally expressed through several types of legal documents, each serving a different role. For instance, statutes and regulations establish legal rules and requirements, contracts define rights and obligations between parties, and judicial decisions apply and interpret legal rules in particular disputes. In common-law systems, judicial decisions also play a broader role by serving as precedents: courts refer to the reasoning and legal principles established in earlier cases when deciding later disputes involving similar issues. Legal briefs and arguments, in turn, present the facts, authorities, and reasoning used by parties to support their positions within this process. Consequently, legal text presents several characteristics that make its automated processing substantially different from ordinary general-domain NLP systems [1].

Legal text presents several characteristics that make it challenging for NLP systems. The language itself is often specialized and syntactically complex, with domain-specific terminology and distinctions whose meaning can depend on small differences in wording [1, 11]. Legal documents are also highly structured and interdependent: statutes refer to other provisions, judicial opinions rely on earlier precedents, and contracts often define terms through other clauses in the same document. At the same time, legal corpora are heterogeneous, ranging from short contractual clauses to long judicial opinions, regulations,

and administrative records, each with different structures and conventions. As a result, correctly interpreting a legal passage may require understanding both its local wording and its broader document and legal context [1].

A second challenge is the combination of a high cost of error and limited access to high-quality training data. Legal NLP systems may be used in settings involving contractual rights, financial obligations, regulatory compliance, or judicial decisions, where incorrect predictions can have substantial consequences [1]. At the same time, producing reliable labelled legal data is often expensive because annotation may require legal expertise, while many practical datasets are small, confidential, or proprietary. This makes reliable task-specific adaptation particularly difficult when only limited labelled data are available.

### 2.1.2 Legal NLP Tasks and Legal Text Classification

Legal NLP spans a broad range of tasks that reflect different ways of working with legal documents and legal information. Common examples include legal question answering, where a system answers questions based on statutes, cases, or other legal sources; legal judgment prediction, where models predict outcomes or decisions from case information; legal document summarization, which condenses long legal texts into smaller representations without losing the important information; named entity recognition, which identifies entities such as courts, statutes, parties, and judges; argument mining, which extracts claims, premises, and their relationships; and legal text classification, where a document, passage, or clause is assigned to one or more predefined categories [1]. In this thesis, we focus specifically on legal text classification, particularly multi-way classification tasks in which a legal passage must be assigned to one label from a fixed set of categories.

### 2.1.3 Legal Language Models and Benchmarks

Pre-trained language models have become the dominant foundation for many legal NLP systems, but general-domain models are not always well adapted to the terminology, structure, and reasoning patterns of legal text. This has motivated the development of legal-domain models through continued pre-training, domain-specific pre-training, and instruction tuning, like LEGAL-BERT [4], SaulLM [6]. In parallel, benchmark and datasets such as LexGLUE [5], LEXTREME [21], CUAD [14], MAUD [29] and LegalBench [13] have been introduced to evaluate legal language understanding across different tasks, jurisdictions, and languages.

This thesis uses LegalBench [13], a collaborative benchmark designed to evaluate legal reasoning in large language models. LegalBench contains 162 tasks spanning a broad range of legal abilities, including information extraction, rule application, interpretation, and classification. Its tasks are derived from realistic legal reasoning problems and cover several areas of law and forms of legal analysis. In this work, we focus on four of its multi-way text-classification tasks with data scarcity, which are described in detail in Chapter 4.

## 2.2 Knowledge Distillation

### 2.2.1 General Form of Knowledge Distillation

Knowledge distillation refers to the transfer of knowledge from a typically larger or more capable *teacher* model to a smaller *student* model. Traditionally, distillation has been motivated primarily by model compression, as the objective is to retain as much of the teacher's predictive behavior on the target task as possible while reducing the computational and memory requirements of the student model [15, 12, 33, 3].

Knowledge can be transferred from the teacher in several forms, depending on what information is available during training. In *response-based distillation*, the student learns from the teacher's final outputs, either as a probability distribution (*soft-label distillation*) or as the teacher's final prediction label (*hard-label distillation*). In *feature-based distillation*, the student is trained to reproduce intermediate representations of the teacher model, while *relation-based distillation* transfers relationships among different parts of the teacher/student models, including layers, training examples, and outputs, rather than relying solely on the output predictions [12]. These approaches differ in the information transferred from the teacher, but share the same general goal on reproducing useful aspects of the teacher's behavior in a smaller and more efficient model.

To illustrate response-based distillation on a given neural network, consider a classification problem with $K$ possible classes. Let $T$ and $S$ denote the teacher and student models, respectively. For an input $x$, the teacher produces a vector of logits

$$\mathbf{z}^T(x) = \left[z_1^T(x), z_2^T(x), \ldots, z_K^T(x)\right]. \tag{2.1}$$

These logits can be transformed into a probability distribution using a temperature-scaled softmax function,

$$p_T^{(\tau)}(y = k \mid x) = \frac{\exp\left(z_k^T(x)/\tau\right)}{\sum_{j=1}^{K} \exp\left(z_j^T(x)/\tau\right)}, \tag{2.2}$$

where $\tau > 0$ denotes the temperature parameter. At $\tau = 1$, the expression reduces to the standard softmax function, while larger values of $\tau$ produce a softer probability distribution by assigning more probability mass to non-maximal classes. These probabilities can reveal information that is not contained in the final predicted class alone. For example, the probabilities assigned to alternative classes indicate which predictions the teacher considers relatively plausible.

In *soft-label distillation*, the student is trained to approximate the teacher model's output distribution on the given input data samples. A common objective minimizes the divergence between the teacher and student predictions,

$$\mathcal{L}_{\text{soft}} = D_{\text{KL}}\left(p_T^{(\tau)}(\cdot \mid x) \,\|\, p_S^{(\tau)}(\cdot \mid x)\right), \tag{2.3}$$

where $p_S^{(\tau)}$ denotes the corresponding distribution produced by the student model [15]. In this way, the student learns not only the teacher's preferred class, but also information encoded in the relative probabilities assigned to competing classes.

In *hard-label distillation*, only the teacher's final prediction is retained. The hard target can be expressed as

$$\hat{y}^T = \arg\max_k p_T(y = k \mid x), \tag{2.4}$$

and the student is then trained on this teacher-generated target using the standard cross-entropy objective,

$$\mathcal{L}_{\text{hard}} = -\log p_S(\hat{y}^T \mid x). \tag{2.5}$$

Thus, whereas soft-label distillation transfers the complete output distribution of the teacher, hard-label distillation retains only its final decision. The distinction is particularly important in practice because soft-label distillation requires access to the teacher's output layer distributions, whereas hard-label distillation requires only its final outputs.

Feature-based and relation-based approaches provide additional mechanisms for transferring teacher knowledge. Feature-based methods align intermediate representations between the teacher and student, while relation-based methods transfer structural relationships among representations or training examples [12].

### 2.2.2 Synthetic Data as a Distillation Channel in Data-Scarce Settings

The distillation approaches described above generally assume the availability of a sufficiently informative set of unlabeled training data from the target task to begin the distillation. Although these training inputs do not necessarily require human-provided labels because the teacher performs the labeling, the inputs themselves must still be available. In conventional distillation, the teacher therefore transfers its knowledge over an existing set of task-relevant training samples. When the available training data are scarce and provide only limited coverage of the target task distribution, this process becomes constrained: the teacher can provide richer supervision for the available samples, but it cannot provide supervision for task situations that are absent from the training data. Consequently, direct distillation may be insufficient when only a small number of real examples are available.

Synthetic data generation provides a natural way to address this limitation [35, 36, 34]. Rather than using the teacher only to provide supervision for an existing set of training inputs, it can be prompted to synthesize both new inputs and their corresponding labels. Let $I$ denote the task description, including information of the task and label definitions, and let $\mathcal{D}_{\text{seed}}$ denote the small set of available real training examples. A synthetic training example can then be defined as:

$$(x_i^{\text{syn}}, y_i^{\text{syn}}) \sim p_T\left(x, y \mid I, \mathcal{D}_{\text{seed}}\right), \tag{2.6}$$

where the teacher model generates a pair of input-output, conditioned on the task specification and the available seed data. Repeating this process for $M$ times produces a synthetic training dataset:

$$\mathcal{D}_{\text{syn}} = \{(x_i^{\text{syn}}, y_i^{\text{syn}})\}_{i=1}^{M}. \tag{2.7}$$

The underlying hypothesis is that the broad knowledge acquired by modern LLMs through large-scale pre-training and subsequent alignment can be elicited through appropriate prompting to produce useful task-specific training data. In a low-resource setting, a small number of real examples can therefore be used as the few-shot input of the teacher model to generate a new synthetic dataset beyond the observed samples [35, 36, 30].

Synthetic data generation can take several forms depending on how new training information is constructed. A useful distinction is between *horizontal* and *vertical* synthesis. In horizontal synthesis, the teacher generates entirely new task examples beyond the available

seed data [35, 36, 30]. In vertical synthesis, existing examples or source information are instead transformed, expanded, or made more complex in order to give more comprehensive understanding to the target fine-tuning model about the training data. This includes simple paraphrasing, evolving the examples into more complex variants [32], or restructuring information into new richer representations [34]. More recently, synthesis has also been formulated as an optimization problem: rather than specifying only how the data should be generated, the generator can be optimized directly according to its effect on a downstream model or differentiable target [27].

However, LLM synthetic data generation is generally a **high-recall, low-precision** process [20, 9]. When we prompt an LLM and ask to generate thousands of training samples, there is no initial guarantee that all synthetic samples have high quality and correct. The teacher model might understand the task incompletely or make mistake, resulting in synthetic samples to be mislabeled, ambiguous, redundant, poorly aligned with the target distribution, or even hurt the performance of the target student model. Therefore, once we generate a candidate training pool of synthetic data, it is essential to filter the useful samples for the target task. This phenomenon motivates the use of *data attribution* methods to estimate the contribution of individual training examples to the behavior of the fine-tuned model on the evaluation data, which are discussed in Section 2.3.

## 2.3 Data Attribution and Selection

The behavior of a machine learning model is shaped by its training data. However, individual training examples do not contribute equally to every model prediction: some examples may strongly support a particular behavior, while others may have little effect or may even push the model in an opposing direction. Data attribution studies this relationship by assigning an attribution score to each training example with respect to a model output of interest. More formally, given a training set

$$\mathcal{D}_{\text{train}} = \{z_1^{\text{train}}, \ldots, z_n^{\text{train}}\},$$

a target example $z_j^{\text{target}}$, and a model output $f(z_j^{\text{target}}; \theta)$, a data attribution method produces a vector

$$\tau\left(z_j^{\text{target}}, \mathcal{D}_{\text{train}}\right) \in \mathbb{R}^n,$$

where $\tau(z_j^{\text{target}}, \mathcal{D}_{\text{train}})_i$ represents the estimated influence of training example $z_i^{\text{train}}$ on the chosen model output for $z_j^{\text{target}}$. Depending on the definition of the model output, positive and negative attribution scores may respectively correspond to training examples

that support or oppose the desired model behavior. Data attribution has therefore been studied not only for explaining predictions, but also for identifying harmful or mislabeled examples and for selecting useful training data.

### 2.3.1 Early Approaches to Data Attribution: Influence Functions, Shapley Values, TracIn, and Datamodels

Early approaches to training-data attribution were motivated by classical statistical tools such as influence functions and Shapley values. Influence functions estimate how the model parameters, and consequently a target prediction, would change if the contribution of a particular training example to the training objective were slightly increased or decreased [19]. Their classical formulation combines the gradients of the target and training examples with the inverse Hessian of the training loss, providing a principled approximation to leave-one-out retraining. However, computing or approximating the Hessian of a large neural network can be expensive, and the local approximations underlying influence functions may become unreliable in highly non-convex settings. Shapley-value-based approaches instead quantify the value of a training example through its average marginal contribution over different subsets of the training data [10]. Although Shapley values provide an appealing notion of data value, estimating them generally requires training models on many subsets of the data, making direct application expensive for modern neural networks.

To obtain more scalable estimates, several methods use gradients to approximate the interaction between training and target examples. TracIn [24], for example, traces these interactions along the optimization trajectory. Given a sequence of checkpoints $\theta_t$, a commonly used form of TracIn computes

$$\tau_{\text{TracIn}}\left(z_j^{\text{target}}\right)_i = \sum_t \eta_t \nabla_\theta L(z_j^{\text{target}}; \theta_t)^\top \nabla_\theta L(z_i^{\text{train}}; \theta_t),$$

where $\eta_t$ denotes the learning rate associated with checkpoint $t$. The underlying intuition is gradient agreement: if the gradients of a training and target example point in similar directions, parameter updates induced by the training example are expected to also benefit the target example. Conversely, opposing gradients indicate a potentially detrimental interaction. This makes TracIn substantially more scalable than methods requiring repeated counterfactual retraining, although its comparison is primarily based on pairwise gradient similarity and does not explicitly account for how the same gradient directions are distributed across the entire training set.

Gradient-based influence estimation has also been adapted directly to data selection. LESS [31], for example, considers targeted selection for language-model fine-tuning, where

only a subset of a large training pool may be useful for a particular downstream task. It represents training and target examples through low-dimensional gradient features and uses their estimated influence to rank training examples according to their relevance to the target task. This establishes a direct connection between data attribution and data selection: attribution scores need not only be used retrospectively to explain a trained model, but can also be used prospectively to determine which examples should be included in training.

A different perspective is provided by *datamodels* [18]. Rather than estimating influence only from local gradient information, datamodels explicitly study how model behavior changes as the composition of the training set changes. Models are repeatedly trained on different subsets of $\mathcal{D}_{\text{train}}$, and a linear model is then fitted to predict the output on a fixed target example from an indicator vector describing which training examples were included in each subset. The resulting coefficients act as attribution scores for individual training examples. This approach directly models counterfactual training behavior and can provide accurate attribution estimates, but its main limitation is computational cost: obtaining high-quality datamodels may require training thousands or tens of thousands of models on different data subsets. TRAK later adopts this counterfactual view of attribution while seeking a substantially more scalable approximation [23].

### 2.3.2 TRAK: Attributing Model Behavior at Scale

TRAK (*Tracing with the Randomly-projected After Kernel*) [23] is designed to bridge the gap between scalable gradient-based methods and computationally expensive sampling-based approaches like datamodels. Its central idea is to approximate a nonlinear neural network (in our context, the transformer network of an LLM) with a simpler linear model around the trained parameters $\theta^*$ using Taylor expansion, and apply the linear attribution methods to the simplified representation. For a model output $f(z;\theta)$, TRAK applies the first-order Taylor approximation

$$f(z;\theta) \approx f(z;\theta^*) + \nabla_\theta f(z;\theta^*)^\top (\theta - \theta^*).$$

Under this approximation, the nonlinear model can be viewed as a linear model whose feature representation for an example $z$ is given by its gradient with respect to the model parameters,

$$g(z) = \nabla_\theta f(z;\theta^*).$$

Thus, rather than comparing examples in their original input or hidden representation space, TRAK represents each example according to the parameter directions that affect

its model output, so gradients with respect to the model parameters become features of the training and target samples.

For modern neural networks, these gradient vectors may contain millions or billions of dimensions. TRAK therefore projects them into a lower-dimensional space using

$$\phi(z) = P^\top \nabla_\theta f(z; \theta^*),$$

where $P$ is a random projection matrix. The random projection substantially reduces the computational cost while approximately preserving the inner-product structure of the original gradients. For the training set, the projected features are stacked into

$$\Phi = \begin{bmatrix} \phi(z_1^{\text{train}})^\top \\ \vdots \\ \phi(z_n^{\text{train}})^\top \end{bmatrix}.$$

For an example of interest $z$ on a single trained model/checkpoint, TRAK estimates the attribution vector with respect to the training set as

$$\tau\left(z, \mathcal{D}_{\text{train}}\right) = \phi(z)^\top (\Phi^\top \Phi)^{-1} \Phi^\top Q,$$

where

$$Q = \text{diag}(1 - p_1^*, \ldots, 1 - p_n^*)$$

contains training-example-specific terms related to the model's confidence on the correct output. Applying $Q$ results in down-weighting the attribution of the samples that the model is very confident about them, and up-weighting the attribution of less confident samples.

An important distinction between TRAK and simpler gradient-similarity methods (like TracIN [24] and LESS [31]) lies in the reweighting term

$$(\Phi^\top \Phi)^{-1}.$$

A direct comparison such as

$$\phi(z_j^{\text{target}})^\top \phi(z_i^{\text{train}})$$

measures only the pairwise alignment between a training and target example. TRAK instead evaluates this relationship with respect to the geometry of the entire training set. The matrix $\Phi^\top \Phi$ captures how strongly different gradient directions are represented and correlated across the training examples, while its inverse reweights these directions during attribution. **Consequently, alignment along a direction that is repeated**

**across many training examples may receive less individual importance, whereas alignment along a less redundant direction can receive greater weight.** This pool-aware reweighting is a major conceptual distinction between TRAK and methods based primarily on pairwise gradient agreement such as TracIn. In addition, TRAK uses several checkpoints with multiple random projections and aggregates the scores to average out noise and reduce sensitivity to the stochasticity of neural-network training.

Although data attribution is commonly introduced as a mechanism for tracing model behavior back to individual training examples, the resulting scores can also be used for data selection. Given a training set

$$\mathcal{D}_{\text{train}} = \{z_i^{\text{train}}\}_{i=1}^{n}, \qquad \mathcal{D}_{\text{target}} = \{z_j^{\text{target}}\}_{j=1}^{m},$$

we denote the attribution of training example $z_i^{\text{train}}$ to target example $z_j^{\text{target}}$ as

$$\tau\left(z_i^{\text{train}} \to z_j^{\text{target}}\right) \equiv \tau\left(z_j^{\text{target}}, \mathcal{D}_{\text{train}}\right)_i .$$

and the pairwise attribution scores can then be shown via matrix

$$A \in \mathbb{R}^{n \times m}, \qquad A_{ij} = \tau\left(z_i^{\text{train}} \to z_j^{\text{target}}\right).$$

where $A_{ij}$ includes the attribution score of training sample $i$ on the target sample $j$. This perspective forms the basis for the data-selection methodology used in this thesis.

## 2.4 Supervised Fine-Tuning and LoRA

Supervised fine-tuning (SFT) adapts a pretrained language model to a desired task or response format using examples of the expected input–output behaviour [22]. For a causal language model, each example consists of an input prompt and a target completion, and training minimizes the negative log-likelihood of the target tokens conditioned on the prompt and preceding completion tokens. Text classification can be expressed in this format by using the text and task instructions as the prompt and the class label as the completion. In this thesis, the selected synthetic input–label pairs provide this supervision, allowing the student to specialize in a legal classification task beyond the limited real seed examples.

Full fine-tuning updates all model parameters, which requires substantial memory for gradients and optimizer states. Low-Rank Adaptation (LoRA) instead freezes the pre-trained weights and learns an additive update to selected weight matrices, represented as

the product of two smaller trainable matrices [16]. The rank of this update is chosen to be much smaller than the dimensions of the original matrix, substantially reducing the number of trainable parameters and the associated training-memory requirements. LoRA therefore provides a practical way to perform SFT repeatedly across the student sizes, selected datasets, and random seeds studied here. The student training procedure and experimental configuration are described in Chapters 3 and 4, respectively.

# Chapter 3

# Framework and Methodology

This chapter presents LegalSynth, our framework for distilling task-specific knowledge into compact language models under data scarcity. As illustrated in Figure 3.1, the framework consists of three main stages: synthesis, selection, and fine-tuning. At the first stage, a strong reasoning model analyzes the task and the limited seed examples from multiple perspectives, and the resulting analysis is used to guide large-scale synthetic data generation with a cheaper teacher model. Second, the resulting high-recall candidate pool is scored using TRAK attribution with respect to a held-out validation set, enabling us to use the scores to find the best candidates for training. Finally, the selected synthetic data are used to fine-tune small language models (Qwen3.5-0.8B, 2B, and 4B) with LoRA, with the goal of matching or outperforming substantially larger teacher models.

## 3.1 Problem Setup and Data Organization

This section fixes the notation and data organization used throughout the rest of the chapter. It first states what a LegalBench classification task consists of and how it is represented in the LegalSynth implementation, then defines the three-way split of the available real data and the distinct role each split plays across synthesis, selection, and fine-tuning.

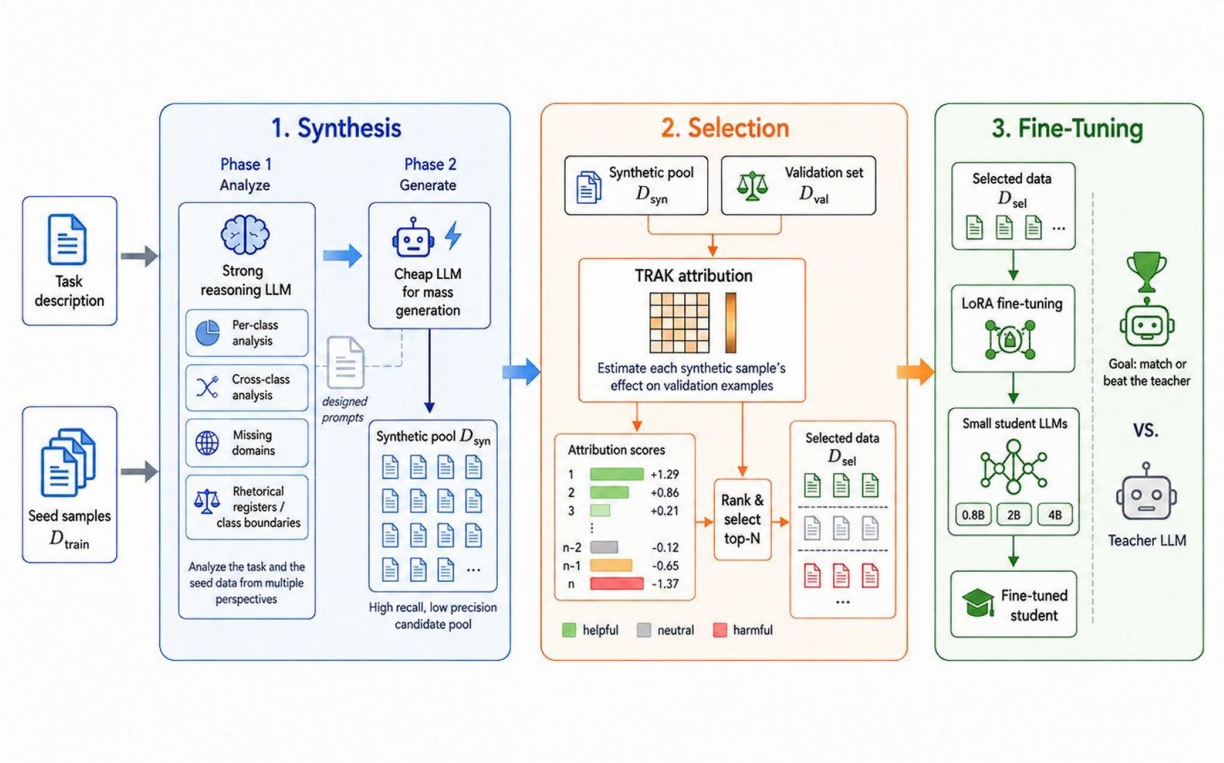



Figure 3.1: End-to-end overview of the LegalSynth framework. Task-aware synthesis produces a broad candidate pool, TRAK attribution ranks candidates by their estimated effect on a held-out validation set, and the selected subset is used to fine-tune compact student language models.

### 3.1.1 LegalBench Task Structure

A LegalBench classification task consists of a natural-language task description $d$, a finite label set $\mathcal{Y} = \{y_1, \ldots, y_K\}$ together with a natural-language definition for each label, and a small labelled dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, where each input $x_i \in \mathcal{X}$ is a short passage of legal or quasi-legal text and $y_i \in \mathcal{Y}$ is its ground-truth label [13]. The task description explains the classification problem in prose – for example, the legal rule used to decide trademark distinctiveness, or the purpose behind a lawyer's question during oral argument – and each label's definition states the criteria that separate it from every other class.

In the LegalSynth implementation, $d$ and $\mathcal{Y}$ are represented by a single structured task manifest per task, holding the task name, domain, natural-language description, and an ordered mapping from each label name to its definition; $\mathcal{D}$ is stored as a tabular dataset of labelled examples. Teacher prompting and multi-step synthesis use the task information in this manifest, while attribution and fine-tuning operate on a simple input–label format. The same pipeline supports tasks with different label counts, label semantics, and text domains, making the four LegalBench tasks studied in this thesis – *Abercrombie* (trademark-distinctiveness classification), *Insurance Policy Interpretation*, *Oral Argument Question Purpose*, and *Function of Decision Section* – instances of one framework rather than four separate ones. Task-specific descriptions, label sets, and dataset sizes are given alongside the corresponding results in Chapter 4.

### 3.1.2 Train, Validation, and Test Splits

Each task's real dataset $\mathcal{D}$ is partitioned once, before any synthesis or training, into three disjoint splits,

$$\mathcal{D} = \mathcal{D}_{\text{train}} \sqcup \mathcal{D}_{\text{val}} \sqcup \mathcal{D}_{\text{test}}, \qquad |\mathcal{D}_{\text{train}}| : |\mathcal{D}_{\text{val}}| : |\mathcal{D}_{\text{test}}| \approx 40 : 30 : 30, \tag{3.1}$$

using the same ratio for every task in this study. The three splits are not interchangeable general-purpose partitions: each has defined uses in synthesis, model selection, and evaluation, as described below.

- $\mathcal{D}_{\text{train}}$ is the seed data for synthetic generation (Section 3.2): it is the material the teacher model analyzes to produce task-aware prompts and the source of the few-shot examples embedded in the synthesis prompts. For the teacher few-shot evaluation reported in Chapter 4, five demonstrations per class are sampled from the combined $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{val}}$ pool.

- $\mathcal{D}_{\text{val}}$ provides held-out real examples for early stopping during LoRA fine-tuning and serves as the target set for TRAK attribution (Section 3.3).
- $\mathcal{D}_{\text{test}}$ is used for the final evaluation of students and teacher baselines. In an additional experiment, it also serves as the attribution target set to estimate the maximum achievable performance through data selection, which is only treated as an upper-bound for selection, and not treated as the main headline results (Section 3.3).

## 3.2 Synthetic Data Generation

Section 2.2.2 introduced the general synthetic-example relation of Equation (2.6), in which a teacher samples a new input-label pair conditioned on the task description $I$ and a small seed set $\mathcal{D}_{\text{seed}}$, and noted that the resulting pool is a high-recall, low-precision object: broad but unevenly useful. This section describes how $\mathcal{D}_{\text{train}}$ is turned into the synthetic pool $\mathcal{D}_{\text{syn}}$ of Equation (2.7) through two complementary generation channels, and is explicit throughout about which design choices target the *recall* side of that trade-off – covering more of the boundaries, domains, and registers a task can contain – since precision is deliberately left to the attribution stage of Section 3.3.

### 3.2.1 Multi-Step Task-Aware Synthesis

Prompting a teacher with only the task description and a handful of seed examples, as in Equation (2.6) taken literally, tends to reproduce whatever is already easy about a task: the teacher knows the label names and roughly what each one means, but not which pairs of labels it is likely to confuse, which subject-matter areas the seed examples fail to cover, or which of several possible phrasings and registers a label can appear in. Multi-step synthesis addresses this by having the teacher analyze $\mathcal{D}_{\text{train}}$ before generating anything, so that the resulting prompts target the specific gaps and boundaries a generic prompt would not know to ask for. This does not make generation high-precision – individual synthetic examples can still be ambiguous, redundant, or mislabelled – it only improves the composition of the candidate pool that the attribution stage in Section 3.3 then has to work with. The pipeline has four stages:

1. **Per-class analysis.** A strong reasoning model (Gemini 3.1 pro), acting as an orchestrator, examines the seed examples of one class at a time – alongside a contrast set drawn from every other class – and characterizes what a text must and must not

contain to belong to that class, the surface patterns that tend to signal it, and the types of examples the seed data does not yet contain.

2. **Cross-class analysis.** The same model compares every class's findings from the previous stage to identify which pairs of labels are hardest to tell apart and why, which subject-matter domains are absent across the task as a whole rather than any single class, and which rhetorical registers – ways of expressing a label, independent of the label itself – are underrepresented in the seed data.[1]

3. **Design and prompt writing.** Given the previous stage's findings, a fixed budget of generation *slots* is assigned by rule rather than by the model:

   - **Contrastive:** Extracts the hardest-to-distinguish label pairs found during cross-class analysis that are most likely to be confused by the model, and generates examples to help the model understand the boundaries between the two classes. Therefore, the contrastive synthetic samples are written specifically to demonstrate the discriminator that separates the two classes.
   - **Domain sweep:** Extracts the domain subjects that are found missing from the seed data during cross-class analysis, and cycles through every class within each missing area, so the model learns to recognize each class across topics $\mathcal{D}_{\text{train}}$ never covered, not just the few topics the seed examples happened to include.
   - **Rhetorical register sweep:** Extracts the underrepresented rhetorical registers found in the seed training set during cross-class analysis, and cycles through every class within each underrepresented register, so a class is demonstrated in forms of expression beyond whichever register happens to dominate $\mathcal{D}_{\text{train}}$.
   - **Broad detailed generation:** A standard few-shot prompt for generation, consisting of all analysis findings from the first three stages, to better guide the generation process.

   Only once these slot assignments and per-slot sample counts are fixed does the orchestrator take a further pass, whose only task is to write the natural-language prompt text for each slot – it cannot change which classes, domains, or registers a

[1] A rhetorical register denotes *how* a legal statement is expressed – for example, formal statutory or contractual phrasing, a plain-language paraphrase, a narrative case-summary description, or a terse question-and-answer exchange of the kind found in oral-argument transcripts – as distinct from *why* the statement belongs to a given class. The registers relevant to a task are not fixed in advance; they are derived per task from the patterns observed in that task's own seed data during the per-class and cross-class analysis stages above.

slot targets, or how many examples it should produce. Keeping this split explicit is a deliberate safeguard: it stops the label distribution and topical coverage of $\mathcal{D}_{\text{syn}}$ from drifting silently between runs, and keeps every synthetic example traceable to a specific, inspectable slot. Before any generation is paid for, every designed prompt is read by a human, who edits and refines the prompt if a part of it needs correction. This human-in-the-loop check operates on prompt text, not on individual generated rows.

4. **Mass generation.** With slots and prompts fixed, a fast, low-reasoning-effort model (Gemini-3.7-Flash) generates the examples for every slot in parallel, dispatched in batches, cycling through few-shot shards drawn from $\mathcal{D}_{\text{train}}$ so that the whole seed set informs generation without over-representing any single example. A cheaper model is deliberately used here rather than for analysis: recognizing a subtle class boundary is a qualitatively harder task than writing one further example of a boundary that analysis has already identified.

The static templates used by the multi-step synthesis pipeline are provided in Appendix A.

### 3.2.2 Synthetic Augmentation

Independently of multi-step synthesis, every example in $\mathcal{D}_{\text{train}}$ is directly paraphrased by the teacher at three difficulty tiers – easy, medium, and hard – with the teacher instructed to preserve the source label and its underlying legal or communicative function while varying wording, sentence structure, and rhetorical presentation, inspired by [34]. Where multi-step synthesis targets *breadth* – boundaries, domains, and registers not yet observed in $\mathcal{D}_{\text{train}}$ – paraphrasing targets *density*: many close variants of examples already known to carry a correct label, staying near the observed task distribution rather than extending it.

The multi-step generations and the paraphrased augmentations are combined into a single pool, $\mathcal{D}_{\text{syn}}$ of Equation (2.7). Consistent with the high-recall, low-precision characterization of Section 2.2.2, this pool is intentionally not curated at generation time: it may contain highly useful examples alongside ones that are ambiguous, redundant, off-distribution, or incorrectly labelled. Determining which of these candidates provide useful supervision for a given student is the task of the attribution stage described next, in Section 3.3.

## 3.3 Data Selection via Attribution

Section 3.2 deliberately left $\mathcal{D}_{\text{syn}}$ uncurated: it is a high-recall, low-precision pool (Section 2.2.2) that may help or hurt a given student depending on the example. This section describes how we select the most useful vs. hurtful samples for the final training set subset using TRAK, a model-aware data attribution method that estimates each candidate's effect on a fixed set of target examples [23]. TRAK's derivation and general estimator are given in Section 2.3 and are not repeated here; this section covers only how attribution is set up and used in this thesis: which attribution target set and reference model each score is computed against, how checkpoint and projection randomness are averaged out, the resulting selection policies, and the adaptation required to run TRAK on Qwen 3.5 students.

### 3.3.1 Attribution Setup

Every candidate $x \in \mathcal{D}_{\text{syn}}$ is scored against a fixed target set $\mathcal{Q}$, using either $\mathcal{D}_{\text{val}}$ or $\mathcal{D}_{\text{test}}$ (Section 3.1.2). Validation-set attribution supplies the rankings used for the main student results. Test-set attribution serves as an oracle comparison intended to estimate an expected upper bound on selection performance within the evaluated setup, although it does not need to outperform validation-set attribution in every experiment. For every task, TRAK is computed only with a `Qwen3.5-0.8B-Base` reference model, LoRA fine-tuned on the full synthetic candidate pool $\mathcal{D}_{\text{syn}}$. Its scores and rankings are reused to select training subsets for all three student sizes: 0.8B, 2B, and 4B. Every candidate and target example is passed through the reference checkpoints to obtain the gradient-based features used by TRAK. For a candidate–target pair $(x, q)$, this produces an estimate of the candidate's effect on the reference model's behaviour at $q$. Averaging these scores across the chosen target set gives the per-candidate score used for ranking.

### 3.3.2 Checkpoint and Projection Ensembling

TRAK scores depend on the reference model's training trajectory and the random projection used to compress per-example gradients. LegalSynth reduces sensitivity to these choices by averaging scores across checkpoints and projections. Reference fine-tuning runs use different random seeds, and multiple epoch checkpoints are retained from each run. Each checkpoint is scored with several independent random projections, and the resulting estimates are averaged. Checkpoints from the same run share a training trajectory and

are therefore not independent trained models. The numbers of runs, checkpoints, and projections are specified in Chapter 4.

### 3.3.3 Selection Policies

Given the aggregated score, a selection policy picks a subset $\mathcal{D}_{\text{sel}} \subset \mathcal{D}_{\text{syn}}$ of a chosen size $N$. Every comparison in this thesis is budget-matched: for the same $N$, several policies are compared using an otherwise identical training procedure. Five policies are used with validation- and test-set attribution, together with a random baseline:

- **validation_top / validation_bottom**: the $N$ highest- or lowest-scoring candidates by validation-set attribution; for a fixed budget $N$, these rankings require no test labels;
- **test_top / test_bottom**: the same, but ranked by test-set attribution; these are oracle variants, with **test_top** providing the expected upper-bound comparison and **test_bottom** providing its low-ranked counterpart;
- **random**: a uniform random sample of size $N$, independent of any attribution score.

Each comparison isolates a different question. Top versus random tests whether attribution-based selection is better than no selection at all; top versus bottom tests whether the score is directional – whether higher scores identify more useful examples and lower scores identify less useful ones. Validation versus test attribution examines how strongly the ranking depends on the attribution target set; closer results indicate that the validation set yields a ranking similar to the test-set oracle.

### 3.3.4 Adapting TRAK to Qwen 3.5

The standard TRAK implementation could not be directly applied to the Qwen 3.5 models used in this work because its per-example gradient computation was incompatible with the model's attention implementation. We therefore compute the required per-example gradients directly for the trainable LoRA parameters during ordinary forward and backward passes, using the model output at the first answer-token position of each example. The resulting gradients are then projected and used in the standard TRAK estimator described in Section 2.3.2.

### 3.3.5 Data Selection Procedure

Algorithm 1 summarizes the TRAK-based selection procedure for one task. Attribution is computed using the designated reference model, and the resulting ranking is reused across student sizes. For each checkpoint, we compute TRAK scores under multiple independent random projections and average the resulting estimators to reduce the noise. The selected $\mathcal{D}_{\text{sel}}$ is then used as the training set for the student fine-tuning procedure described in Section 3.4.

## 3.4 Student Model Fine-Tuning

Once a selection policy fixes a budget-matched subset $\mathcal{D}_{\text{sel}} \subseteq \mathcal{D}_{\text{syn}}$ (Section 3.3), a student is trained by supervised fine-tuning on $\mathcal{D}_{\text{sel}}$. Each student is a compact, open-weight causal language model from the Qwen family, adapted with LoRA rather than full fine-tuning; the choice of LoRA is not merely the standard parameter-efficiency argument reviewed in Chapter 2, but a practical necessity here, since the experimental design in Chapter 4 repeats fine-tuning across every task, every selection policy, several subset sizes, several student sizes, and multiple random seeds, and only a cheap per-run adaptation makes that sweep tractable.

### 3.4.1 Question-Answer Format and Training Objective

Every training pair $(x, y) \in \mathcal{D}_{\text{sel}}$ is turned into a single prompt-completion sequence using the fixed format

```
Question: x
Answer: y
```

Here, $x$ is the question text and $y$ is its label. The students are base language models rather than instruction-tuned models, so their fine-tuning inputs contain no task-specific instruction, task description, or label definitions. This student format is distinct from the task-aware prompts used for teacher classification and synthetic generation in Section 3.2. The label $y$ is followed by an end-of-sequence token. Training minimizes the standard autoregressive cross-entropy loss, but restricted to the label and end-of-sequence tokens:

$$\mathcal{L}_{\text{SFT}} = \sum_{(x,y) \in \mathcal{D}_{\text{sel}}} -\log p_S(y \mid x), \tag{3.2}$$

**Algorithm 1** TRAK-based synthetic data selection

**Require:** synthetic pool $\mathcal{D}_{\text{syn}}$, attribution target set $\mathcal{D}_{\text{attr}} \in \{\mathcal{D}_{\text{val}}, \mathcal{D}_{\text{test}}\}$, reference checkpoints $\{\theta_k\}_{k=1}^{K}$, number of random projections $J$, selection policy, budget $N$

**function** FEATURIZE($\theta$, $P$, $\mathcal{D}$)
  **for** each $z = (x, y) \in \mathcal{D}$ **do**
    compute the first-completion-token classification margin $f(z; \theta)$
    reconstruct the LoRA gradient $g(z) \leftarrow \nabla_{\theta_{\text{LoRA}}} f(z; \theta)$
    project the gradient: $\phi(z) \leftarrow P^{\top} g(z)$

  **return** $\{\phi(z)\}_{z \in \mathcal{D}}$

**function** TRAKCOMPONENTS($\theta$, $P$)
  $\Phi_{\text{syn}} \leftarrow$ FEATURIZE$(\theta, P, \mathcal{D}_{\text{syn}})$
  $\Phi_{\text{attr}} \leftarrow$ FEATURIZE$(\theta, P, \mathcal{D}_{\text{attr}})$
  **for** each $z \in \mathcal{D}_{\text{syn}}$ **do**
    $w(z) \leftarrow 1 - p_y(z; \theta)$

  $Q \leftarrow \text{diag}\big(\{w(z)\}_{z \in \mathcal{D}_{\text{syn}}}\big)$
  $A \leftarrow \Phi_{\text{syn}}(\Phi_{\text{syn}}^{\top}\Phi_{\text{syn}})^{-1}\Phi_{\text{attr}}^{\top}$
  **return** $(A, Q)$

$(\overline{A}, \overline{Q}) \leftarrow (0, 0)$
**for** $k = 1, \ldots, K$ **do**
  **for** $j = 1, \ldots, J$ **do**
    sample an independent random projection matrix $P_{k,j}$
    $(A_{k,j}, Q_{k,j}) \leftarrow$ TRAKCOMPONENTS$(\theta_k, P_{k,j})$
    $(\overline{A}, \overline{Q}) \leftarrow (\overline{A} + A_{k,j}, \overline{Q} + Q_{k,j})$

$T \leftarrow (\overline{Q}/(KJ))(\overline{A}/(KJ))$ ▷ average over checkpoints and projections
**for** each $z \in \mathcal{D}_{\text{syn}}$ **do**
  $s(z) \leftarrow \frac{1}{|\mathcal{D}_{\text{attr}}|} \sum_{t \in \mathcal{D}_{\text{attr}}} T[z, t]$ ▷ average attribution scores over all target samples

**if** policy is **random then**
  $\mathcal{D}_{\text{sel}} \leftarrow$ uniformly sample $N$ examples from $\mathcal{D}_{\text{syn}}$
**else if** policy is **top then**
  $\mathcal{D}_{\text{sel}} \leftarrow$ the $N$ examples with largest $s(z)$
**else if** policy is **bottom then**
  $\mathcal{D}_{\text{sel}} \leftarrow$ the $N$ examples with smallest $s(z)$

**return** $\mathcal{D}_{\text{sel}}$

where tokens belonging to the fixed prompt and $x$ are masked out of the loss, and only the label and end-of-sequence tokens contribute. Equation (3.2) has the same form as the hard-label distillation objective in Equation (2.5) (Section 2.2): fine-tuning here is exactly hard-label distillation, except that the label $y$ is not the teacher's prediction on a fixed real input but the label $y^{\text{syn}}$ the teacher itself attached when synthesizing $x$ (Equation (2.6)). SFT trains on the complete label completion, whereas TRAK uses the classification margin at its first token (Section 3.3). The initial answer tokens distinguish the classes within each task, so this position supplies a class-specific attribution signal without requiring the attribution output to equal the training loss.

### 3.4.2 Adaptation and Early Stopping

Three Qwen 3.5 student sizes are evaluated – 0.8B, 2B, and 4B parameters – using a shared LoRA rank and scaling. Training uses two complementary early-stopping criteria on the validation set: one monitors macro-F1, while the other monitors cross-entropy loss. Each maintains its own patience counter, and either can stop training when its monitored quantity fails to improve for the specified number of evaluations. Macro-F1 measures classification performance directly, while loss tracks the probability assigned to the correct label completion. For each run, the reported metrics are taken from the evaluated epoch with the highest validation macro-F1; this reporting choice does not require the trainer to restore a checkpoint. Additionally, the validation set guide attribution ranking as well, where the final reported subset size $N$ for the primary results is chosen using mean validation macro-F1, according to the reporting protocol described in Section 4.2.1.

# Chapter 4

# Evaluation

## 4.1 Experimental Setup

### 4.1.1 Tasks

Four LegalBench tasks are used to evaluate the hypotheses of this thesis. LegalBench consists of 162 tasks, spanning information extraction, binary classification, and multi-way classification problems across a wide range of dataset sizes [13]. To represent the data-scarce scenario this thesis targets, tasks with more than 400 real examples were first filtered out, and 4 challenging multi-way classification tasks were then chosen from the remainder, each posing a fully different legal problem, so that the results below are not specific to one kind of legal problem.

Each task is described below using the verbatim explanation of the background, category definitions, and task framing given in the original LegalBench paper [13].

#### Abercrombie

**Background.** A particular mark (e.g., a name for a product or service) is only eligible for trademark protection if it is considered to be distinctive. In assessing whether a mark is distinctive, lawyers and judges follow the framework set out in the case *Abercrombie & Fitch Co. v. Hunting World, Inc.*, which enumerates five categories of distinctiveness. These categories characterize the relationship between the dictionary definition of the term used in the mark, and the service or product it is being attached to.

The five categories are:

- **Generic:** A name is generic with respect to a product or service if it connotes the basic nature of the product/service, rather than more individualized characteristics of the product. For example, the mark *"Salt"* for packaged sodium chloride would be generic under *Abercrombie* because "salt" is the common name for sodium chloride. It is also common to think of generic marks as merely referring to the class of goods for which a particular product is a species.

- **Descriptive:** A name is descriptive if it identifies a characteristic or quality of an article or service, such as color, odor, function, dimensions, or ingredients. For example, the name *"Sharp"* for a television would be descriptive, because it describes a plausible characteristic of televisions (i.e., their sharp image quality).

- **Suggestive:** A name is suggestive if it suggests, rather than describes, some particular characteristic of the goods or services to which it applies. An essential aspect of suggestive names is that it requires the consumer to exercise the imagination in order to draw a conclusion as to the nature of the goods and services. For example, the name *"Greyhound"* would be suggestive for a bus service, because greyhounds are considered to be fast, and "fast" is an adjective that could be used to describe a bus service.

- **Arbitrary:** A name is arbitrary if it is a "real" word but seemingly "arbitrary" with respect to the product or service. For example, the mark *"Apple"* for a software company is arbitrary, because apples are unrelated to software.

- **Fanciful:** A name is fanciful if it is entirely made up, and not found in the English dictionary. For example, *"Lanmbe"* is a fanciful mark, because it is a made-up word.

The *Abercrombie* spectrum is commonly taught as part of Intellectual Property courses in law school, and students are expected to understand how to determine the *Abercrombie* classification for a particular product/mark combination.

Performing the *Abercrombie* task requires reasoning about the literal meaning of a word and the degree of its connection to a particular product/service. It requires having some understanding of the types of words that could plausibly be used to describe a particular good/service, and the extent to which those words relate to a particular mark. It also requires reasoning as to whether a particular word is a real English word.

**Task.** The Abercrombie task requires an LLM to determine – given a candidate mark and a description of a product/service – which of the five *Abercrombie* categories above apply.

**Significance.** Given how easy this task is for lawyers with a basic training in intellectual property law, it is unlikely that LLMs will be called on to perform this task in the actual practice of law, or that the ability for LLMs to perform this task would alter the way in which lawyers approach IP practice. Instead, the Abercrombie task is significant as a measurement of reasoning ability. Because it is "simplistic" by the standards of human lawyers, it provides a useful objective measure of reasoning progress for LLMs.

Table 4.1: Abercrombie: label distribution across the full real dataset ($n = 100$).

| Label | Count |
|---|---|
| Arbitrary | 20 |
| Descriptive | 20 |
| Fanciful | 20 |
| Generic | 20 |
| Suggestive | 20 |
| Total | 100 |

### Insurance Policy Interpretation

**Background.** Insurance disputes often arise when parties disagree on whether a claim is covered under a certain insurance policy. To study such disagreements in interpretation, researchers at Stanford recruited crowdsource workers to review a pair of an insurance policy and a claim and respond whether they believe the claim is covered. A policy-claim pair whose applicability the workers disagree with each other on suggests ambiguity in the policy text.

**Task.** The Insurance Interpretation task requires an LLM to review a pair of an insurance policy and a claim and determine whether the policy:

- Clearly covers the claim
- Clearly does not cover it
- Is unclear whether it covers it or not

LegalSynth's task manifest (Section 3.1) encodes these three outcomes as labels A, B, and C, respectively.

**Significance and Value.** The ability to determine whether an insurance claim is covered under a given policy can significantly reduce claim processing time. It can also shine light on potential ambiguity in existing policies.

Additionally, this task represents one of the rare benchmarks where an LLM is required to predict laypeople's legal interpretations, as ground truth labels are retrieved based on crowdsourced responses.

Table 4.2: Insurance Policy Interpretation: label distribution across the full real dataset ($n = 138$).

| Label | Count |
|---|---|
| C (ambiguous) | 60 |
| A (covered) | 48 |
| B (not covered) | 30 |
| Total | 138 |

### Oral Argument Question Purpose

**Background.** Before a court decides a case, it typically calls before it the attorneys for the parties to the lawsuit to orally present their arguments for why the case should be resolved in favor of their clients and to answer any questions that the judge or judges have of them. In modern times, however, oral argument is not lawyers' primary avenue for explaining their positions. Instead, parties submit their arguments in written form ("briefs") and use their oral argument time primarily to supplement those submissions by reiterating their key positions, clarifying areas of ambiguity, and seeking to persuade judges who are uncertain how the case should be resolved.

Although there is no universally accepted listing, judges' questions at oral argument tend to fall into a few categories:

- **Background:** A question seeking factual or procedural information that is missing or not clear in the briefing.
- **Clarification:** A question seeking to get an advocate to clarify her position or the scope of the rule being advocated.

- **Implications:** A question about the limits of a rule or its implications for future cases.
- **Support:** A question offering support for the advocate's position.
- **Criticism:** A question criticizing an advocate's position.
- **Communicate:** A question designed primarily to communicate with one or more other judges on the court.
- **Humor:** A question designed to interject humor into the argument and relieve tension.

A lawyer presenting her case before a court at oral argument must be able to quickly and accurately determine why the judge has asked a particular question so as to answer it on behalf of her client in the most persuasive way possible. It is also a difficult task. Under the pressure of persistent and difficult questioning, it is easy for a lawyer to misread a question and offer an unresponsive or misguided answer. Skillful lawyers learn to quickly understand not only judges' questions but the reasons they are asked.

**Task.** The Purpose of Oral Argument Questions task requires an LLM to determine – given a question from an oral argument transcript – for which of the seven purposes above the judge asked the question.

**Significance and Value.** Young attorneys, and even many experienced ones, struggle with oral advocacy. It requires comfort in the courtroom, quick thinking, and careful demeanor to assess from the tone and content of a question why the judge is asking it and how most effectively to respond.

In one regard, LLMs will be superior: they do not suffer from human nervousness. However, given only a text prompt rather than an audible question from which to infer tone, and given only a judge's question and not the full case context, this task would be a challenge even for a seasoned lawyer. Whether LLMs can succeed will be an extremely interesting measure of progress in legal analysis.

### Function of Decision Section

In LegalBench, the *Function of Decision Section* task is denoted as `function_of_decision_section`.

Table 4.3: Oral Argument Question Purpose: label distribution across the full real dataset ($n = 319$).

| Label | Count |
|---|---|
| Clarification | 84 |
| Implications | 68 |
| Background | 58 |
| Criticism | 52 |
| Humor | 29 |
| Communicate | 15 |
| Support | 13 |
| Total | 319 |

**Background.** In common-law legal systems, written judicial decisions serve two functions: (1) they resolve the dispute that litigants brought before the court and explain the reason for the court's decision, and (2) they become new law, binding on future parties and future courts should another case arise that presents sufficiently similar facts.

Because judicial decisions not only describe the law, but are themselves the law, lawyers in common-law legal systems must be able to read and digest case law to extract key legal principles and apply those principles to their own cases. This skill takes time and practice to develop.

Importantly, not every word in a judicial decision is binding – only the facts and reasoning that were required for the court to reach its decision. Thus, lawyers must distinguish important from trivial facts across numerous past decisions before they can conclude what the law on a particular issue is.

One of the most foundational case-reading skills is the ability to review a legal decision and identify the function that each section of the decision serves. In the American legal education system, this skill is taught beginning in the first year of law school, often by encouraging students to identify the function of each section of a decision. A typical classification scheme is as follows:

- **Facts:** A section of the decision that recounts the historical events and interactions between the parties that gave rise to the dispute.
- **Procedural History:** A section of the decision that describes the parties' prior legal filings and prior court decisions that led up to the issue to be resolved by the

decision.

- **Issue:** A section of the decision that describes a legal or factual issue to be considered by the court.
- **Rule:** A section of the decision that states a legal rule relevant to resolution of the case.
- **Analysis:** A section of the decision that evaluates an issue before the court by applying governing legal principles to the facts of the case.
- **Conclusion:** A section of the decision that articulates the court's conclusion regarding a question presented to it.
- **Decree:** A section of the decision that announces and effectuates the court's resolution of the parties' dispute, for example, granting or denying a party's motion or affirming, vacating, reversing, or remanding a lower court's decision.

Identifying the function of sections within judicial decisions is a fundamental skill for lawyers in common-law legal systems. Without it, precedent-based legal reasoning would be impossible.

**Task.** The Function of Decision Sections task requires an LLM to determine – given a one-paragraph excerpt of a legal decision – which of the seven functions above that paragraph serves in the context of the entire decision.

**Significance and Value.** Beginning law students may initially have trouble identifying the function of a particular section within a judicial opinion, but it quickly becomes a simple task. LLMs would not be called on to perform this task in the actual practice of law, but because it is a foundational legal reasoning skill, it provides a useful measure of reasoning progress for LLMs.

The published LegalBench distribution above contains 374 examples. The preprocessing used for the experiments retains 373 examples, partitioned into 150 training examples, 112 validation examples, and 111 test examples.

### 4.1.2 Settings

**Synthesis.** All four tasks use the multi-step synthesis pipeline of Section 3.2. The analysis phase is performed by a strong reasoning model (`gemini-3.1-pro-preview`, reasoning

Table 4.4: Function of Decision Section: label distribution across the full real dataset ($n = 374$).

| Label | Count |
|---|---|
| Procedural History | 59 |
| Analysis | 57 |
| Rule | 57 |
| Issue | 52 |
| Conclusion | 51 |
| Facts | 50 |
| Decree | 48 |
| Total | 374 |

effort *high*, temperature 0.3), and the consecutive mass generation is performed by a faster, non-reasoning model (`gemini-3.7-flash`, temperature 1.0). We use low temperature on the analysis part using a strong model for more determinism and reproducibility, and conversely, we use higher temperature for mass generation to have more diversity and recall. Table 4.5 gives the number of synthetic examples produced per method for each task.

Table 4.5: Number of synthetic examples generated per method for each task. Contrastive counts are summed over the three hardest label pairs identified during cross-class analysis.

| | Paraphrase | | | Multi-step | | | |
|---|---|---|---|---|---|---|---|
| Task | Easy | Medium | Hard | Contrastive | Domain | Register | Analysis |
| Abercrombie | 700 | 700 | 700 | 2,250 | 2,000 | 0[1] | 3,691 |
| Insurance Policy Interpretation | 2,000 | 2,000 | 2,000 | 3,660 | 6,240 | 3,690 | 6,795 |
| Oral Argument Question Purpose | 2,000 | 2,000 | 2,000 | 3,850 | 5,600 | 3,500 | 6,975 |
| Function of Decision Section | 2,000 | 2,000 | 2,000 | 3,850 | 5,600 | 3,500 | 6,975 |

| Task | Total pool size |
|---|---|
| Abercrombie | 10,041 |
| Insurance Policy Interpretation | 26,385 |
| Oral Argument Question Purpose | 25,925 |
| Function of Decision Section | 25,925 |

[1]The rhetorical register sweep is omitted for Abercrombie, whose inputs are a fixed mark–product pair rather than free-form legal prose, and so admit no meaningful variation in register.

**Data selection.** TRAK attribution (Section 3.3) uses `Qwen3.5-0.8B-Base` as the reference model for every task, regardless of which student size is ultimately being trained on the selected subset. Two independently seeded reference fine-tuning runs are each checkpointed at 3 epochs, and every checkpoint is scored with 8 independent random projections (dimension 1024), giving $2 \times 3 \times 8 = 48$ TRAK estimators averaged into each candidate's final score. Scores are computed separately against $\mathcal{D}_{\text{val}}$ and $\mathcal{D}_{\text{test}}$, producing the `validation_top`/`validation_bottom` and `test_top`/`test_bottom` selection policies of Section 3.3.

**Data-scaling sweep.** After computing attribution scores, we rank the synthetic candidates from highest to lowest score and fine-tune students on the top-$N$ examples at multiple subset sizes. This sweep examines how performance changes as more synthetic examples are included in training. For each student size, we select the best evaluated $N$ for reporting according to the validation- or test-based selection protocols described in Section 4.2.1.

For Abercrombie, we evaluate training-subset sizes from 1,000 to 9,000 examples in increments of 1,000. For Insurance Policy Interpretation, Oral Argument Question Purpose, and Function of Decision Section, we evaluate 500 and 1,000 examples, followed by sizes from 2,500 to 20,000 in increments of 2,500, and finally 25,000. Each value of $N$ denotes the number of synthetic examples retained for fine-tuning. Within each task, the same subset sizes are also used for bottom-$N$ and random selection, allowing comparisons across all five selection policies and all three student sizes.

**Fine-tuning.** Three Qwen3.5 student sizes are fine-tuned per task – `Qwen3.5-0.8B-Base`, `Qwen3.5-2B-Base`, and `Qwen3.5-4B-Base` – with identical LoRA parameters of rank $r = 16$, scaling $\alpha = 32$, and dropout 0.05. Every run trains for up to 5 epochs with AdamW, a cosine learning-rate schedule, learning rate $2 \times 10^{-4}$, and an effective batch size of 64 held constant across sizes by trading off per-device batch size against gradient-accumulation steps as the model grows. Two complementary early-stopping criteria monitor validation-set macro-F1 (patience of 3 epoch evaluations) and validation-set cross-entropy loss (patience of 5 checks, checked every tenth of an epoch). Either criterion can stop training independently. For each run, reported metrics are taken from the epoch with the highest validation macro-F1, as described in Section 3.4. Every fine-tuned configuration is repeated over 3 random seeds; tables below report the mean and standard error across those seeds.

**Teachers.** As the baseline models to be beaten by the student, we use `GPT-5.6-Terra`, `Gemini-3.7-Flash`, and `Qwen3.5-397B-A17B`, each evaluated under three prompting regimes of *zero-shot*, *few-shot*, and *many-shot*. For *zero-shot*, we only use the task description in the prompt and ask the model to label the given test sample. For few-shot, we include five examples per class sampled from the combined train and validation splits, in addition to the task description. For many-shot, we include all train and validation examples in the prompt.

## 4.2 Results

### 4.2.1 Main Results

For each task, Table 4.6–4.9 reports the three teachers under zero-shot, few-shot, and many-shot prompting, followed by each of the three student sizes under five configurations:

- *Zero-shot*: Prompting the model with only the task description, without any fine-tuning. We used the Instruct version of Qwen3.5 for it.
- *SFT on all synthetic data*: LoRA fine-tuning on the entire synthetic data pool $\mathcal{D}_{\text{syn}}$ with no selection.
- *Validation-set attribution, $N$ chosen on validation*: We rank synthetic candidates by their mean TRAK attribution against $\mathcal{D}_{\text{val}}$. For each student size, we fine-tune students on the top $N$ candidates at several subset sizes, then choose the $N$ with the highest mean validation macro-F1 across the three training seeds. We report the corresponding test-set performance as the primary result.
- *Validation-set attribution, $N$ chosen on test*: We use the same validation-based ranking and fine-tuning runs, but choose the subset size with the highest mean test macro-F1 across the three training seeds. This comparison shows how the reported performance changes when test results are used to choose $N$.
- *Test-set attribution, $N$ chosen on test (oracle)*: We rank synthetic candidates by their mean TRAK attribution against $\mathcal{D}_{\text{test}}$ and fine-tune students on the top $N$ candidates at the same evaluated subset sizes. We then choose the $N$ with the highest mean test macro-F1 across the three training seeds. This oracle experiment estimates the performance attainable when test information guides both candidate ranking and subset-size selection; it does not establish a guaranteed upper bound.

All performance columns in the tables report test-set metrics: macro-F1 as the headline metric, alongside macro recall and macro precision. Macro recall equals balanced accuracy, but generally differs from ordinary accuracy on imbalanced datasets. Ordinary accuracy is omitted from the tables for concision and retained in the policy plots. The attribution target set determines the candidate ranking, whereas the $N$ *chosen on* column identifies the split used to select the budget. For each seed and budget, validation and test metrics are taken from the epoch with the highest validation macro-F1. The budget is then selected by averaging the appropriate selection metric across the three seeds, and the table reports the mean and standard error of test metrics at that same budget. For every task, four figures accompany the table: a macro-F1 data-scaling plot for all three student sizes (Qwen3.5-0.8B/2B/4B) under TRAK top-$N$ validation-set selection, together with the GPT-5.6-Terra zero-shot, few-shot, and many-shot baselines for comparison; and, for each of the three student sizes, the full data-scaling curve across all five selection policies (`random`, `validation_top`/`validation_bottom`, `test_top`/`test_bottom`) and all four metrics.

### Abercrombie

Table 4.6: Teacher and student results on the *Abercrombie* task. In every TRAK row, **bold** macro-F1 means exceed at least one teacher's zero-shot macro-F1; † and ‡ mark means exceeding at least one teacher's few-shot and many-shot macro-F1, respectively. Each comparison uses the lowest teacher macro-F1 within that prompting regime and is assessed independently.

| Model | Method | $N$ chosen on | $N$ | Macro F1 (%) | Macro Recall (%) | Macro Precision (%) |
|---|---|---|---|---|---|---|
| GPT-5.6-Terra | Zero-shot | – | – | 83.5 | 83.3 | 86.1 |
| GPT-5.6-Terra | 25-shot (few) | – | 25 | 86.9 | 86.7 | 87.6 |
| GPT-5.6-Terra | 70-shot (many) | – | 70 | 80.3 | 80.0 | 82.5 |
| Gemini-3.7-Flash | Zero-shot | – | – | 93.3 | 93.3 | 93.8 |
| Gemini-3.7-Flash | 25-shot (few) | – | 25 | 93.3 | 93.3 | 93.3 |
| Gemini-3.7-Flash | 70-shot (many) | – | 70 | 93.1 | 93.3 | 95.0 |
| Qwen3.5-397B | Zero-shot | – | – | 86.7 | 86.7 | 86.7 |
| Qwen3.5-397B | 25-shot (few) | – | 25 | 76.8 | 76.7 | 77.2 |
| Qwen3.5-397B | 70-shot (many) | – | 70 | 89.7 | 90.0 | 89.8 |
| Qwen3.5-0.8B | Zero-shot | – | – | 19.3 | 26.7 | 17.1 |
| Qwen3.5-0.8B | SFT on all synthetic data | – | 10,041 | 77.9 ± 3.1 | 78.9 ± 2.9 | 79.8 ± 2.0 |
| Qwen3.5-0.8B | SFT on TRAK top-$N$ scored against validation | validation | 5,000 | **84.2**†‡ ± 2.1 | 84.4 ± 2.2 | 87.1 ± 2.8 |
| Qwen3.5-0.8B | SFT on TRAK top-$N$ scored against validation | test | 5,000 | **84.2**†‡ ± 2.1 | 84.4 ± 2.2 | 87.1 ± 2.8 |
| Qwen3.5-0.8B | SFT on TRAK top-$N$ scored against test (oracle) | test | 4,000 | **88.1**†‡ ± 2.8 | 87.8 ± 2.9 | 90.3 ± 2.6 |
| Qwen3.5-2B | Zero-shot | – | – | 27.5 | 30.0 | 28.7 |
| Qwen3.5-2B | SFT on all synthetic data | – | 10,041 | 77.4 ± 1.1 | 77.8 ± 1.1 | 80.3 ± 2.1 |
| Qwen3.5-2B | SFT on TRAK top-$N$ scored against validation | validation | 2,000 | 82.2†‡ ± 1.0 | 82.2 ± 1.1 | 85.9 ± 1.1 |
| Qwen3.5-2B | SFT on TRAK top-$N$ scored against validation | test | 7,000 | **85.5**†‡ ± 2.4 | 85.6 ± 2.2 | 86.6 ± 2.3 |
| Qwen3.5-2B | SFT on TRAK top-$N$ scored against test (oracle) | test | 2,000 | **90.1**†‡ ± 0.1 | 90.0 ± 0.0 | 91.2 ± 0.2 |
| Qwen3.5-4B | Zero-shot | – | – | 56.1 | 56.7 | 69.5 |
| Qwen3.5-4B | SFT on all synthetic data | – | 10,041 | 79.6 ± 3.9 | 80.0 ± 3.8 | 80.4 ± 4.4 |
| Qwen3.5-4B | SFT on TRAK top-$N$ scored against validation | validation | 3,000 | **86.4**†‡ ± 2.3 | 86.7 ± 1.9 | 88.2 ± 1.4 |
| Qwen3.5-4B | SFT on TRAK top-$N$ scored against validation | test | 5,000 | **86.6**†‡ ± 0.2 | 86.7 ± 0.0 | 88.3 ± 0.8 |
| Qwen3.5-4B | SFT on TRAK top-$N$ scored against test (oracle) | test | 5,000 | **89.9**†‡ ± 1.9 | 90.0 ± 1.9 | 92.2 ± 2.0 |

**Finding.** Choosing $N$ on the validation set, the 4B student reaches 86.4 macro-F1 and beats 1 of the 3 teachers in each prompting regime – GPT-5.6-Terra's zero-shot (+2.9) and many-shot (+6.1), and Qwen3.5-397B's few-shot (+9.6). Choosing $N$ on the test set moves this only marginally (86.6), and the test-attribution oracle raises the best student to 90.1, which additionally clears Qwen3.5-397B's zero-shot and many-shot scores. Gemini-3.7-Flash remains out of reach under every prompting regime and for every row, roughly 3–7 points ahead throughout. Notably, more shots do not uniformly help the teachers here: GPT-5.6-Terra's many-shot score (80.3) is its *weakest* of the three regimes, below both its zero-shot and its few-shot performance.

With validation-attributed selection and $N$ chosen on test, all three student sizes exceed GPT-5.6-Terra's many-shot baseline (80.29 macro-F1) using at most 7,000 of the 10,041-example candidate pool, and the 4B student (86.59 macro-F1 at $N = 5{,}000$) approaches

Qwen3.5-397B many-shot (89.67) without matching Gemini-3.7-Flash (93.14). The test-top oracle closes most of the remaining gap: the 2B and 4B students both exceed 89.9 macro-F1, ahead of Qwen3.5-397B and approximately 3.0–3.4 points behind Gemini.

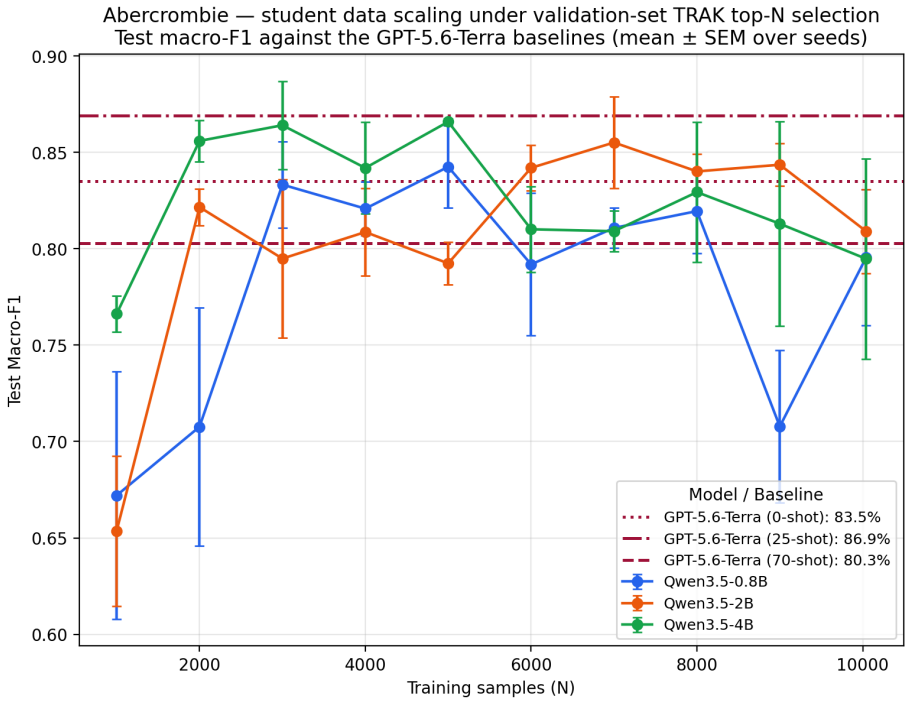



Figure 4.1: Abercrombie: test macro-F1 vs. training subset size $N$ under `validation_top` TRAK selection, across all three Qwen3.5 student sizes. Horizontal lines mark the GPT-5.6-Terra zero-shot, few-shot, and many-shot baselines from Table 4.6.

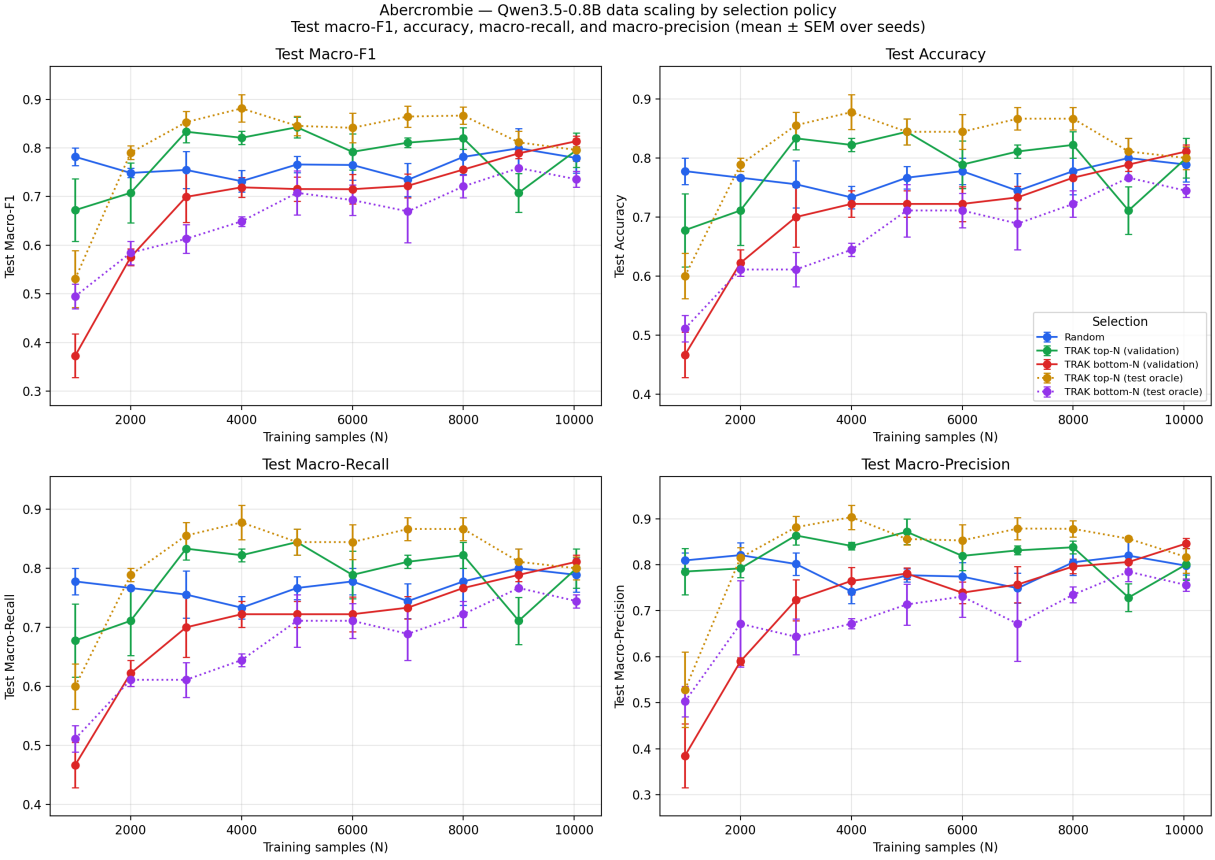



Figure 4.2: Abercrombie, Qwen3.5-0.8B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies.

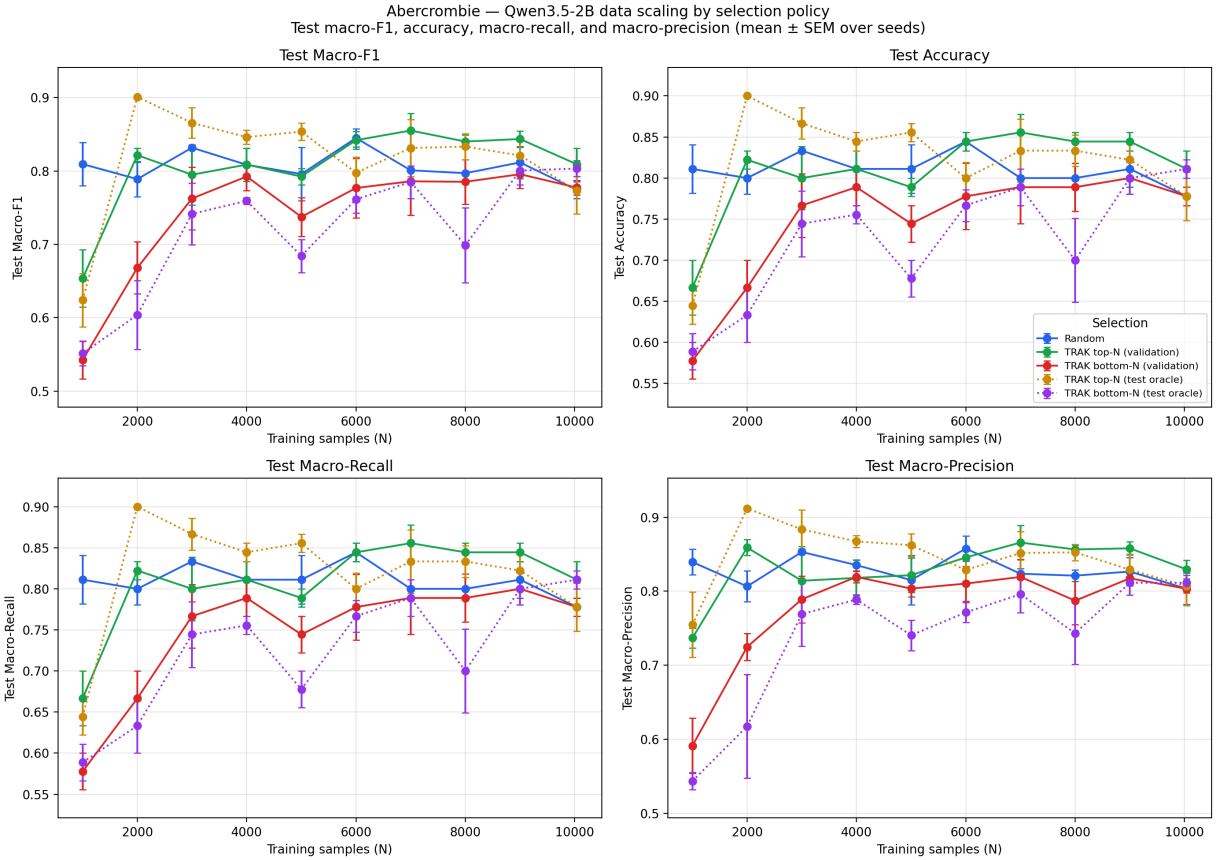



Figure 4.3: Abercrombie, Qwen3.5-2B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies.

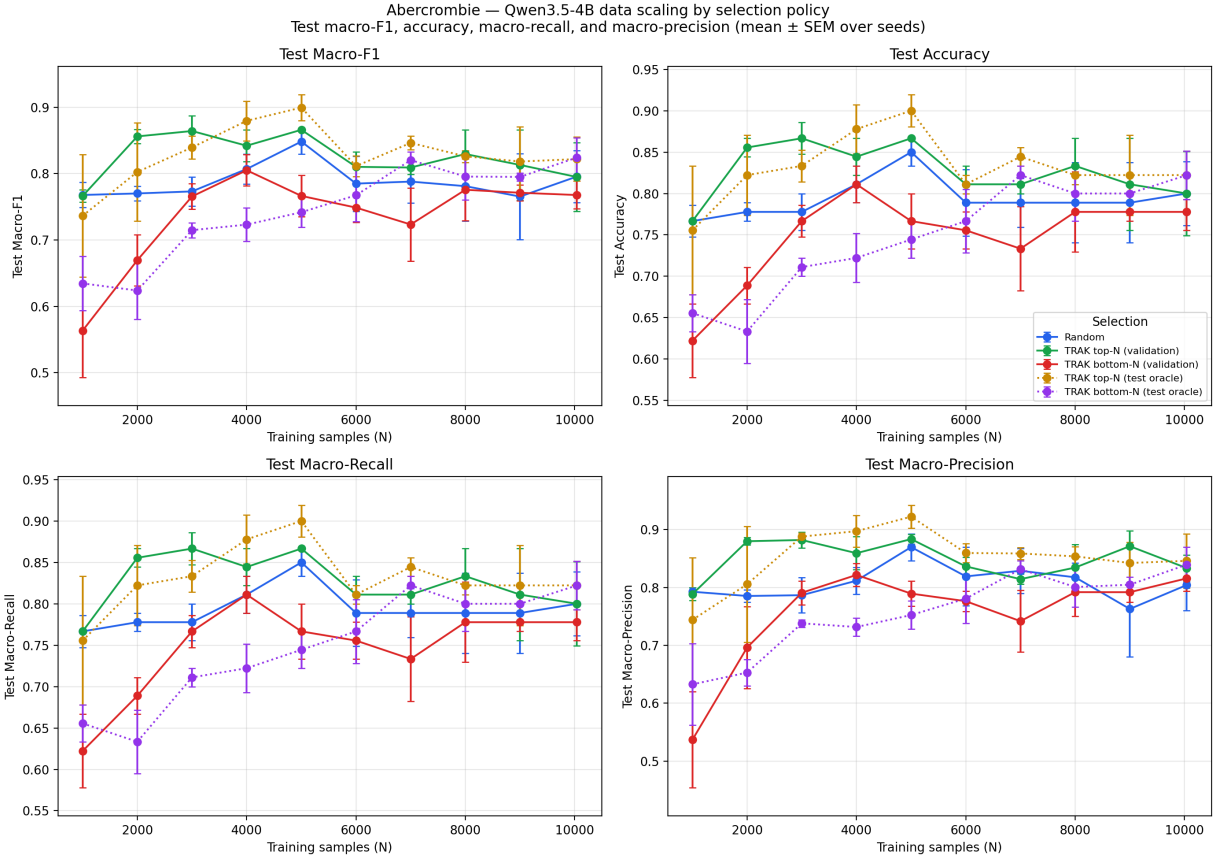



Figure 4.4: Abercrombie, Qwen3.5-4B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies.

### Insurance Policy Interpretation

Table 4.7: Teacher and student results on the *Insurance Policy Interpretation* task. In every TRAK row, **bold** macro-F1 means exceed at least one teacher's zero-shot macro-F1; † and ‡ mark means exceeding at least one teacher's few-shot and many-shot macro-F1, respectively. Each comparison uses the lowest teacher macro-F1 within that prompting regime and is assessed independently.

| Model | Method | $N$ chosen on | $N$ | Macro F1 (%) | Macro Recall (%) | Macro Precision (%) |
|---|---|---|---|---|---|---|
| GPT-5.6-Terra | Zero-shot | – | – | 59.1 | 73.3 | 71.9 |
| GPT-5.6-Terra | 15-shot (few) | – | 15 | 63.9 | 68.6 | 63.5 |
| GPT-5.6-Terra | 96-shot (many) | – | 96 | 69.6 | 71.8 | 68.8 |
| Gemini-3.7-Flash | Zero-shot | – | – | 60.4 | 68.1 | 63.5 |
| Gemini-3.7-Flash | 15-shot (few) | – | 15 | 65.8 | 71.0 | 65.9 |
| Gemini-3.7-Flash | 96-shot (many) | – | 96 | 73.9 | 77.0 | 73.1 |
| Qwen3.5-397B | Zero-shot | – | – | 52.7 | 70.0 | 70.6 |
| Qwen3.5-397B | 15-shot (few) | – | 15 | 55.3 | 64.3 | 58.7 |
| Qwen3.5-397B | 96-shot (many) | – | 96 | 72.3 | 76.7 | 72.3 |
| Qwen3.5-0.8B | Zero-shot | – | – | 21.1 | 32.7 | 15.8 |
| Qwen3.5-0.8B | SFT on all synthetic data | – | 26,385 | 60.8 ± 2.6 | 61.8 ± 2.2 | 60.8 ± 2.5 |
| Qwen3.5-0.8B | SFT on TRAK top-$N$ scored against validation | validation | 15,000 | **61.7**† ± 0.7 | 62.1 ± 0.5 | 62.2 ± 1.3 |
| Qwen3.5-0.8B | SFT on TRAK top-$N$ scored against validation | test | 12,500 | **65.6**† ± 1.3 | 66.1 ± 1.9 | 68.3 ± 0.8 |
| Qwen3.5-0.8B | SFT on TRAK top-$N$ scored against test (oracle) | test | 10,000 | **68.5**† ± 3.7 | 67.7 ± 3.3 | 76.0 ± 4.4 |
| Qwen3.5-2B | Zero-shot | – | – | 25.7 | 38.2 | 66.5 |
| Qwen3.5-2B | SFT on all synthetic data | – | 26,385 | 57.6 ± 3.1 | 59.9 ± 3.3 | 57.8 ± 3.0 |
| Qwen3.5-2B | SFT on TRAK top-$N$ scored against validation | validation | 5,000 | **68.7**† ± 2.4 | 66.7 ± 2.1 | 73.2 ± 3.3 |
| Qwen3.5-2B | SFT on TRAK top-$N$ scored against validation | test | 2,500 | **69.7**†‡ ± 2.8 | 67.7 ± 2.9 | 74.7 ± 1.7 |
| Qwen3.5-2B | SFT on TRAK top-$N$ scored against test (oracle) | test | 5,000 | **71.7**†‡ ± 0.7 | 71.2 ± 1.2 | 74.6 ± 1.3 |
| Qwen3.5-4B | Zero-shot | – | – | 35.6 | 53.0 | 28.9 |
| Qwen3.5-4B | SFT on all synthetic data | – | 26,385 | 64.2 ± 0.8 | 64.8 ± 1.6 | 66.6 ± 1.4 |
| Qwen3.5-4B | SFT on TRAK top-$N$ scored against validation | validation | 7,500 | **65.4**† ± 5.1 | 64.0 ± 5.0 | 71.5 ± 4.8 |
| Qwen3.5-4B | SFT on TRAK top-$N$ scored against validation | test | 5,000 | **70.7**†‡ ± 3.3 | 69.4 ± 3.5 | 76.8 ± 3.2 |
| Qwen3.5-4B | SFT on TRAK top-$N$ scored against test (oracle) | test | 15,000 | **72.7**†‡ ± 3.3 | 70.8 ± 3.3 | 77.5 ± 3.4 |

**Finding.** This is the task where students compete most strongly. With $N$ chosen on the validation set, the best student (2B, 68.7 macro-F1) beats all three teachers under both zero-shot (+8.3–16.0) and few-shot (+2.9–13.4) prompting, but none under many-shot. Choosing $N$ on the test set lifts the best student to 70.7 (4B), which additionally passes GPT-5.6-Terra's many-shot score by 1.1 points, and the test-attribution oracle reaches 72.7, clearing two of the three many-shot baselines. Only Gemini-3.7-Flash's many-shot result (73.9) resists every student configuration.

With $N$ chosen on test, validation-attributed selection lets the 2B and 4B students (69.72 and 70.72 macro-F1) match or slightly exceed GPT-5.6-Terra's many-shot baseline (69.63), while all three sizes remain below Gemini-3.7-Flash (73.92) and Qwen3.5-397B (72.33) many-shot. The test-top oracle raises the 4B student to 72.69 macro-F1, ahead of GPT-5.6-Terra and level with Qwen3.5-397B, but falls behind Gemini by 1.2 percentage points.

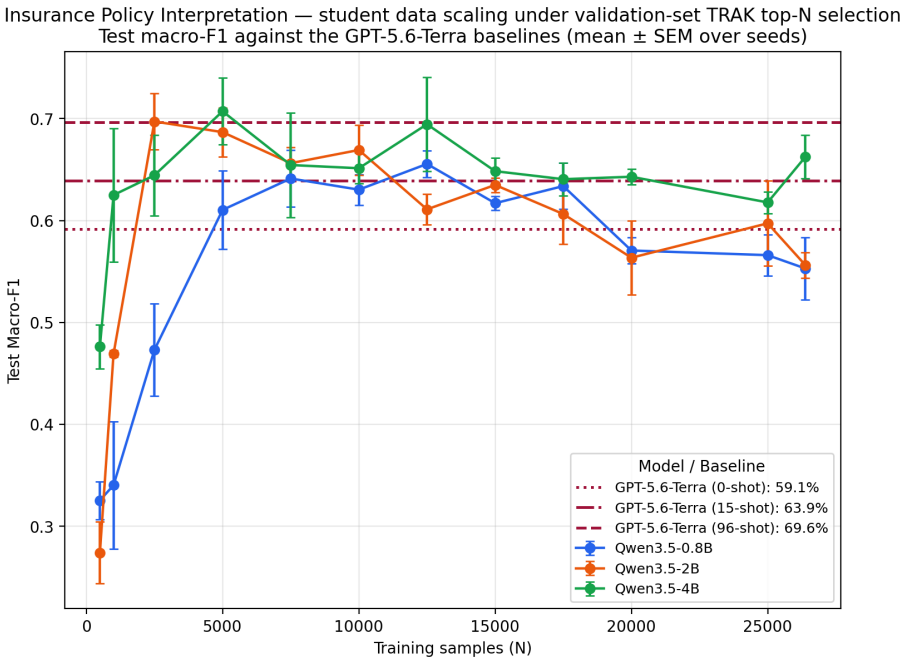



Figure 4.5: Insurance Policy Interpretation: test macro-F1 vs. training subset size $N$ under `validation_top` TRAK selection, across all three Qwen3.5 student sizes. Horizontal lines mark the GPT-5.6-Terra zero-shot, few-shot, and many-shot baselines from Table 4.7.

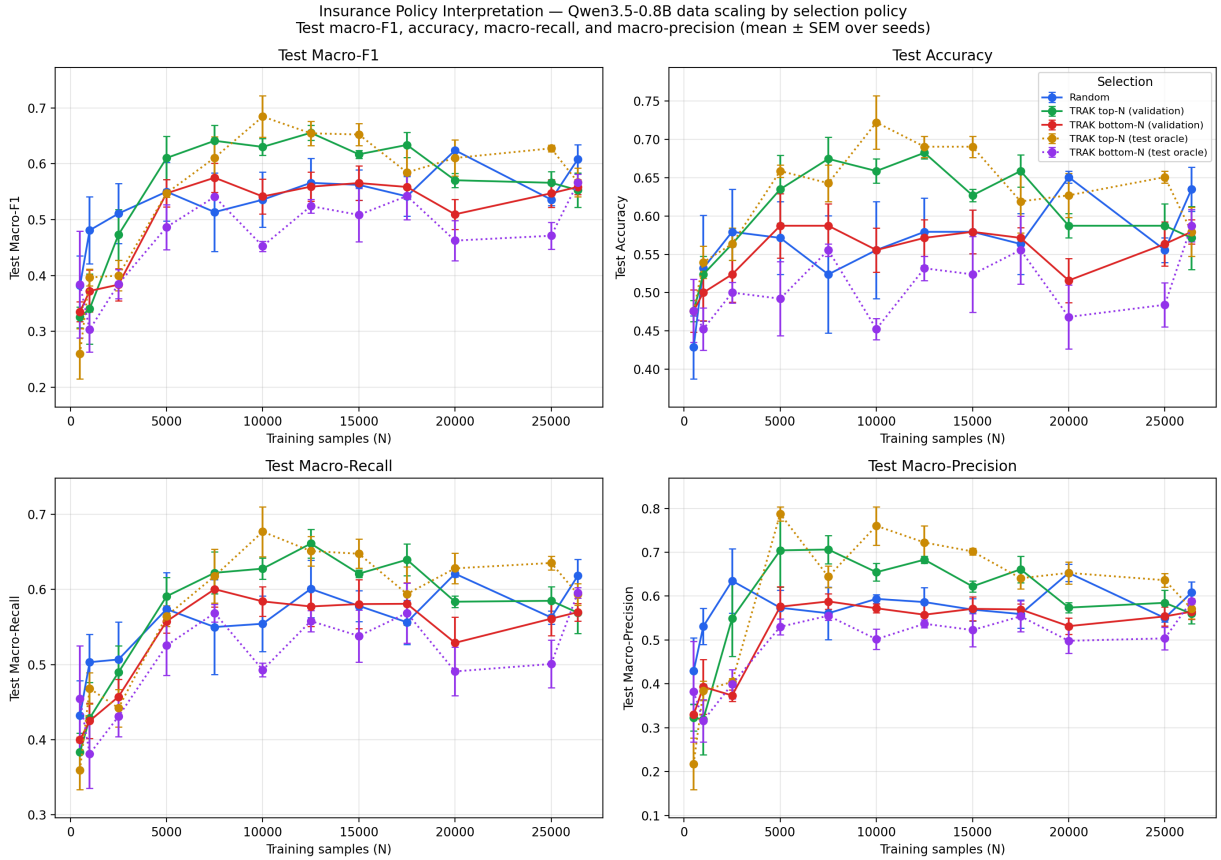



Figure 4.6: Insurance Policy Interpretation, Qwen3.5-0.8B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies.

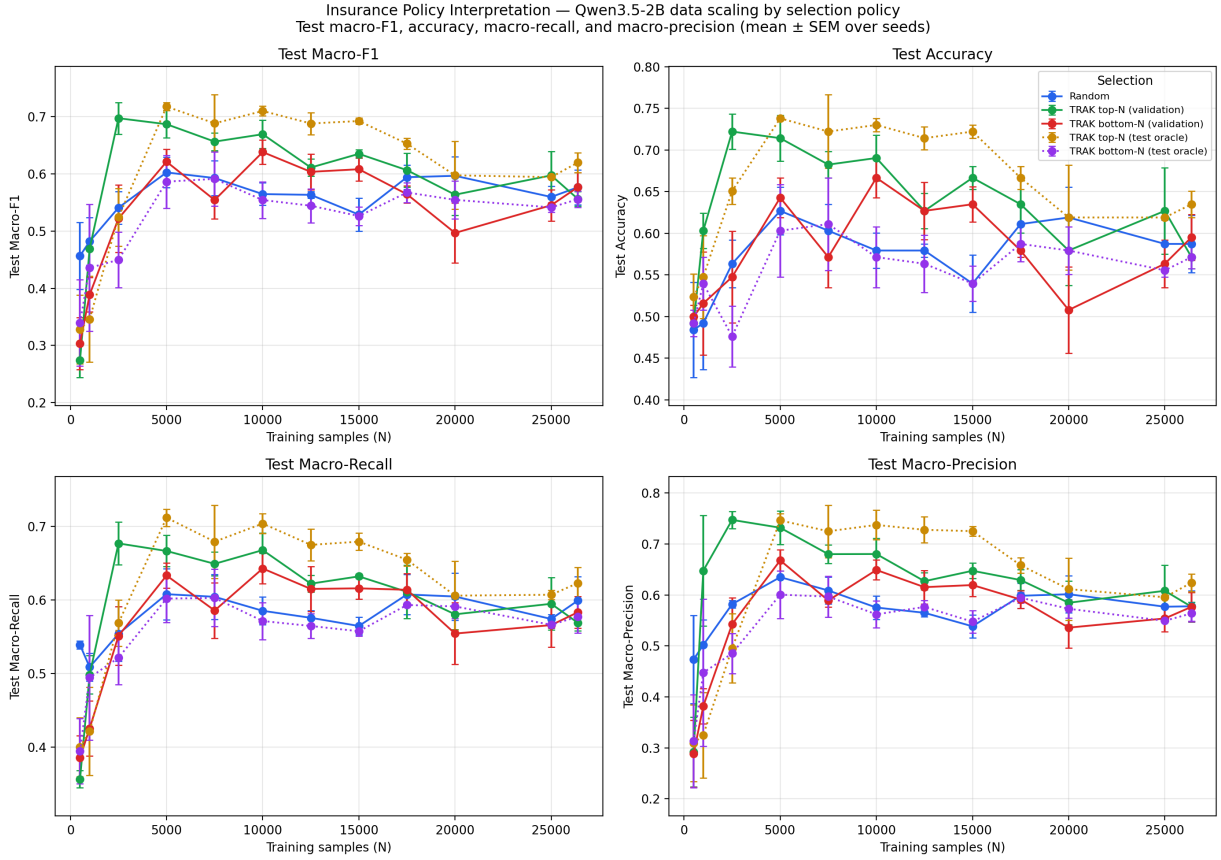



Figure 4.7: Insurance Policy Interpretation, Qwen3.5-2B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies.

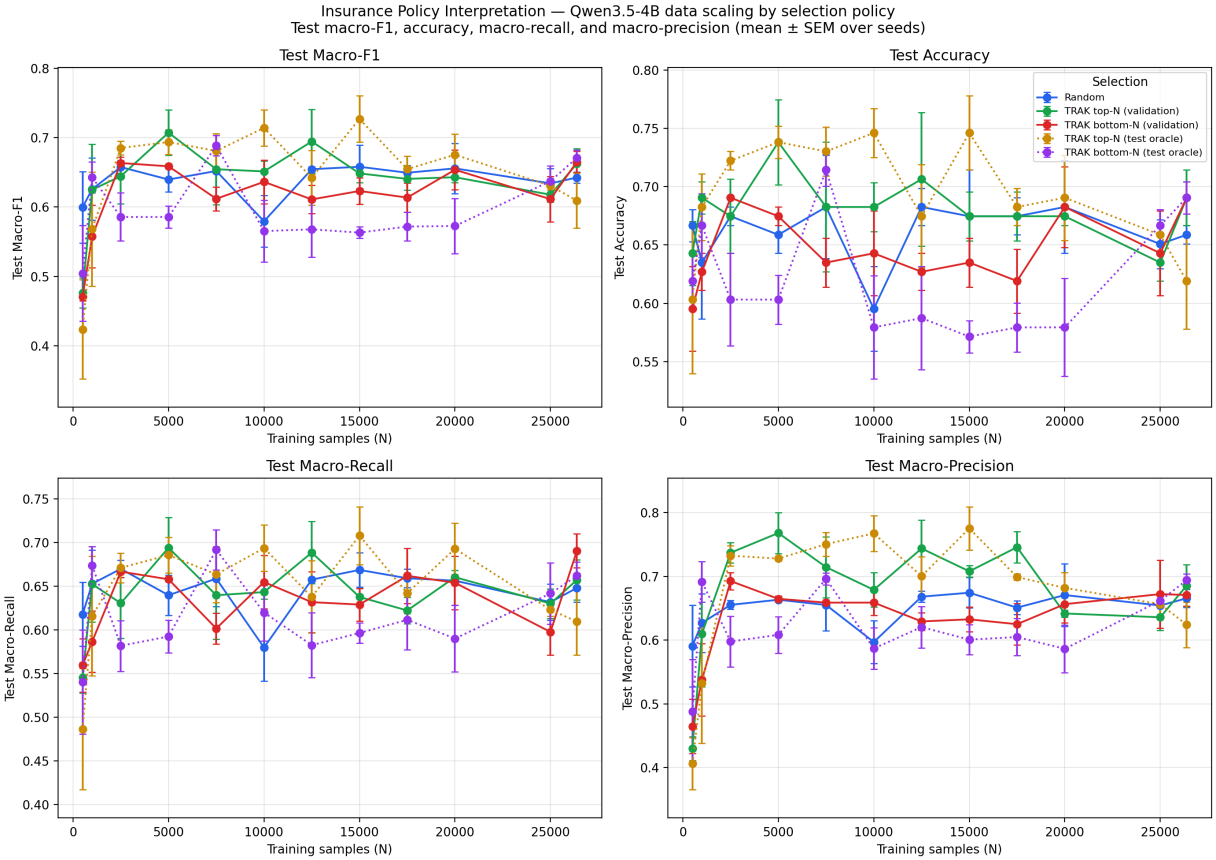



Figure 4.8: Insurance Policy Interpretation, Qwen3.5-4B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies.

## Oral Argument Question Purpose

Table 4.8: Teacher and student results on the *Oral Argument Question Purpose* task. In every TRAK row, **bold** macro-F1 means exceed at least one teacher's zero-shot macro-F1; † and ‡ mark means exceeding at least one teacher's few-shot and many-shot macro-F1, respectively. Each comparison uses the lowest teacher macro-F1 within that prompting regime and is assessed independently.

| Model | Method | $N$ chosen on | $N$ | Macro F1 (%) | Macro Recall (%) | Macro Precision (%) |
|---|---|---|---|---|---|---|
| GPT-5.6-Terra | Zero-shot | – | – | 45.9 | 45.3 | 51.2 |
| GPT-5.6-Terra | 35-shot (few) | – | 35 | 49.0 | 48.8 | 53.9 |
| GPT-5.6-Terra | 223-shot (many) | – | 223 | 53.4 | 53.8 | 57.8 |
| Gemini-3.7-Flash | Zero-shot | – | – | 45.1 | 43.9 | 54.4 |
| Gemini-3.7-Flash | 35-shot (few) | – | 35 | 51.1 | 53.8 | 55.5 |
| Gemini-3.7-Flash | 223-shot (many) | – | 223 | 63.2 | 61.8 | 69.9 |
| Qwen3.5-397B | Zero-shot | – | – | 38.5 | 38.2 | 44.3 |
| Qwen3.5-397B | 35-shot (few) | – | 35 | 52.6 | 52.5 | 59.6 |
| Qwen3.5-397B | 223-shot (many) | – | 223 | 58.3 | 56.5 | 63.5 |
| Qwen3.5-0.8B | Zero-shot | – | – | 12.6 | 12.3 | 28.8 |
| Qwen3.5-0.8B | SFT on all synthetic data | – | 25,925 | 41.2 ± 4.0 | 44.6 ± 4.5 | 42.8 ± 2.6 |
| Qwen3.5-0.8B | SFT on TRAK top-$N$ scored against validation | validation | 17,500 | **39.3** ± 1.3 | 41.3 ± 2.0 | 46.9 ± 1.2 |
| Qwen3.5-0.8B | SFT on TRAK top-$N$ scored against validation | test | 20,000 | **46.2** ± 2.5 | 48.1 ± 2.9 | 51.4 ± 3.8 |
| Qwen3.5-0.8B | SFT on TRAK top-$N$ scored against test (oracle) | test | 17,500 | **54.1**†‡ ± 1.9 | 56.6 ± 2.2 | 58.2 ± 1.7 |
| Qwen3.5-2B | Zero-shot | – | – | 17.1 | 21.0 | 19.4 |
| Qwen3.5-2B | SFT on all synthetic data | – | 25,925 | 46.0 ± 2.3 | 50.1 ± 3.2 | 47.4 ± 2.6 |
| Qwen3.5-2B | SFT on TRAK top-$N$ scored against validation | validation | 10,000 | **46.3** ± 2.3 | 49.0 ± 2.9 | 51.4 ± 3.9 |
| Qwen3.5-2B | SFT on TRAK top-$N$ scored against validation | test | 17,500 | **52.7**† ± 3.3 | 55.2 ± 3.3 | 58.6 ± 1.3 |
| Qwen3.5-2B | SFT on TRAK top-$N$ scored against test (oracle) | test | 12,500 | **50.7**† ± 2.0 | 53.8 ± 2.7 | 58.0 ± 0.5 |
| Qwen3.5-4B | Zero-shot | – | – | 36.8 | 35.9 | 41.9 |
| Qwen3.5-4B | SFT on all synthetic data | – | 25,925 | 45.3 ± 2.9 | 50.1 ± 3.0 | 48.1 ± 2.0 |
| Qwen3.5-4B | SFT on TRAK top-$N$ scored against validation | validation | 15,000 | **49.7**† ± 1.1 | 53.0 ± 1.5 | 53.6 ± 2.8 |
| Qwen3.5-4B | SFT on TRAK top-$N$ scored against validation | test | 15,000 | **49.7**† ± 1.1 | 53.0 ± 1.5 | 53.6 ± 2.8 |
| Qwen3.5-4B | SFT on TRAK top-$N$ scored against test (oracle) | test | 12,500 | **56.9**†‡ ± 1.6 | 59.9 ± 1.1 | 60.2 ± 3.6 |

**Finding.** With $N$ chosen on the validation set, the best student (4B, 49.7 macro-F1) beats all three teachers' zero-shot performance (+3.8–11.2) but only GPT-5.6-Terra's few-shot score (49.0), and no many-shot baseline. Choosing $N$ on the test set raises the best student to 52.7 (2B), which then clears every teacher's few-shot score as well (+0.1–3.7), and the test-attribution oracle reaches 56.9 (4B), passing GPT-5.6-Terra's many-shot result. This 7-class task is harder for every model than the previous two, and the stronger many-shot baselines – Gemini-3.7-Flash at 63.2 and Qwen3.5-397B at 58.3 – stay ahead of all student configurations.

With $N$ chosen on test, validation-attributed selection brings the 2B student to 52.75 macro-F1, just below GPT-5.6-Terra's many-shot baseline (53.38) and below Gemini's (63.22) and Qwen3.5-397B's (58.28) many-shot (223-shot) performances. The test-top oracle is more informative here than for the previous two tasks: the 4B student reaches 56.92 macro-F1, ahead of GPT-5.6-Terra but still short of both remaining teachers' many-shot results.

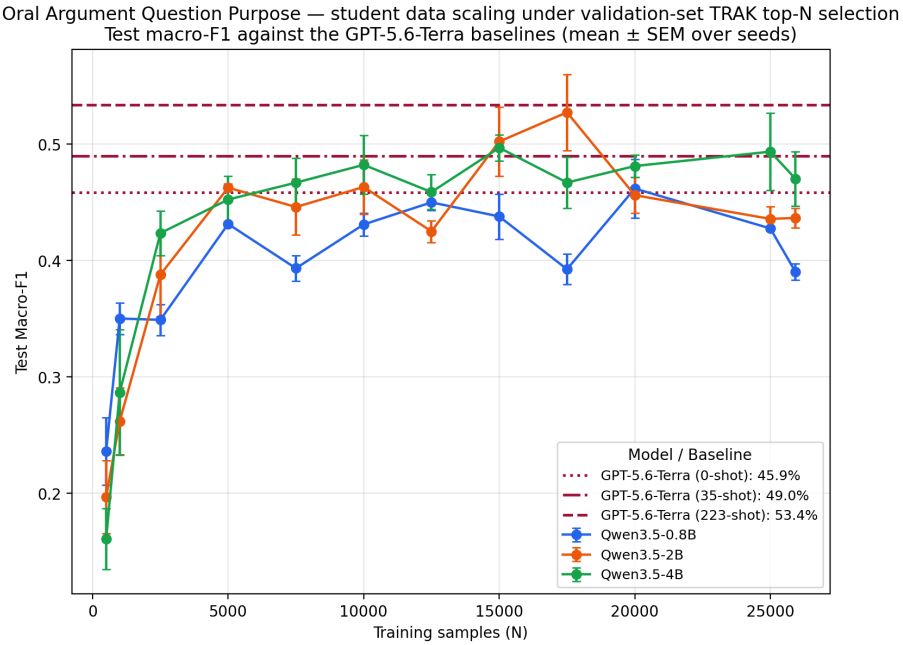



Figure 4.9: Oral Argument Question Purpose: test macro-F1 vs. training subset size $N$ under `validation_top` TRAK selection, across all three Qwen3.5 student sizes. Horizontal lines mark the GPT-5.6-Terra zero-shot, few-shot, and many-shot baselines from Table 4.8.

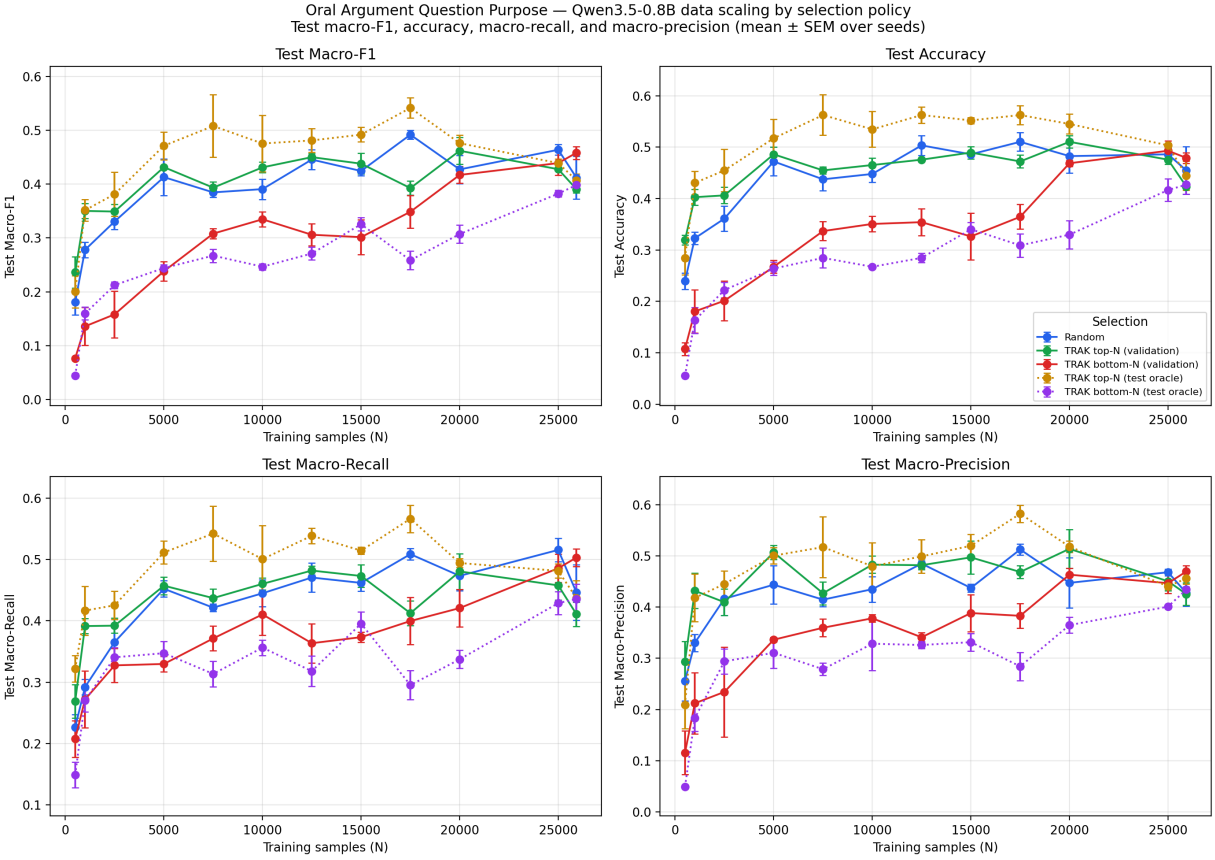



Figure 4.10: Oral Argument Question Purpose, Qwen3.5-0.8B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies.

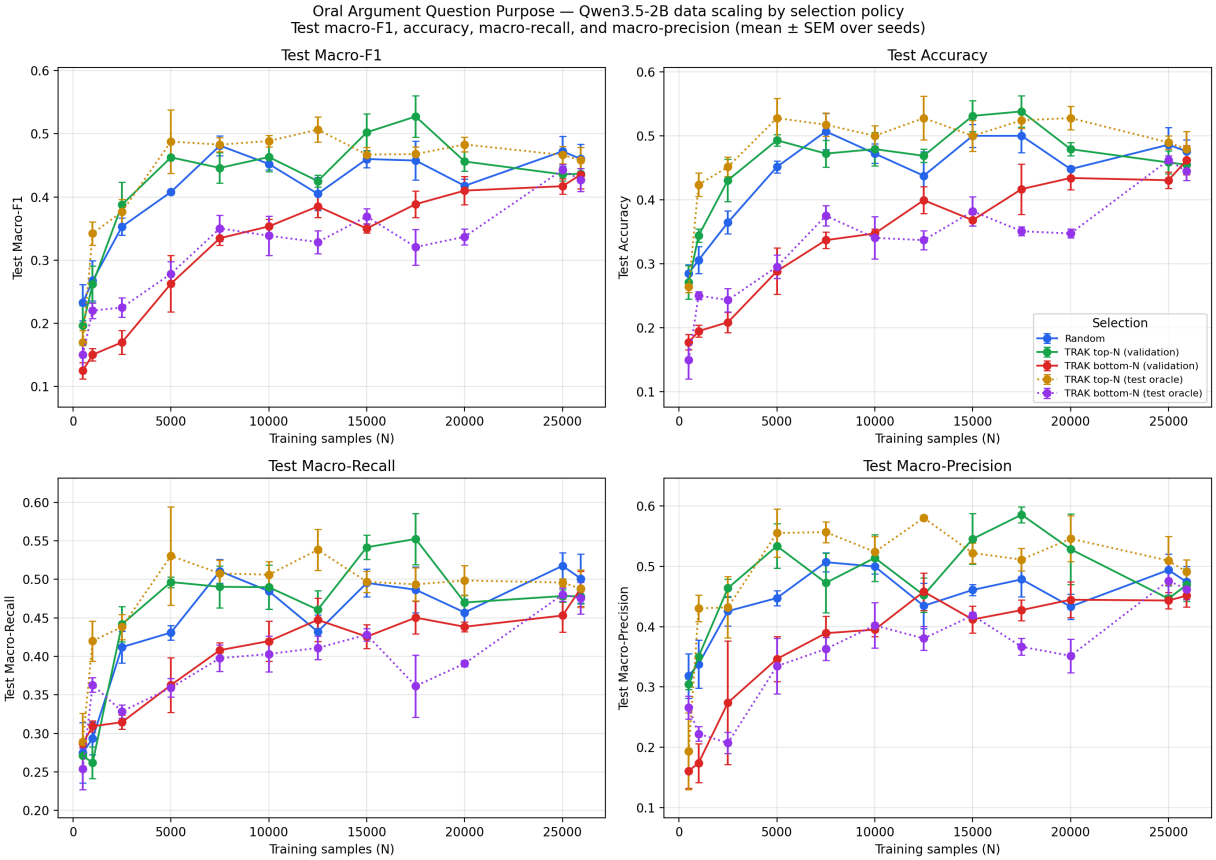



Figure 4.11: Oral Argument Question Purpose, Qwen3.5-2B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies.

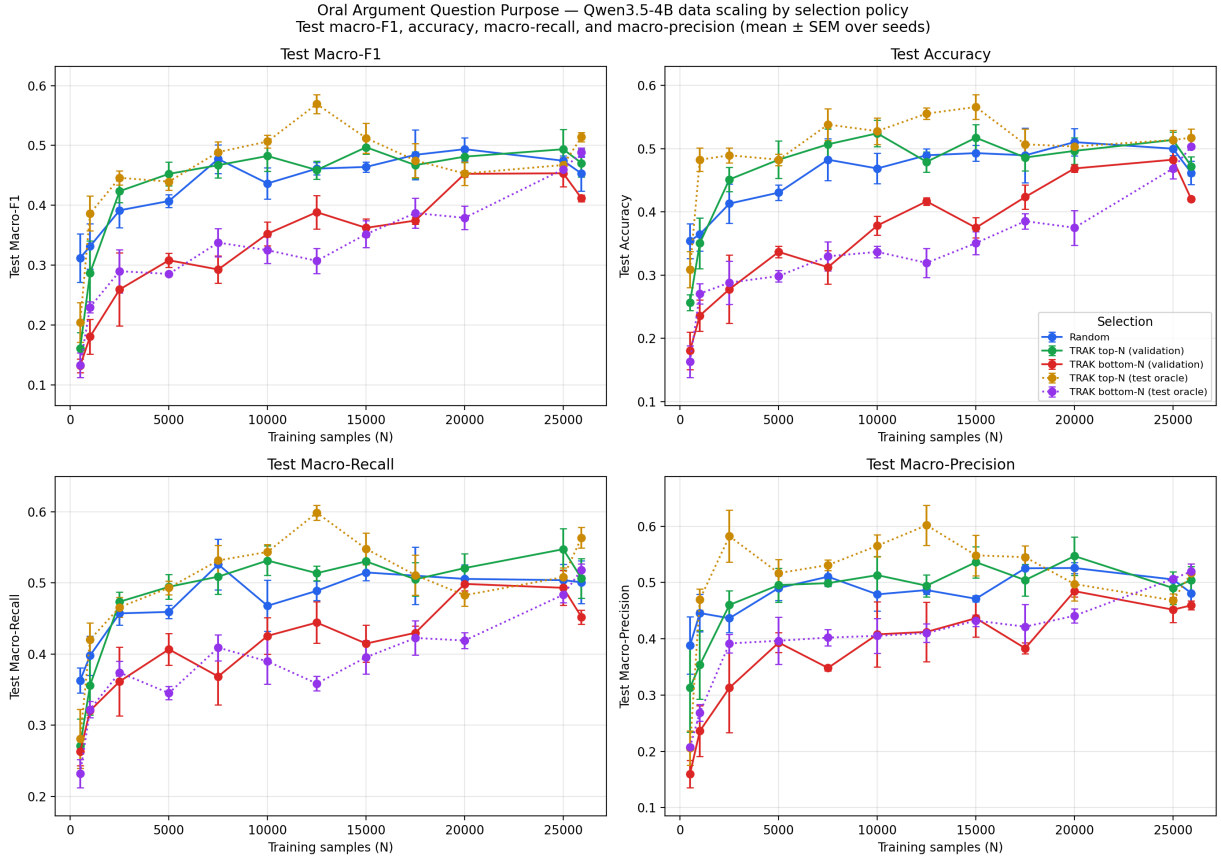



Figure 4.12: Oral Argument Question Purpose, Qwen3.5-4B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies.

### Function of Decision Section

Table 4.9: Teacher and student results on the *Function of Decision Section* task. In every TRAK row, **bold** macro-F1 means exceed at least one teacher's zero-shot macro-F1; † and ‡ mark means exceeding at least one teacher's few-shot and many-shot macro-F1, respectively. Each comparison uses the lowest teacher macro-F1 within that prompting regime and is assessed independently.

| Model | Method | $N$ chosen on | $N$ | Macro F1 (%) | Macro Recall (%) | Macro Precision (%) |
|---|---|---|---|---|---|---|
| GPT-5.6-Terra | Zero-shot | – | – | 54.8 | 53.9 | 57.3 |
| GPT-5.6-Terra | 35-shot (few) | – | 35 | 52.2 | 52.4 | 52.4 |
| GPT-5.6-Terra | 262-shot (many) | – | 262 | 58.9 | 59.1 | 60.1 |
| Gemini-3.7-Flash | Zero-shot | – | – | 47.7 | 49.6 | 50.5 |
| Gemini-3.7-Flash | 35-shot (few) | – | 35 | 55.4 | 56.5 | 56.1 |
| Gemini-3.7-Flash | 262-shot (many) | – | 262 | 57.2 | 58.3 | 58.7 |
| Qwen3.5-397B | Zero-shot | – | – | 48.2 | 48.3 | 52.8 |
| Qwen3.5-397B | 35-shot (few) | – | 35 | 53.8 | 54.0 | 55.3 |
| Qwen3.5-397B | 262-shot (many) | – | 262 | 55.6 | 55.5 | 57.4 |
| Qwen3.5-0.8B | Zero-shot | – | – | 8.5 | 17.1 | 7.3 |
| Qwen3.5-0.8B | SFT on all synthetic data | – | 25,925 | 44.4 ± 3.1 | 44.7 ± 3.2 | 46.5 ± 2.4 |
| Qwen3.5-0.8B | SFT on TRAK top-$N$ scored against validation | validation | 12,500 | 46.9 ± 1.5 | 47.3 ± 1.3 | 48.7 ± 1.9 |
| Qwen3.5-0.8B | SFT on TRAK top-$N$ scored against validation | test | 12,500 | 46.9 ± 1.5 | 47.3 ± 1.3 | 48.7 ± 1.9 |
| Qwen3.5-0.8B | SFT on TRAK top-$N$ scored against test (oracle) | test | 12,500 | **48.3** ± 0.6 | 48.7 ± 0.6 | 49.5 ± 1.3 |
| Qwen3.5-2B | Zero-shot | – | – | 21.5 | 23.7 | 25.2 |
| Qwen3.5-2B | SFT on all synthetic data | – | 25,925 | 46.9 ± 1.2 | 47.1 ± 0.9 | 48.5 ± 1.0 |
| Qwen3.5-2B | SFT on TRAK top-$N$ scored against validation | validation | 15,000 | 44.8 ± 2.2 | 45.1 ± 2.2 | 48.3 ± 1.7 |
| Qwen3.5-2B | SFT on TRAK top-$N$ scored against validation | test | 7,500 | **47.9** ± 1.8 | 47.9 ± 1.9 | 50.5 ± 1.3 |
| Qwen3.5-2B | SFT on TRAK top-$N$ scored against test (oracle) | test | 12,500 | **51.7** ± 0.9 | 51.5 ± 0.7 | 53.4 ± 1.2 |
| Qwen3.5-4B | Zero-shot | – | – | 40.5 | 42.3 | 46.1 |
| Qwen3.5-4B | SFT on all synthetic data | – | 25,925 | 45.4 ± 2.1 | 46.0 ± 1.9 | 47.1 ± 1.3 |
| Qwen3.5-4B | SFT on TRAK top-$N$ scored against validation | validation | 7,500 | 44.8 ± 1.5 | 45.3 ± 1.6 | 47.3 ± 1.5 |
| Qwen3.5-4B | SFT on TRAK top-$N$ scored against validation | test | 17,500 | **49.3** ± 1.5 | 48.9 ± 1.6 | 51.5 ± 2.1 |
| Qwen3.5-4B | SFT on TRAK top-$N$ scored against test (oracle) | test | 12,500 | **51.1** ± 3.1 | 51.3 ± 3.0 | 53.1 ± 3.8 |

**Finding.** This is the weakest task for the students, as with $N$ chosen on the validation set, the best student (0.8B, 46.9 macro-F1) beats no teacher baseline in any prompting regime. Choosing $N$ on the test set raises the best student to 49.3 (4B), which passes Gemini-3.7-Flash's and Qwen3.5-397B's zero-shot scores (+1.6 and +1.1) but nothing else, and even the test-attribution oracle (51.7) adds no further wins beyond those two zero-shot baselines. Every teacher, under any prompting regime, scores at or above 47.7 macro-F1. Hence, the fine-tuned students beat few zero-shot teachers while stays behind the few-shot and many-shot settings even when using test as the attribution signal.

**Why top-$N$ selection is weakest here.** Function of Decision Section is also the one task where `validation_bottom`, taken at its own best $N$, matches or exceeds `validation_top` as depicted in Figure 4.14 (48.2 vs. 46.9 for the 0.8B student). This is not evidence that the score is reversed. The cause is visible in the class composition of the selected subsets: mean attribution on this task is dominated by two classes, and the top-2,500 candidates are

40% *Facts* and 33% *Procedural History*, against a near-uniform pool. At small $N$, top-$N$ therefore trains what is effectively a two-class model on a seven-class task, and macro-F1 collapses to 9.4 at $N = 500$ while random selection reaches 40.5. Bottom-$N$ wins early only because it is less class-skewed. Once $N$ is large enough for coverage to recover, the ordering returns to the expected one: at a matched $N = 12{,}500$, `validation_top` reaches 46.9 against 42.7 for `validation_bottom` and 42.2 for random. The apparent advantage of bottom selection is thus an artifact of comparing each policy at its own best $N$ on curves that converge as $N$ approaches the pool size. Detailed analysis of attribution signals across different classes is provided in Section 4.2.2.

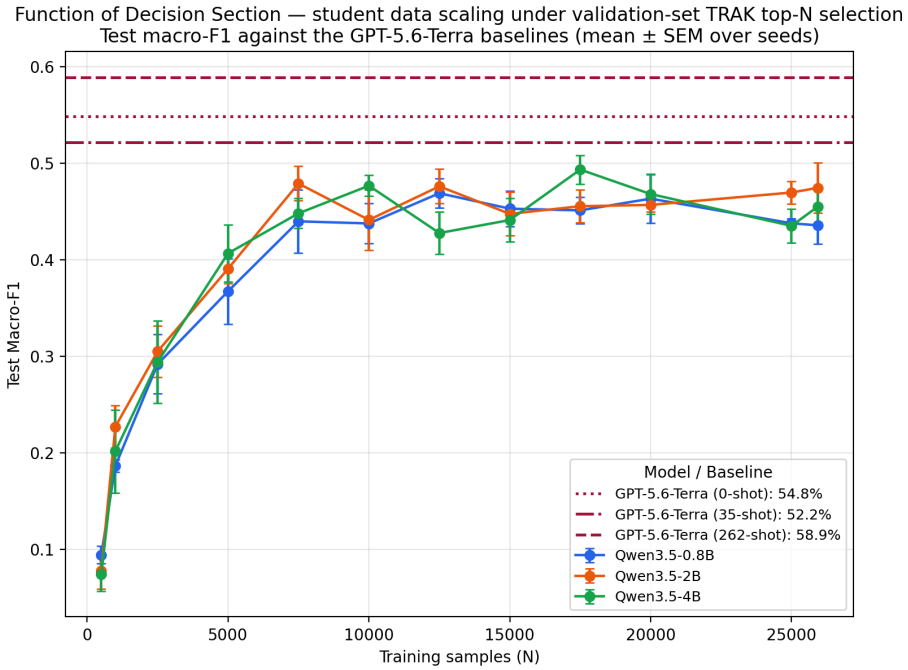



Figure 4.13: Function of Decision Section: test macro-F1 vs. training subset size $N$ under `validation_top` TRAK selection, across all three Qwen3.5 student sizes. Horizontal lines mark the GPT-5.6-Terra zero-shot, few-shot, and many-shot baselines from Table 4.9.

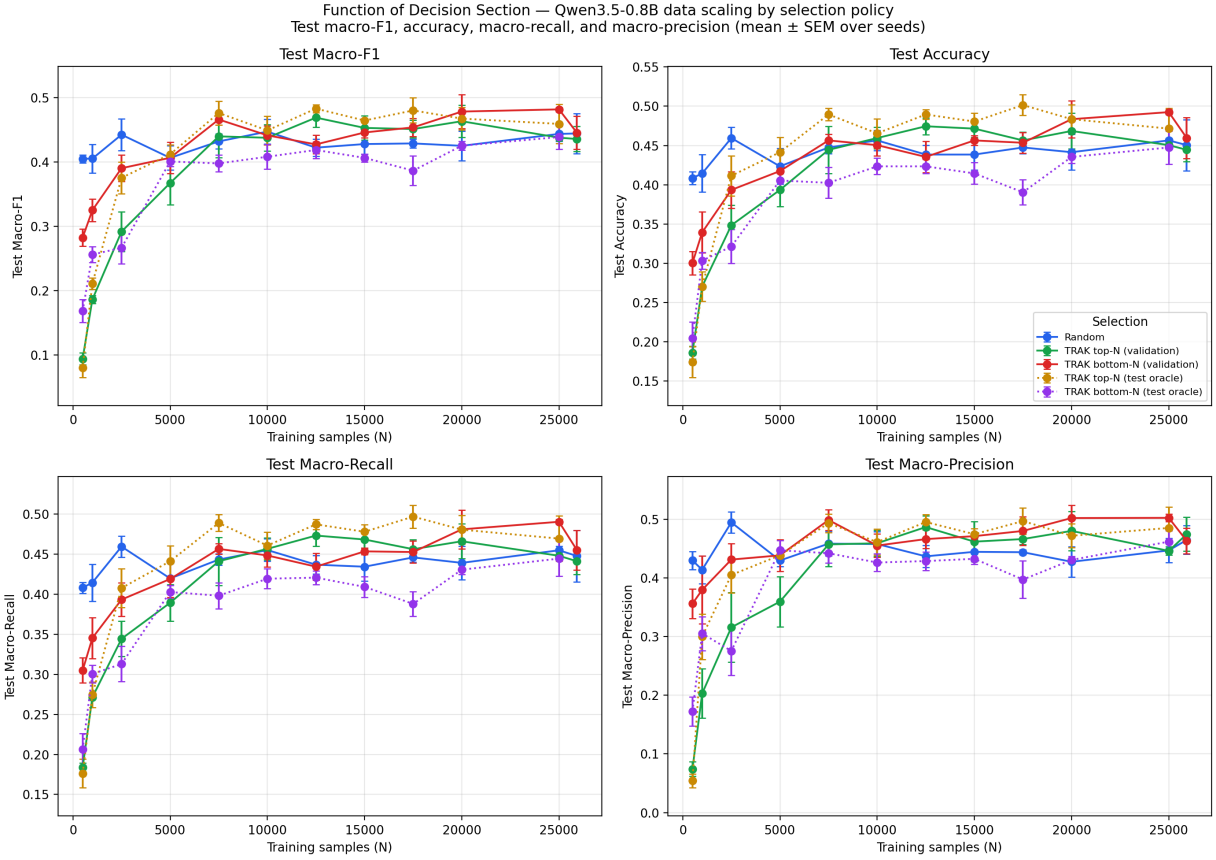



Figure 4.14: Function of Decision Section, Qwen3.5-0.8B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies.

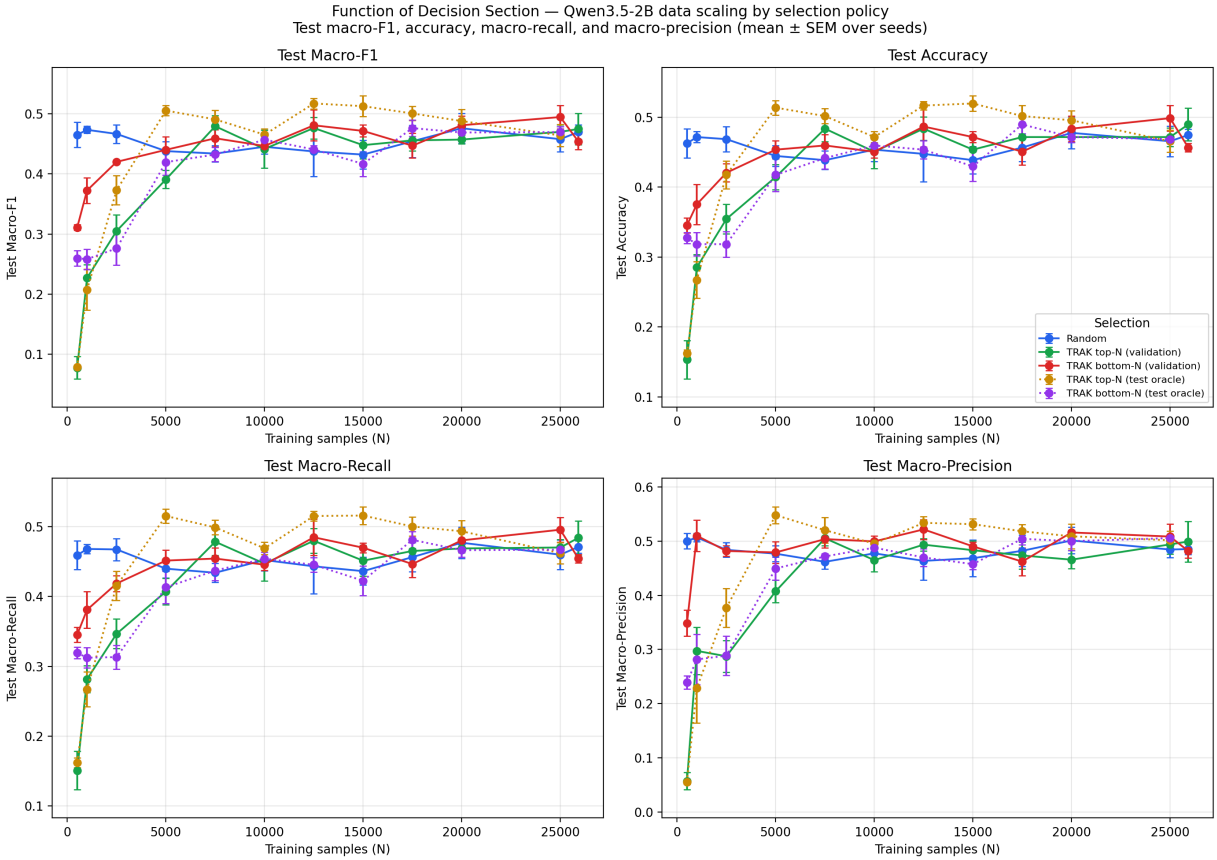



Figure 4.15: Function of Decision Section, Qwen3.5-2B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies.

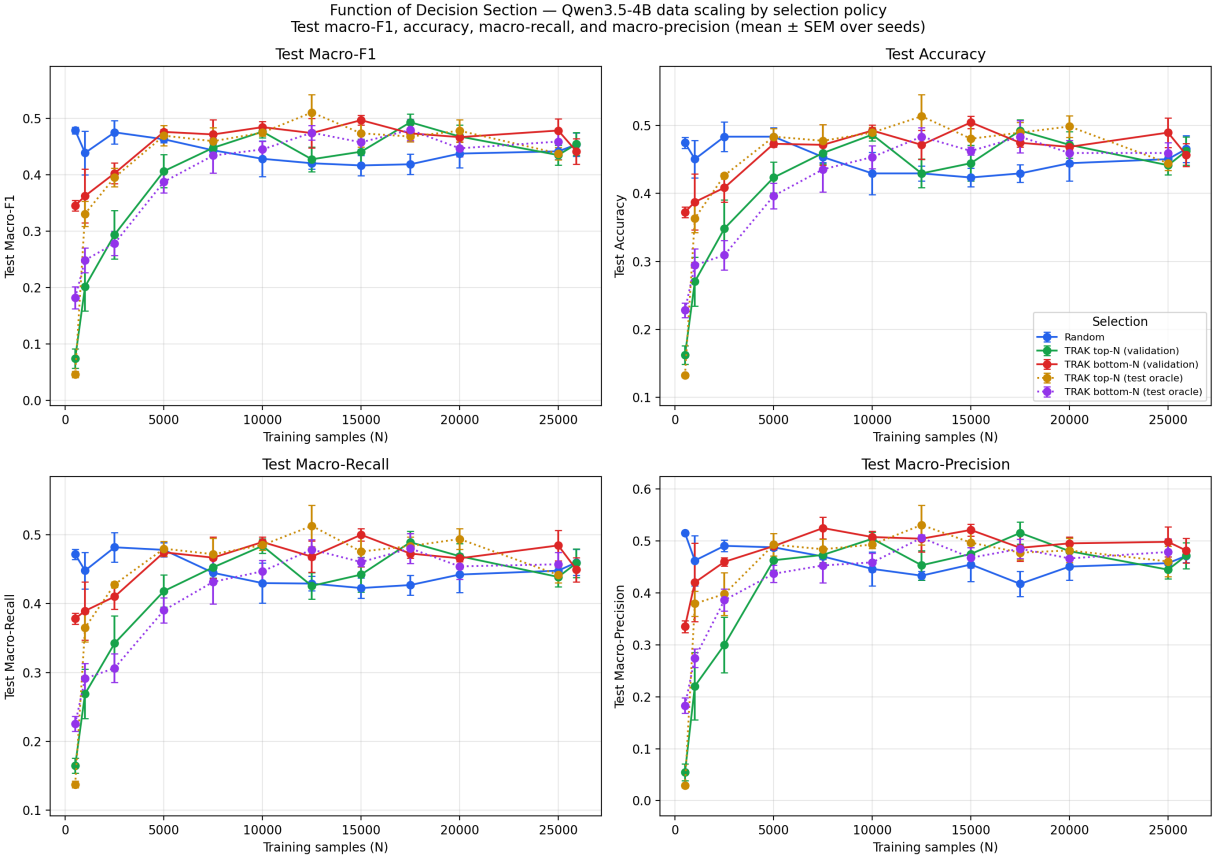



Figure 4.16: Function of Decision Section, Qwen3.5-4B: test macro-F1, accuracy, macro-recall, and macro-precision vs. $N$ under all five selection policies.

### 4.2.2 Benchmarking Synthesis Prompt Strategies via Attribution Analysis

The previous section used TRAK to select synthetic examples for fine-tuning. The same scores can also profile the quality and utility of the data produced by each synthesis prompt strategy. Here, we evaluate the prompt strategies described in Section 3.2 from the following perspectives:

- **Sample-wise analysis:** What portion of the synthetic pool is useful, harmful, or has little estimated effect?
- **Prompt analysis:** Which synthesis prompt produce useful examples more consistently?
- **Class-wise analysis:** Which validation classes does each prompt's generated dataset support or harm?

Let $T[x_i, q_j]$ be the TRAK score between synthetic candidate $x_i$ and validation example $q_j$. A positive value means that $x_i$ is estimated to support the model on $q_j$, while a negative value means that it is estimated to work against it. A value close to zero means that no meaningful effect was recorded. Because TRAK score scales differ across validation examples, we $L_1$-normalize each validation column by its total absolute attribution:

$$\widetilde{T}[x_i, q_j] = \frac{T[x_i, q_j]}{\sum_{r=1}^{|\mathcal{D}_{\text{syn}}|} |T[x_r, q_j]| + \epsilon}. \tag{4.1}$$

where $\epsilon = 10^{-12}$ is a small constant that prevents division by zero. All results below use $\mathcal{D}_{\text{val}}$, rather than the test set, as the validation set. They also use the Qwen3.5-0.8B reference model described in the experimental settings. The results therefore measure usefulness of the training data on this model and the validation set, in terms of contributing to correct prediction.

**Sample-wise analysis.** We begin with the attribution profile of each synthetic example. We average its normalized scores across the validation set and record whether the result is positive, exactly zero, or negative. Figure 4.17 shows the proportion of the complete synthetic pool in each group. This indicates how much of the generated pool provides useful training signal, has no recorded average effect, or may hurt the prediction.

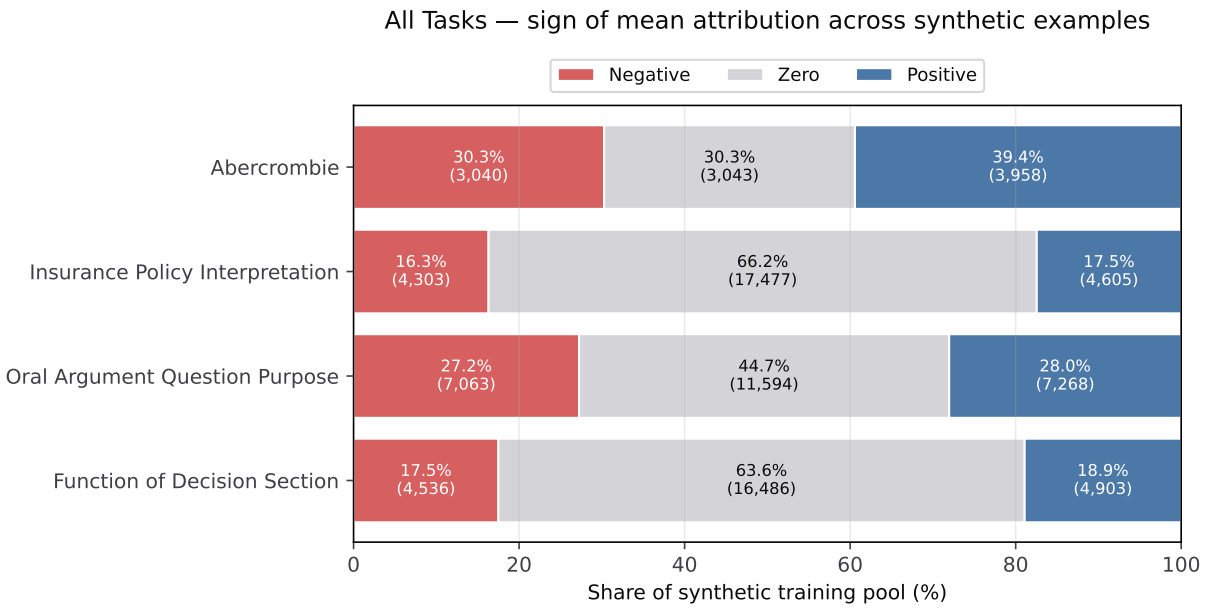



Figure 4.17: Composition of each synthetic training pool by the sign of its mean normalized attribution across validation examples. Zero denotes an exactly zero mean; no near-zero threshold is applied. Percentages and sample counts are shown within each segment.

The proportions of positive, zero, and negative mean attribution vary across tasks despite the use of the same synthesis framework. Exactly zero mean attribution accounts for 66.2% of the Insurance Policy Interpretation pool and 63.6% of the Function of Decision Section pool, while it is 30.3% and 44.7% for the Abercrombie and Oral Argument Question Purpose tasks. Abercrombie contains the largest positive share at 39.4%, while its positive, zero, and negative groups are comparatively balanced. Oral Argument Question Purpose is also balanced between positive and negative examples, at 28.0% and 27.2%, respectively. A zero here means that the candidate scored exactly zero against *every* validation example, not that its positive and negative scores cancelled. Exact-zero rates vary substantially across tasks and are included here as part of the observed attribution profile. These attribution profiles show that increasing the number of generated examples does not ensure a proportional increase in candidates with positive estimated effects for the validation set. The distributions depend on the task, generation prompts and settings, and generator model.

**Prompt-strategy analysis.** Figure 4.18 compares the attribution distributions of the synthesis prompt strategies within each task. The distributions show that no prompt strategy is consistently best across tasks. The easy paraphrase prompt has the highest mean attribution for Abercrombie and Oral Argument Question Purpose, the medium paraphrase prompt performs best for Function of Decision Section, and the A–C class-boundary prompt performs best for Insurance Policy Interpretation. Every strategy produces positive-, negative-, and zero-effect examples, in proportions that vary by task.

The mean summarizes each prompt strategy separately, but it does not show how its positive examples rank relative to those from other strategies. We therefore pool all examples with positive raw mean attribution, rank them by that score, and measure the fraction of each strategy's positive examples contained in the first $N$ positions. Figure 4.19 shows these curves. A curve above the uniform-ranking line means that the strategy has more relative utility compared to others, while a curve below means that its positive examples tend to appear later in the ranking.

The relative ordering again changes across tasks. The domain-coverage and easy paraphrase prompts are concentrated near the top for Abercrombie, whereas the A–C class-boundary prompt ranks highest for Insurance Policy Interpretation. The paraphrase prompts lead in Oral Argument Question Purpose and are especially dominant in Function of Decision Section. Thus, even after restricting the comparison to positive examples, no single prompt strategy is strongest across all tasks.

Besides the prompt-strategy analysis, averaging across validation examples hides how

each sample affects individual classes: a sample might have strongly positive attribution for one class while hurting predictions for others. We therefore further analyze attribution scores classified by both synthesis prompt strategy and the prediction classes of each task.

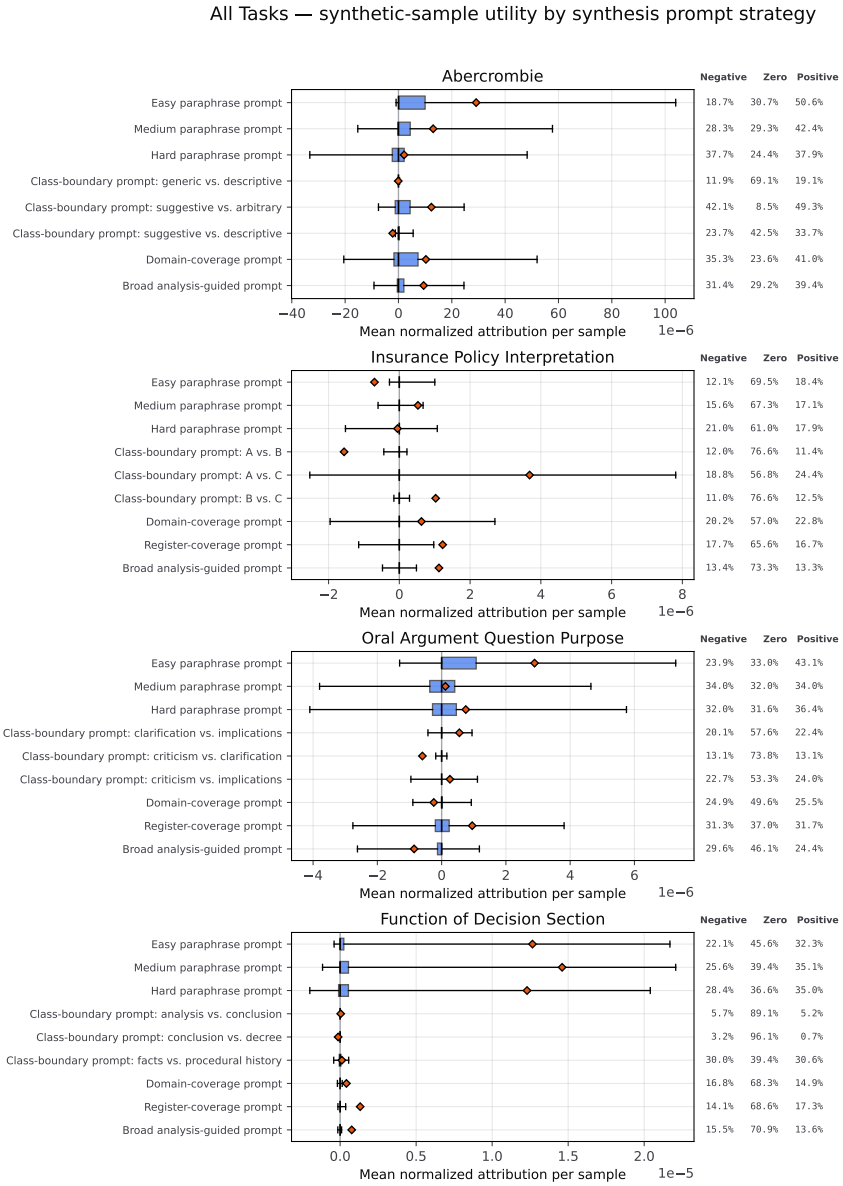



Figure 4.18: Distribution of average normalized TRAK scores for each synthesis prompt strategy. Boxes show the interquartile range, centre lines show the median, diamonds show the mean, and whiskers span the 10th to 90th percentiles. Annotations report the negative, exactly zero, and positive shares.

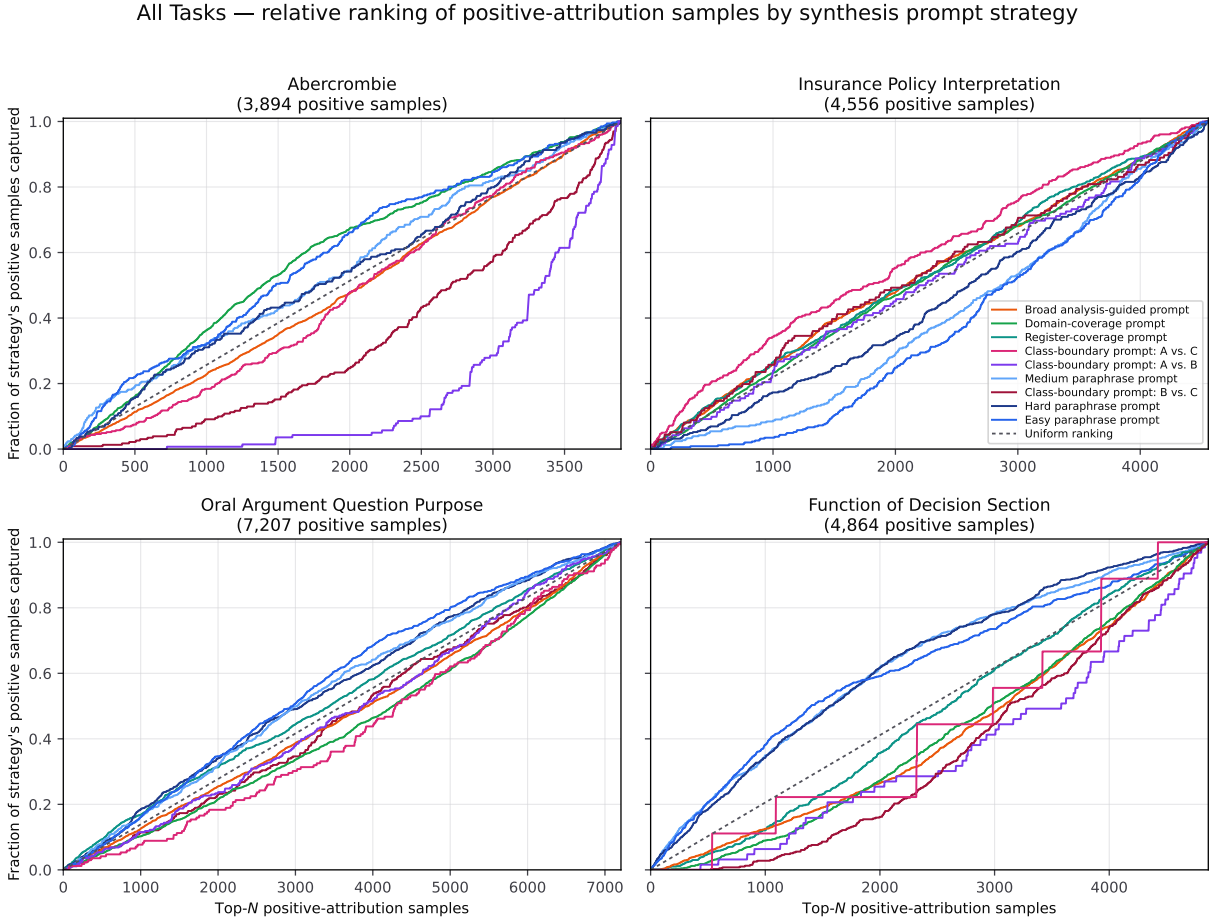



Figure 4.19: Relative ranking of positive-attribution examples by synthesis prompt strategy. Within each task, examples with positive raw mean attribution are pooled and ranked from highest to lowest. At each top-$N$ cutoff, each curve reports the fraction of that strategy's positive examples captured. The dotted line is the expected curve under a uniform ranking.

**Class-wise analysis.** To expose these differences, we group the validation examples by their true class and analyze every combination of prompt strategy and class. Positive support is the mean over all positively attributed examples from the strategy, while net signed support is the mean over all its examples, including negative and zero scores. The first measure shows the strength of the useful examples that could be retained, while the second shows the overall effect of the complete generated pool. Figures 4.20–4.23 present both views for the four tasks.

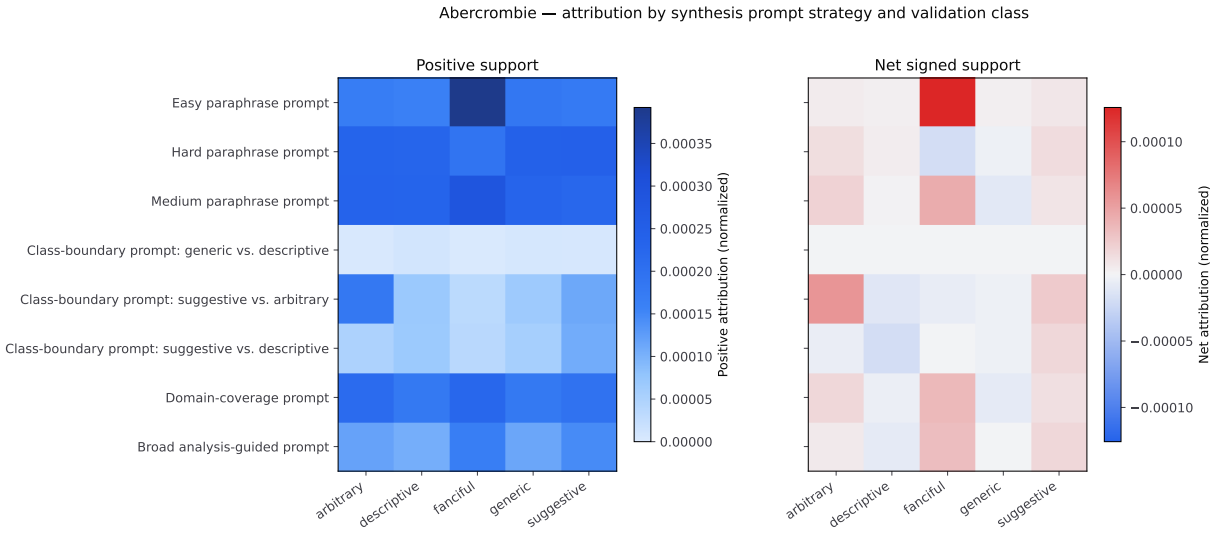



Figure 4.20: Attribution by prompt strategy and class for Abercrombie. The left panel shows mean positive support, while the right panel shows net signed support.

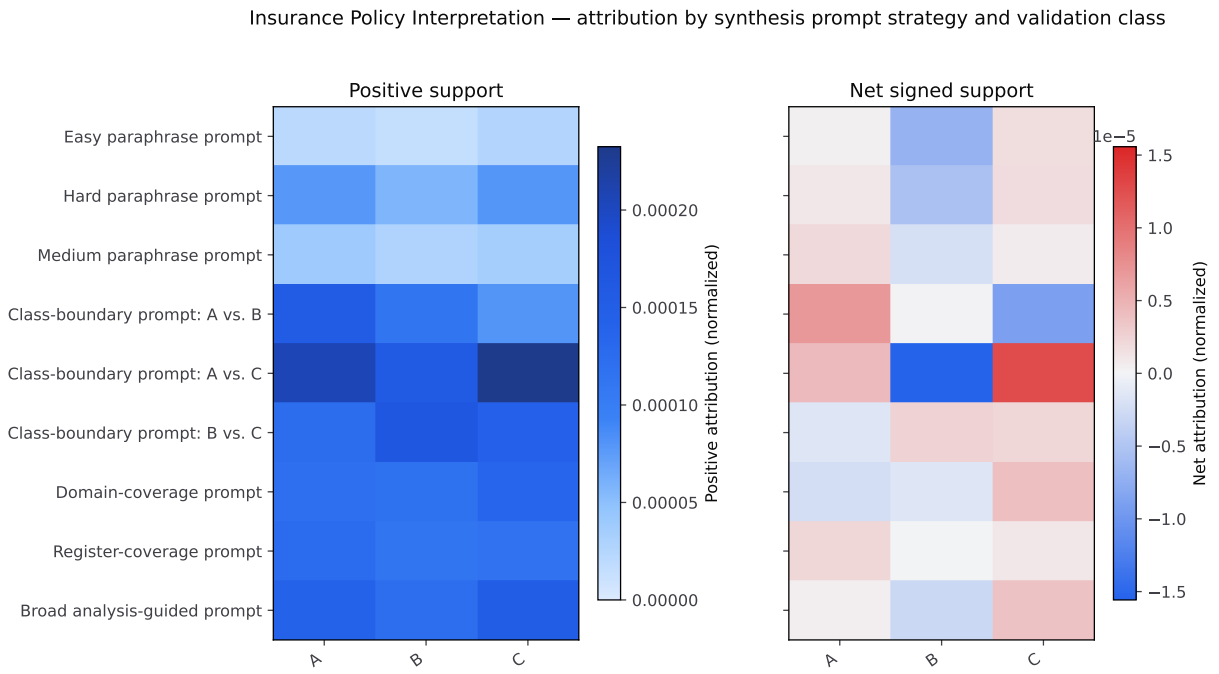



Figure 4.21: Attribution by prompt strategy and class for Insurance Policy Interpretation. The left panel shows mean positive support, while the right panel shows net signed support.

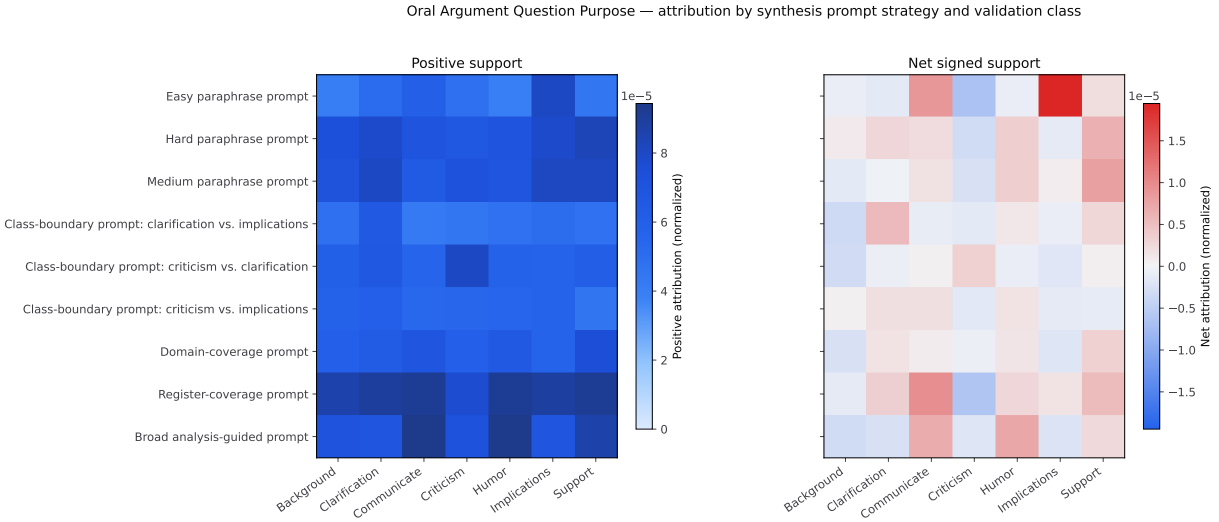



Figure 4.22: Attribution by prompt strategy and class for Oral Argument Question Purpose. The left panel shows mean positive support, while the right panel shows net signed support.

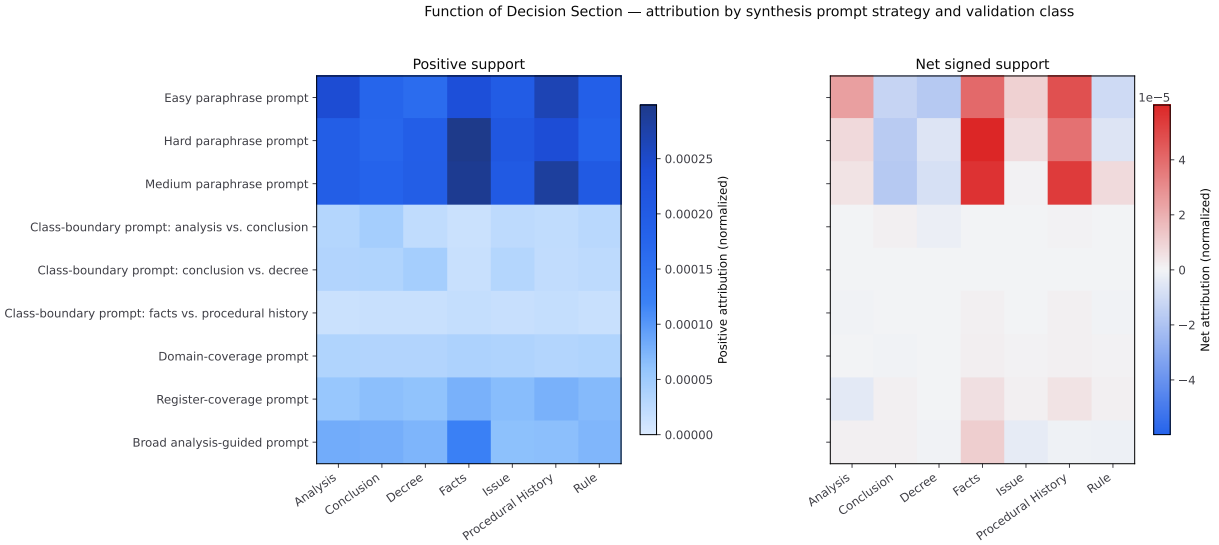



Figure 4.23: Attribution by prompt strategy and class for Function of Decision Section. The left panel shows mean positive support, while the right panel shows net signed support.

The positive-support heatmaps show that the strongest prompt strategy depends on the task. Paraphrase prompts provide the strongest positive examples for every class in Abercrombie and Function of Decision Section. Insurance Policy Interpretation shows the opposite pattern: the A–C class-boundary prompt leads for classes A and C, while the B–C class-boundary prompt leads for class B, and paraphrase prompts have the least positive effect among the strategies. Oral Argument Question Purpose benefits from a broader mixture, as all strategies contribute more equally compared to other tasks. Consistent with the previous plots, the signed heatmaps show why these positive examples must still be selected carefully, as a strategy can provide strong positive support for a class while having a weak or negative average when all its outputs are included. Synthesis prompt strategies are therefore complementary rather than interchangeable: their value depends on the task and class, and their outputs still require sample-level filtering.

Figure 4.24 shows that positive attribution is generally more concentrated on the diagonal, particularly for Abercrombie, Oral Argument Question Purpose, and Function of Decision Section, while most off-diagonal pairs are balanced or slightly negative and Insurance Policy Interpretation shows weaker separation. Thus, a synthetic example is usually most useful for validation examples of its own class, but its effect on other classes can be neutral or harmful, which motivates averaging the scores across all validation samples to prevent selecting samples that are useful on their own class, but hurts other classes.

Overall, attribution gives a useful profile of synthetic-data quality. Simple paraphrasing often has the best average usefulness, while multi-step generation can provide stronger support for particular classes. Average score measures how consistently a method produces useful data, while class-wise support shows where that usefulness occurs. Together, these results support the design used in this thesis: generate a broad pool through several methods, then use attribution to keep the candidates that are most useful for the target model.

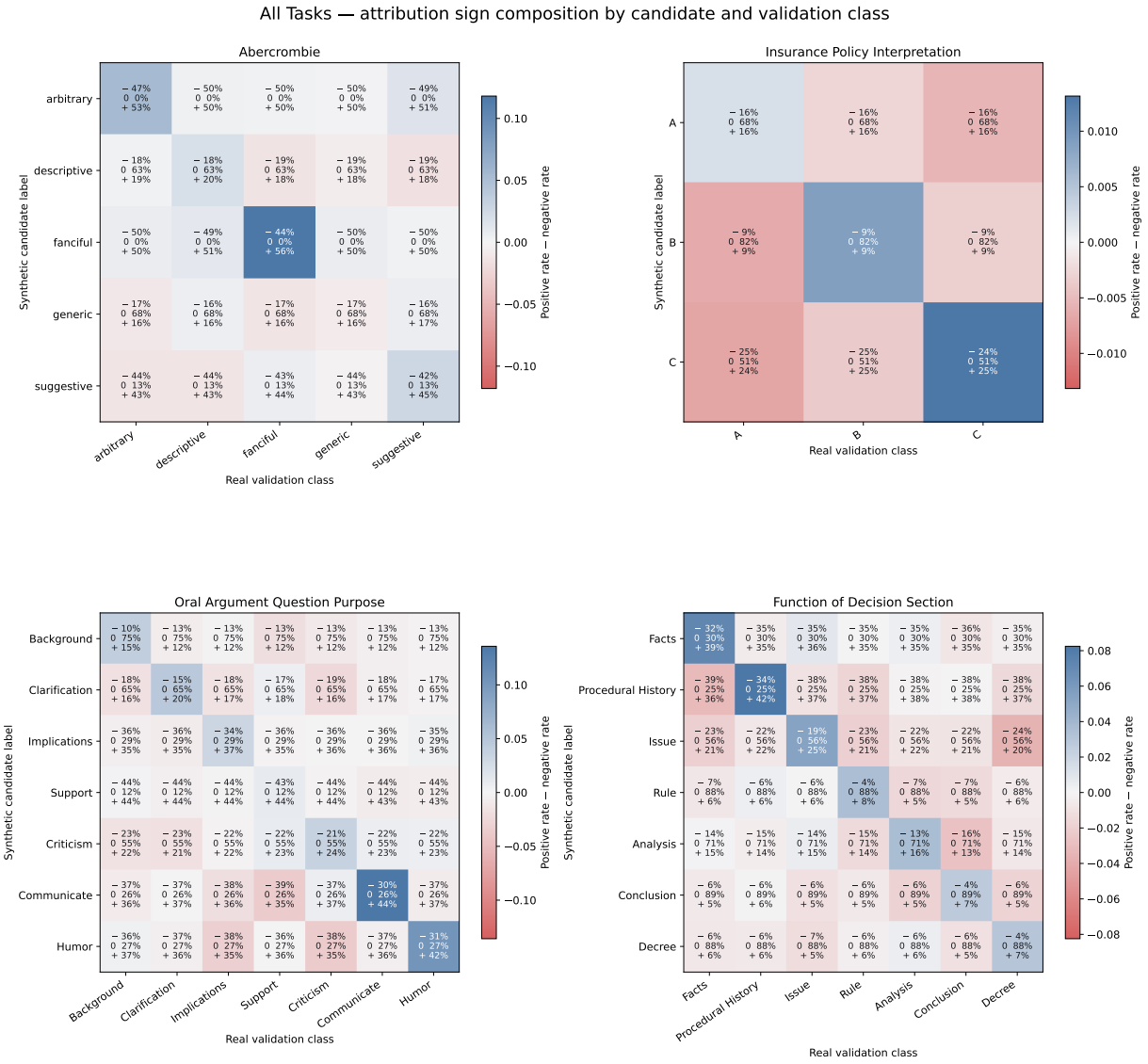



Figure 4.24: Class–class attribution sign composition across the four tasks. Rows denote the labels of synthetic candidates and columns denote the labels of real validation examples. Cell annotations report the negative, exactly zero, and positive attribution rates, while colour represents the positive rate minus the negative rate.

# Chapter 5

# Discussion

## 5.1 Student Models Outperforming Their Teachers

A central finding of the experiments is that the performance of the teacher models under direct prompting does not necessarily constitute an upper bound on the performance of a student trained from synthetic data. Across the four tasks and three teacher models there are 12 teacher baselines per prompting regime, and how many the best student surpasses depends on how the subset size $N$ is chosen. Selecting $N$ on the validation set, which uses no test information, surpasses 7 of the 12 zero-shot baselines, 5 of the 12 few-shot, and 1 of the 12 many-shot. Selecting $N$ on the test set raises this to 9, 7, and 2, and attributing against the test set as well – the oracle row – reaches 10, 8, and 5. The honest configuration therefore retains most of the zero-shot and few-shot advantage, while the margin that separates it from the oracle is concentrated in the many-shot comparisons. The pattern is therefore strongest when the comparison is against teachers operating with limited task-specific context, and becomes weaker as increasingly large amounts of real task data are supplied to the teacher in-context. Nevertheless, the results demonstrate that even a 0.8B–4B parameter student can, after specialization through synthetic-data fine-tuning, match or outperform models that are orders of magnitude larger on the same classification task.

This result highlights an important distinction between a teacher's ability to **solve a task directly** and its **ability to provide the knowledge required to construct a specialized training distribution**. The teacher models are designed to perform well across a very broad range of tasks, and are used here through prompting without changing their parameters for the specific LegalBench task. The student, in contrast, is explic-

itly optimized for a single classification problem using thousands of task-specific synthetic examples selected according to their estimated usefulness. A teacher may therefore be imperfect at solving a specialized task directly while still possessing enough general knowledge to generate examples that expose the relevant concepts, class boundaries, domains, and linguistic variations needed to train a specialist. Through synthesis, selection, and fine-tuning, this broad knowledge is converted into a narrow task-specific capability. The resulting student does not need to reproduce the teacher's general abilities; it only needs to become highly effective on the target task, which helps explain how a much smaller specialized model can occasionally outperform the general model that generated its training data.

## 5.2 High-Recall, Low-Precision Nature of Weakly-Guided Synthetic Data and the Motivation for Attribution Analysis

The results support viewing synthetic data generation as a high-recall, low-precision candidate-generation process, rather than as a mechanism for directly producing a final training set. In the low-resource setting considered in this thesis, the teacher has access to only a limited number of real examples, so synthesis is deliberately designed to expand beyond the observed data through multiple generation strategies, including paraphrase prompts, class-boundary prompts, domain-coverage prompts, register-coverage prompts, and broad analysis-guided prompts. This increases coverage of potentially useful task regions, but it also introduces uncertainty: generated examples may be redundant, ambiguous, weakly aligned with the target distribution, incorrectly labelled, or simply provide little useful learning signal for the student. The attribution analysis in Chapter 4 makes this heterogeneity visible. Depending on the task, only 17.5%–39.4% of the synthetic pool has positive mean attribution, while substantial portions receive exactly zero or negative mean attribution. Thus, unlike the traditional view of training data, generating a larger pool does not imply that a proportionally larger amount of useful supervision has been obtained.

This heterogeneity also occurs within prompt strategies, rather than only between them. The prompt-strategy and class-wise analyses show that no synthesis strategy is consistently strongest across tasks, and that even strategies with high average usefulness still produce mixtures of positive, zero-effect, and negative-attribution examples. Moreover, a strategy can provide strong support for one validation class while offering weak or negative support for another. Consequently, selecting data only at the level of a prompt strategy

is insufficient; usefulness must also be assessed at the level of individual examples. This motivates the role of attribution in LegalSynth: synthesis is used first to maximize coverage and construct a broad candidate pool, after which TRAK provides a model- and target-dependent signal for identifying which candidates are estimated to be most useful for the student. In this way, synthesis and attribution serve complementary roles: the former expands the space of possible supervision, while the latter recovers precision by filtering that space into a more effective training subset.

## 5.3 Cost, Efficiency, and Privacy

Beyond predictive performance, synthesis-based distillation changes the cost structure of deploying a model for a specialized task. Using a large black-box LLM directly incurs a recurring inference cost for every new input, whereas LegalSynth concentrates most of its expense into a one-time upfront process: teacher calls are used to construct the synthetic candidate pool, GPU computation is used for TRAK attribution and selection, and the resulting data are used to fine-tune a small student model. Once this process is complete, inference no longer requires access to the teacher. The resulting 0.8B–4B parameter students are small enough for cheap local deployment, including on consumer hardware such as laptops or even smartphones. The initial synthesis and attribution cost can therefore be amortized over subsequent predictions, making the approach increasingly attractive as deployment volume grows.

Privacy provides a second motivation that is particularly important in the legal domain. Many practical legal-classification workloads involve proprietary or confidential documents that cannot simply be sent to a third-party black-box LLM for every prediction. Local deployment is therefore desirable, but data scarcity creates a second problem: an organization may not possess enough labelled real data to train a high-performing specialized model directly, while conventional distillation is also limited when there are too few task-relevant inputs over which the teacher can provide supervision. LegalSynth is intended to bridge this gap by using the limited available task information to construct a much larger task-specific synthetic training pool and then filtering it before fine-tuning a compact local student. After this one-time specialization stage, deployment-time proprietary data can be processed entirely by the local student rather than being transmitted to an external model provider.

## 5.4 Implications for Legal NLP

The findings of this thesis suggest that building effective legal NLP systems does not necessarily require either a large proprietary model at inference time or a large labelled legal dataset. Even when only a small amount of real task data is available, a general-purpose LLM can be used to expand that limited information into a broader synthetic candidate pool, while attribution can identify the examples that are most useful for specializing a small student model. This is particularly relevant for legal classification, where datasets are often limited, class distinctions can be highly task-specific, and deployment may involve proprietary or confidential documents that favor local inference. Rather than treating general-purpose LLMs only as end-user legal systems, these results suggest another useful role for them: as temporary knowledge sources for constructing smaller, task-specialized models that can be deployed locally and efficiently.

# Chapter 6

# Conclusion

## 6.1 Summary of Findings

This thesis investigated language-model distillation for legal text classification under data scarcity, where only a small set of real examples is available. To address this setting, we introduced LegalSynth, a framework that combines task-aware synthetic data generation, TRAK-based data attribution, and supervised fine-tuning of small language models. Since synthetic data generation with limited signals has high-recall and low-precision, the framework first produces a broad candidate pool using several complementary synthesis strategies and then uses attribution with respect to held-out real examples to identify the candidates that provide the most useful training signals. The selected data are subsequently used to fine-tune Qwen3.5 students ranging from 0.8B to 4B parameters, and compared them with much larger, frontier language models.

Across the four LegalBench classification tasks, the experiments show that the quality of the synthetic training set matters substantially in addition to its size. Fine-tuning on attribution-selected subsets generally outperformed training on the complete synthetic pool, while low-ranked subsets performed substantially worse, indicating that TRAK captures a meaningful directional signal for data selection. The resulting specialized students were also competitive with much larger general-purpose models. Across the twelve teacher–task combinations per prompting regime, the best student with validation-set attribution and $N$ chosen on validation surpassed seven zero-shot teacher baselines, five few-shot baselines, and one many-shot baseline; choosing the subset size on test instead raised this to nine, seven, and two, and the test-attribution oracle to ten, eight, and five. These gains were nevertheless task-dependent: on Function of Decision Section, neither larger synthetic

pools nor attribution-based selection were sufficient to close the gap to most results of the stronger teacher configurations, illustrating that data selection cannot compensate indefinitely for limitations in the underlying synthetic pool, student capability, or the ambiguity of the task itself.

The attribution analysis further showed that synthetic data have high variance in terms of quality. Within the same candidate pool, examples with positive, negligible, and negative estimated effects coexist, and no single synthesis strategy is consistently strongest across tasks or validation classes. Taken together, these findings support the central hypothesis that **Synthesis expands the space of available task-specific supervision, while attribution recovers precision by identifying which generated examples are most valuable for specialization**. Their combination can therefore enable us to distill a teacher's capabilities into a small student model for a specialized legal classification task.

## 6.2 Limitations and Future Work

This study has several limitations that also motivate natural directions for future work. First, the number of synthetic examples generated for each task, as well as the generation budget assigned to each synthesis method, was chosen manually rather than optimized. The attribution analyses in Chapter 4.2.2 show that the usefulness of different synthesis methods varies substantially across tasks and validation classes, suggesting that attribution signals could be used more actively during generation. A future extension could use TRAK scores to dynamically reallocate generation budget toward methods that produce more useful examples, or even incorporate attribution feedback directly into the synthesis loop. More generally, estimating how many synthetic examples are actually needed for a given task remains an open problem. Instead of fixing the pool size in advance, future work could investigate batch-by-batch generation and stopping criteria based on the marginal utility of newly generated samples.

Second, TRAK attribution was computed only with the Qwen3.5-0.8B reference model, and the resulting scores were reused for selecting data for the 2B and 4B students. While this substantially reduces attribution cost, it implicitly assumes that examples useful for the smallest student are similarly useful for larger models. Future work could examine the consistency of attribution rankings across model capacities and determine whether student-specific attribution provides additional gains. Finally, the experiments are limited to four LegalBench multi-way classification tasks, a single synthesis-model configuration, and one student-model family. Extending the evaluation to a broader range of legal tasks,

domains, teacher models, and student architectures would help establish how well the observed benefits of synthesis and attribution generalize beyond the setting studied in this thesis.

# References


[1] Farid Ariai, Joel Mackenzie, and Gianluca Demartini. Natural language processing for the legal domain: A survey of tasks, datasets, models, and challenges. *ACM Computing Surveys*, 58(6):163:1–163:37, 2026.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] Dan Busbridge, Amitis Shidani, Floris Weers, Jason Ramapuram, Etai Littwin, and Russ Webb. Distillation scaling laws. *arXiv preprint arXiv:2502.08606*, 2025.

[4] Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. Legal-bert: The muppets straight out of law school. In *Findings of the association for computational linguistics: EMNLP 2020*, pages 2898–2904, 2020.

[5] Ilias Chalkidis, Abhik Jana, Dirk Hartung, Michael Bommarito, Ion Androutsopoulos, Daniel Katz, and Nikolaos Aletras. Lexglue: A benchmark dataset for legal language understanding in english. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4310–4330, 2022.

[6] Pierre Colombo, Telmo Pessoa Pires, Malik Boudiaf, Dominic Culver, Rui Melo, Caio Corro, Andre FT Martins, Fabrizio Esposito, Vera Lúcia Raposo, Sofia Morgado, et al. Saullm-7b: A pioneering large language model for law. *arXiv preprint arXiv:2403.03883*, 2024.

[7] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[9] Zhiting Fan, Ruizhe Chen, Tianxiang Hu, Ru Peng, Zenan Huang, Haokai Xu, Yixin Chen, Jian Wu, Junbo Zhao, and Zuozhu Liu. Optimsyn: Influence-guided rubrics optimization for synthetic data generation. *arXiv preprint arXiv:2604.00536*, 2026.

[10] Amirata Ghorbani and James Zou. Data shapley: Equitable valuation of data for machine learning. In *International conference on machine learning*, pages 2242–2251. PMLR, 2019.

[11] John Gibbons and M Teresa Turell. *Dimensions of forensic linguistics*, volume 5. John Benjamins Publishing, 2008.

[12] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International journal of computer vision*, 129(6):1789–1819, 2021.

[13] Neel Guha, Julian Nyarko, Daniel E. Ho, Christopher Ré, et al. Legalbench: A collaboratively built benchmark for measuring legal reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2023.

[14] Dan Hendrycks, Collin Burns, Anya Chen, and Spencer Ball. Cuad: An expert-annotated nlp dataset for legal contract review. *arXiv preprint arXiv:2103.06268*, 2021.

[15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[16] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[17] Yue Huang, Siyuan Wu, Chujie Gao, Dongping Chen, Qihui Zhang, Yao Wan, Tianyi Zhou, Jianfeng Gao, Chaowei Xiao, Lichao Sun, and Xiangliang Zhang. DataGen: Unified Synthetic Dataset Generation via Large Language Models, November 2025. arXiv:2406.18966 [cs].

[18] Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Understanding predictions with data and data with predictions. In *International Conference on Machine Learning*, pages 9525–9587. PMLR, 2022.

[19] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. Pmlr, 2017.

[20] Xiaochuan Li, Zichun Yu, and Chenyan Xiong. Montessori-instruct: Generate influential training data tailored for student learning. In *International Conference on Learning Representations*, volume 2025, pages 74809–74834, 2025.

[21] Joel Niklaus, Veton Matoshi, Pooja Rani, Andrea Galassi, Matthias Stürmer, and Ilias Chalkidis. Lextreme: A multi-lingual and multi-task benchmark for the legal domain. In *Findings of the association for computational linguistics: EMNLP 2023*, pages 3016–3054, 2023.

[22] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, 2022.

[23] Sung Min Park, Kristian Georgiev, Andrew Ilyas, Guillaume Leclerc, and Aleksander Madry. TRAK: Attributing model behavior at scale. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[24] Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. *Advances in neural information processing systems*, 33:19920–19930, 2020.

[25] Marco Siino, Mariana Falco, Daniele Croce, and Paolo Rosso. Exploring llms applications in law: A literature review on current legal nlp approaches. *Ieee Access*, 13:18253–18276, 2025.

[26] Qwen Team. Qwen3. 5-omni technical report. *arXiv preprint arXiv:2604.15804*, 2026.

[27] Tristan Thrush, Sung Min Park, Herman Brunborg, Luke Bailey, Marcel Roed, Neil Band, Christopher Potts, and Tatsunori Hashimoto. Synthetic data for any differentiable target. *arXiv preprint arXiv:2604.08423*, 2026.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[29] Steven Wang, Antoine Scardigli, Leonard Tang, Wei Chen, Dmitry Levkin, Anya Chen, Spencer Ball, Thomas Woodside, Oliver Zhang, and Dan Hendrycks. Maud: An expert-annotated legal nlp dataset for merger agreement understanding. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 16369–16382, 2023.

[30] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 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. Association for Computational Linguistics, 2023.

[31] Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. Less: Selecting influential data for targeted instruction tuning. *arXiv preprint arXiv:2402.04333*, 2024.

[32] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. Wizardlm: Empowering large language models to follow complex instructions. *arXiv e-prints*, pages arXiv–2304, 2023.

[33] Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*, 2024.

[34] Zitong Yang, Neil Band, Shuangping Li, Emmanuel Candes, and Tatsunori Hashimoto. Synthetic continued pretraining. In *International Conference on Learning Representations*, volume 2025, pages 44379–44421, 2025.

[35] Jiacheng Ye, Jiahui Gao, Qintong Li, Hang Xu, Jiangtao Feng, Zhiyong Wu, Tao Yu, and Lingpeng Kong. Zerogen: Efficient zero-shot learning via dataset generation. In *Proceedings of the 2022 conference on empirical methods in natural language processing*, pages 11653–11669, 2022.

[36] Jiacheng Ye, Jiahui Gao, Zhiyong Wu, Jiangtao Feng, Tao Yu, and Lingpeng Kong. Progen: Progressive zero-shot dataset generation via in-context feedback. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 3671–3683, 2022.

# Appendix

# Appendix A

# Synthetic Data Generation Prompts

This appendix records the fixed prompt templates shared across tasks in the multi-step synthesis pipeline. Placeholders are shown in double braces and were filled at runtime. Repeated filled batch requests are omitted because they differ only in substituted fields such as class, domain, register, and sample count.

## A.1 Static Prompt Templates

These templates are task-independent. Task descriptions, labels, examples, and intermediate analysis results are inserted through the displayed placeholders.

### A.1.1 Paraphrasing

The same system and user templates were used at three difficulty levels. Unlike the other placeholders in the user template, `{{difficulty}}` and `{{guidance}}` do not vary per example: each is one of three fixed strings, selected by which tier the request belongs to. Table A.1 gives the literal value substituted for each placeholder at every tier. All three tiers preserve the material facts and the original label; only the degree of permitted rewriting changes.

### System

```
You are a data-augmentation assistant for this text-classification task.

Task: {{task_name}}
{{task_description}}

Labels:
{{labels_block}}

Paraphrase the original example while preserving its exact label and the
semantic feature that makes it belong to that label. Change wording, syntax,
clause order, tone, and register only to the degree requested. Do not add,
drop, or reverse material facts, entities, user choices, limitations, or
instructions. Every output must have the source label exactly.

Return only a valid JSON array. Each object must have `question` and `answer`.
```

### User

```
Original example (category: {{source_label}}):
"{{source_text}}"

Paraphrasing difficulty: {{difficulty}}
Guidance: {{guidance}}

Generate {{n_paraphrases}} distinct paraphrase(s). Preserve the exact category
`{{source_label}}`: {{label_definition}}

Return only a JSON array in which every object has `question` and
`answer`: "{{source_label}}".
```

Table A.1: Literal values substituted for `{{difficulty}}` and `{{guidance}}` in the paraphrase user prompt, by tier.

| Tier | `{{difficulty}}` | `{{guidance}}` |
|---|---|---|
| Easy | `EASY` | Make minor lexical substitutions and slight reordering. Keep sentence structure and all facts effectively unchanged. |
| Medium | `MEDIUM` | Substantially restructure syntax, voice, and wording while preserving all facts, entities, and the label-defining intent. |
| Hard | `HARD` | Use maximally different wording, structure, length, and register while preserving every material fact, entity, and the label-defining intent. |

### A.1.2 Multi-Step Phase 1: per-class analysis

#### System

```
{{task_context}}

You are a research analyst studying patterns in a small legal NLP dataset.
Your analysis will directly drive synthetic data generation prompts - be precise
and mechanistic, not vague. Focus on what a prompt-engineer can act on.

Important: since some classes on some tasks might have very few training samples (even just 1),
squeeze every inference you can from surface form, register, and rhetorical
structure. For single-sample classes, note explicitly what is uncertain.

The user message may include a CONTRAST pane of other-class examples. Those rows
are lookalikes only. Your JSON must describe the TARGET class alone. If a surface
pattern appears in both the target list and the contrast list, do not treat it as
a target-defining generation recipe - put it in forbidden_conditions or warn in
generation_guidance that it also occurs in another class.

Return a valid JSON object - no prose outside the JSON.
```

#### User

```
Analyze ALL training samples for the class "{{class_name}}" ({{n}} sample{{sample_plural}}).

=== TARGET CLASS: "{{class_name}}" ({{n}} samples) ===
sample_quality.idx is 0-based into THIS list only.

{{target_block}}
{{contrast_section}}

=== YOUR TASK ===
Produce a structured analysis as a JSON object with EXACTLY these fields.
Every field describes "{{class_name}}" only - do not write analyses of contrast classes.

{
  "class": "{{class_name}}",
  "n_samples": {{n}},

  "rhetorical_fingerprint": "<string> The core communicative intent - what the text MUST be doing for
  this example to be {{class_name}}. Be specific and mechanistic, not definitional.",

  "necessary_conditions": [
    "<condition that is always true for {{class_name}}> - describe what must be PRESENT",
    "..."
  ],

  "forbidden_conditions": [
    "<condition whose presence would rule OUT {{class_name}}> - what makes it NOT this class",
    "..."
  ],
```

```
  "surface_markers": {
    "lexical":   ["specific words, phrases, or terms that appear reliably"],
    "syntactic": ["sentence structures, question patterns, grammatical forms"],
    "pragmatic": ["rhetorical moves, discourse patterns, stance signals"]
  },

  "typical_length": "<short (1 sentence) / medium (2-3) / long (4+)> - with brief observation",

  "content_domains_in_data": [
    "<content area or topic visible in the training samples>"
  ],

  "content_domains_missing": [
    "<important content area NOT in training samples that would add diversity>"
  ],

  "sample_quality": [
    {
      "idx": <0-indexed into the TARGET CLASS list only>,
      "type": "prototypical | edge_case | ambiguous",
      "note": "<why this is a clear example OR why it is tricky>"
    }
  ],

  "key_discriminators": {
    {{key_discriminators}}
  },

  "generation_gaps": [
    "<type of example missing from training that would add value for training a classifier>"
  ],

  "generation_guidance": "<2-3 actionable sentences telling a prompt engineer how to reliably generate
  new {{class_name}} examples that are clearly this class and not another>"
}

Be concrete. Use excerpts from the samples where helpful. If n_samples is 1, be explicit
about what is uncertain vs. what can be confidently inferred.
```

### A.1.3 Multi-Step Phase 2a: class-boundary analysis

#### System

```
{{task_context}}

You are a classification boundary analyst. Your job is to identify which pairs of
classes a model will most often confuse, and to surface the single most reliable
discriminator for each pair.

Be mechanistic and precise. Focus on decision boundaries, not general descriptions.
Return a valid JSON object - no prose outside the JSON.

{{conflict_rule}}
```

### User

```
You have per-class analyses for all {{n_classes}} classes. Identify which pairs
a classifier will most often confuse and why.

=== PER-CLASS ANALYSES ===
{{analyses_json}}

=== YOUR TASK ===
Return a JSON object with EXACTLY this field:

{
  "confusion_pairs": [
    {
      "pair": ["ClassA", "ClassB"],
      "severity": "high | medium | low",
      "core_discriminator": "<THE single most reliable feature that separates ClassA from ClassB - must
      be mechanistic and testable, not vague>",
      "why_confused": "<the specific shared surface features that cause the confusion>",
      "contrastive_generation_strategy": "<concrete instruction for generating examples that
      unambiguously demonstrate the distinction - what makes ClassA examples NOT ClassB>"
    }
  ]
}

Rules:
  - List ALL plausible pairs (even low-severity ones) - rank by severity, highest first
  - core_discriminator must be a single sentence, not a list
  - Focus on the key_discriminators and forbidden_conditions from the per-class analyses

{{conflict_rule}}
```

## A.1.4 Multi-Step Phase 2b: content-domain coverage

### System

```
{{task_context}}

You are a dataset coverage analyst. Your job is to identify which content/topic areas
are well-represented in the training data and which are absent, so synthetic data
can fill the gaps.

Return a valid JSON object - no prose outside the JSON.
```

### User

```
Below are the content domain observations from per-class analyses.
Aggregate and identify the highest-value gaps for synthetic data generation.

=== PER-CLASS DOMAIN DATA ===
{{domain_json}}

=== YOUR TASK ===
Return a JSON object with EXACTLY this field:

{
  "legal_domain_coverage": {
    "well_covered": ["<topic areas appearing in multiple class analyses>"],
    "underrepresented": ["<topic areas mentioned in only 1 class or few examples>"],
    "high_value_gaps": [
      "<topic area that is common in real examples of this task but missing from ALL classes - list 6-10
      concrete areas ordered by importance>"
    ],
    "domain_class_notes": "<1-2 sentences on whether domain gaps affect some classes more than others>"
  }
}

Rules:
  - Infer what content/topic areas are relevant to this task from the task context above
  - high_value_gaps must name SPECIFIC topic areas, not vague broad categories
  - Consider what real-world variations of this task would naturally cover
  - Order high_value_gaps by expected frequency and value for training diversity
```

## A.1.5 Multi-Step Phase 2c: rhetorical-register analysis

### System

```
{{task_context}}

You are a rhetorical style analyst. Your job is to identify distinct RHETORICAL REGISTERS
- different ways of structuring and delivering a text example - present in the training
data, and determine which are underrepresented.

CRITICAL DISTINCTION: A rhetorical register governs HOW an example is expressed (sentence
structure, tone, stylistic form). It does NOT determine the class label (the WHY - the
purpose or intent). The same class can appear in many different registers, and different
classes can share the same register.

This distinction is what makes register_sweep valuable for data generation: it increases
surface diversity without altering class labels.

Return a valid JSON object - no prose outside the JSON.
```

## User

```
Below are the surface marker analyses from all {{n_classes}} classes.
Derive a rhetorical register taxonomy from this data and analyze its distribution.

=== SURFACE MARKER DATA (from Phase 1) ===
{{style_json}}

=== YOUR TASK ===
Step 1: Based on the surface_markers data above, identify 5-8 distinct rhetorical
registers present in or relevant to this dataset. Each register describes a distinct
way of structuring/delivering an example (e.g. terse/direct, elaborate/discursive,
hypothetical-driven, citation-heavy, socratic/leading, reductio ad absurdum, etc.).
Derive these from the actual syntactic and pragmatic patterns in the data.

Step 2: For each derived register, analyze its coverage across the training data.

Return a JSON object with EXACTLY this field:

{
  "rhetorical_registers": {
    "taxonomy_validation": {
      "<register_name>": {
        "description": "<what characterizes this register - sentence structure, tone, form>",
        "confirmed_in_data": true | false,
        "evidence": "<which samples or surface markers confirm or suggest this register>",
        "classes_where_dominant": ["<class>"],
        "classes_where_rare": ["<class>"],
        "frequency_in_training": "dominant | present | rare | absent"
      }
    },

    "underrepresented_registers": [
      "<register names that are rare or absent - ordered by value for diversity>"
    ],

    "register_class_affinity": {
      "<register_name>": "<which classes it naturally maps to AND importantly which classes it can also
      serve - always note that registers are HOW not WHY>"
    },

    "key_observations": [
      "<insight about register distribution that would inform generation>"
    ],

    "generation_constraint": "The register governs HOW an example is expressed (structure, style, tone),
    NOT the class label. A generated example MUST serve its target class purpose regardless of register.
    The register only changes the surface form."
  }
}

Rules:
  - Use surface_markers.syntactic and surface_markers.pragmatic as primary evidence
  - underrepresented_registers should be ordered: rarest/most valuable first
  - For each register, note at least one class where it would be UNUSUAL but valid
    (this is what makes register_sweep hard and valuable for training)
  - Be specific: name concrete structural patterns, not abstract style categories
```

### A.1.6 Multi-Step Phase 3: generation-prompt design

#### System

```
You are a prompt engineer. You will be given a DESIGN BRIEF with pre-specified
slots for synthetic data generation prompts. Each slot has already been assigned:
  - a strategy type (contrastive | domain_sweep | register_sweep | standard)
  - target classes
  - exact findings from prior analysis phases (discriminators, surface markers,
    fingerprints, register taxonomy) that MUST be embedded verbatim
  - training_examples: READ these so your guidance fits the real paragraphs the
    generator will see. Do not quote, transcribe, or summarise them in the
    system_prompt - Python appends them verbatim after your text

Every finished system_prompt will have this exact opener prepended by the caller:

--- BEGIN AUTO-PREPENDED OPENER ---
{{task_context}}
--- END AUTO-PREPENDED OPENER ---

The caller then inserts a "YOUR ASSIGNMENT:" paragraph that states what this slot
is for (which strategy, which classes, and why). Your text follows that and must
read as its natural continuation:
  - do NOT begin with "You are generating synthetic training data…" or any
    other self-introduction; the opener already said it
  - do NOT restate the task name, the task description, or the class definitions
  - do NOT write your own "YOUR ASSIGNMENT" paragraph or otherwise explain the
    slot's purpose; that paragraph is added for you
  - begin immediately with an ALL-CAPS section header for the slot's own content
    (e.g. "CORE DISCRIMINATOR:", "UNDERREPRESENTED DOMAINS:", "CLASS GUIDANCE:")

The finished prompt therefore reads: task opener → YOUR ASSIGNMENT → your
guidance → REFERENCE EXAMPLES. You write only the guidance.

Your ONLY job is to write the slot-specific system_prompt and user_prompt_template.
Do NOT change slot assignments, strategies, or target classes.
Do NOT invent findings - use only what is provided in each slot's brief.
Every system_prompt must be self-contained (no references to "the analysis above").
Every user_prompt_template must end with the exact JSON output instruction.

Return a valid JSON object - no prose outside the JSON.
```

#### User

```
Here is your design brief. It contains {{n_slots}} pre-specified slots.
For each slot, write the slot-specific system_prompt and user_prompt_template.

official_task_definition in the brief is the opener that will be prepended to
every system_prompt. Do not restate or paraphrase it. Write only the
slot-specific guidance (discriminators, fingerprints, domains, registers),
starting with an ALL-CAPS section header so it flows on from the opener.

Read each slot's training_examples so the guidance you write is consistent with
```

```
those real paragraphs. Do not quote or transcribe them - Python appends the
full REFERENCE EXAMPLES section after your text.

=== DESIGN BRIEF ===
{{brief_json}}

=== OUTPUT FORMAT ===
Return a JSON object:
{
  "prompts": [
    {
      "slot_id": <integer matching the slot>,
      "name": "<use the slot's name_hint exactly>",
      "strategy": "<copy from slot - do not change>",
      "target_classes": ["<copy from slot - do not change>"],
      "priority": <copy slot_id>,

      "rationale": "<1-2 sentences: which specific finding from the brief this prompt addresses>",

      "recommended_n_samples": <copy the integer from the slot's recommended_n_samples field - do NOT
      change it>,
      "recommended_temperature": <copy from slot's recommended_temperature - do NOT change it>,
      "recommended_batch_size": 15,

      "system_prompt": "<WRITE THIS: slot-specific guidance ONLY. It is inserted after
        the task opener and an auto-written 'YOUR ASSIGNMENT:' paragraph, so start
        with an ALL-CAPS section header - no self-introduction sentence, no
        restating the task name, description, or class definitions, and no
        paragraph explaining what the slot is for (all of that is added in Python).
        Embed EVERY listed field from the slot verbatim - never summarise a field
        away, and never drop a field because it feels redundant.
        - contrastive: paste core_discriminator and why_confused word-for-word; then,
          for EACH class in the pair, a labelled sub-section carrying all of:
          necessary_conditions, forbidden_conditions, surface_markers, and a
          generation strategy that leads with the slot's generation_strategy
          followed by that class's generation_guidance
        - domain_sweep: list ALL domains from underrepresented_domains explicitly;
          then, for EACH target class, its rhetorical_fingerprint, surface_markers,
          and both necessary_conditions and forbidden_conditions from
          class_conditions, so the label stays correct in every domain
        - register_sweep: describe each underrepresented_register from register_taxonomy
          in detail; paste the critical_constraint verbatim; then, for EACH target
          class, its rhetorical_fingerprint, surface_markers, and both
          necessary_conditions and forbidden_conditions from class_conditions, so the
          label stays correct while only the delivery style changes
        - standard: for EACH target class, a labelled sub-section carrying ALL of:
          rhetorical_fingerprint, necessary_conditions, forbidden_conditions,
          surface_markers, generation_guidance, and generation_gaps (list every gap
          explicitly - these name the coverage the run exists to add, so they must
          not be omitted)
        Read training_examples so this guidance fits those real paragraphs, but
        stop after the slot-specific guidance. Do not write REFERENCE EXAMPLES -
        Python appends the excerpts verbatim after your text.
        The system prompt must be actionable - a model reading it cold must generate
        clearly correct examples without needing any external context.>",

      "user_prompt_template": "<WRITE THIS: a user prompt with {target_class} and
```

```
        {n_samples} placeholders (single curly braces - these are substituted at runtime).
        For domain_sweep slots: also include a {domain} placeholder that names the topic area.
        For register_sweep slots: also include a {register} placeholder that names the style.
        It MUST ask for {n_samples} DISTINCT examples and MUST instruct the model to
        vary length, sentence structure, subject matter, and named entities across the
        batch so no two examples read alike.
        It MUST restate that the examples have to satisfy the necessary conditions and
        avoid the forbidden conditions for {target_class}.
        Must end with EXACTLY this line:
        'Return ONLY a JSON array where each object has \"question\" (string) and \"answer\":
        \"{target_class}\".'>",

      "generation_notes": "<optional: any special handling during Phase 4 generation -
        e.g. how to alternate target_classes for contrastive prompts;
        for domain_sweep: caller cycles {domain} through the underrepresented_domains list;
        for register_sweep: caller cycles {register} through the underrepresented_registers list>"
    }
  ]
}

=== CRITICAL RULES ===
  1. Do not alter slot_id, strategy, or target_classes - these are fixed
  2. Copy recommended_n_samples, recommended_temperature, and recommended_batch_size
     verbatim from the slot - these numbers are set by the budget and must not be changed
  3. System prompts must embed findings from the brief verbatim, not paraphrase generically
  4. For domain_sweep: list ALL entries from underrepresented_domains explicitly
  5. For register_sweep: describe every register in underrepresented_registers in full;
     paste the critical_constraint field verbatim so the model never conflates style with class
  6. For contrastive: quote core_discriminator and why_confused verbatim
  7. Every user_prompt_template MUST end with the exact JSON instruction above
  8. The system prompt must be self-contained. It is read by a generation model that
     can see NOTHING except this prompt: no training data, no analysis, no sample
     indices. Never write "as per the analysis", "Sample 0", "despite noise in the
     training data", or any other pointer to material it cannot see - state the rule
     as a direct positive/negative instruction instead
  9. Every system_prompt must start with an ALL-CAPS section header and must not
     re-introduce the task, restate the description, or relist the class definitions
 10. For standard slots: every target class needs its own sub-section, and every
     sub-section needs all six fields (rhetorical_fingerprint, necessary_conditions,
     forbidden_conditions, surface_markers, generation_guidance, generation_gaps).
     This is the largest slot in the run; an under-specified standard prompt wastes
     the majority of the budget
 11. The user_prompt_template may only refer to guidance that the system_prompt
     actually states. Never tell the generator to satisfy or avoid a category of
     condition that the system prompt does not spell out
 12. Read training_examples so your guidance matches the real paragraphs. Do not
     write a REFERENCE EXAMPLES section and do not quote or transcribe those
     paragraphs - Python appends them verbatim after your text.
 13. Do not write a "YOUR ASSIGNMENT" paragraph or any other framing of what the
     slot is for - Python inserts that between the opener and your guidance.
```

### A.1.7 Multi-Step Phase 4: generation assembly fragments

#### Assignment contrastive

```
YOUR ASSIGNMENT:
This batch is contrastive. It targets {{pair}} - two classes a classifier trained on the real data keeps
confusing.{{severity_sentence}} You will be asked for one of these two classes at a time. Every paragraph
you write must be an unmistakable instance of the class you are asked for, and must not be readable as
the other one. The sections below give the distinction that separates them, then the rules for each class
individually.
```

#### Assignment domain sweep

```
YOUR ASSIGNMENT:
This batch is a domain sweep. The real data clusters in a narrow range of subject matter, so you will
write examples for all {{n_classes}} classes across {{domains}} the corpus barely covers. Each request
names one class and one domain: the domain decides what the paragraph is about, and the class decides
what the paragraph does. Moving into a new domain must never change which class the paragraph belongs to.
```

#### Assignment register sweep

```
YOUR ASSIGNMENT:
This batch is a register sweep. The real data shows each class in only a few voices, so you will re-voice
all {{n_classes}} classes in {{registers}} the corpus rarely uses. Each request names one class and one
register: the register decides how the paragraph sounds, and the class decides what the paragraph does.
Changing the register must never change which class the paragraph belongs to.
```

#### Assignment standard

```
YOUR ASSIGNMENT:
This batch is the broad-coverage pass and supplies the bulk of the training corpus, so its range matters
as much as its accuracy. You will write examples for all {{n_classes}} classes, one class per request.
For each class the sections below give its form, the conditions it must and must not meet, and the
specific coverage gaps the real data leaves open - write into those gaps rather than repeating the most
obvious version of the class.
```

## Reference contrastive

```
REFERENCE EXAMPLES:
The excerpts below are real instances of {{left}} and {{right}}, grouped so you can see the distinction
in actual prose. Use them as form anchors: a new example must sit as clearly on its labelled side as
these do. Invent new facts, parties, and wording - do not recycle what you read here.
```

## Reference domain sweep

```
REFERENCE EXAMPLES:
The excerpts below show the correct function of each class. Keep that function when you generate, and
move the subject matter into the requested domain - the domain changes the topic, not the label. Invent
new facts, parties, and wording; do not recycle these excerpts.
```

## Reference register sweep

```
REFERENCE EXAMPLES:
The excerpts below show the correct function of each class. Keep that function when you generate, and
recast the delivery into the requested register - the register changes the style, not the label. Invent
new facts, parties, and wording; do not recycle these excerpts.
```

## Reference standard

```
REFERENCE EXAMPLES:
The excerpts below show the standard form of each class. A new example must be as unmistakably its
labelled class as these are, while remaining new in facts, parties, and wording. Cover the generation
gaps named above where they apply; do not recycle these excerpts.
```

## Fewshot closer contrastive

```
When you generate, stay on the labelled side of the distinction. {{jobs_sentence}} Do not blend the
groups.
```

## Fewshot closer other

```
When you generate, match the group that corresponds to the requested class. The new paragraph should be
as easy to label as the references, and should not borrow their facts or phrasing.
```

### User reference line

```
Use the {target_class} reference examples in the system prompt as form anchors; do not copy their facts
or wording.
```