FINDR: A Fast Influential Data Selector for NL2Code Pretraining

Xinliang Frederick Zhang and **Lu Wang**

Computer Science and Engineering, University of Michigan, Ann Arbor, MI

{xlfzhang,wangluxy}@umich.edu

Abstract

Pretraining on massive corpora has given rise to large language models (LLMs) with multi-task capabilities. However, real-world applications often require more specialized training, as is the case of NL2Code. We approach this specialization through the lens of *data selection*, i.e., identifying a subset of a large corpus that aligns with a desired target distribution—a challenge that remains under-explored within NL2Code. Existing methods are typically designed for selecting instruction-tuning data, and might not easily scale to large-scale code repositories; while methods for NL2Code do exist, they primarily rely on coarse heuristics——such as repo stars——for filtering. To bridge this gap, we propose FINDR, an efficient data selection method that extends logistic regression with feature-wise importance reweighting—marking it, to our knowledge, the first fine-grained solution to NL2Code pretraining. Our method uses hashed n-grams and *code-aware features* to capture code-specific patterns, and then apply informative priors to *reweight* feature importance when computing *influence scores*. Extensive experiments on NL2Python and NL2SQL, with two model families, show that FINDR consistently outperforms strong baselines in both execution accuracy and token efficiency. Notably, pretraining on only 2% of FINDR-selected data boosts Gemma by over 29% in both domains, even surpassing CodeGemma (pretrained on 300x more examples) by 10% in Python.[1]

1 Introduction

Large language models (LLMs), such as GPT-4 (OpenAI, 2023), LLaMA3 (AI@Meta, 2024), Gemma (Mesnard et al., 2024), and Mistral (Jiang et al., 2023) have demonstrated remarkable capabilities across a wide range of natural language (NL) tasks, often surpassing expert-engineered NLP systems. Their success can be largely attributed to training on massive corpora (Gao et al., 2021), which capture diverse linguistic patterns and world knowledge (Chowdhery et al., 2023). However, as these models are adapted to more specialized downstream tasks, a significant fraction of pretraining data can become extraneous and counterproductive to such target tasks. In NL2Code (translating natural language into code; Dehaerne et al., 2022; Xu and Zhu, 2022), the choice of relevant and quality training data is even more crucial, since irrelevant data could introduce noisy, misleading examples that hinder the model's code generation quality (Wang et al., 2023a; Zan et al., 2023). Thus, selecting a subset of code-specific data is essential for adapting a general-purpose LLM to NL2Code tasks. However, manually curated code datasets are time-consuming, error-prone, and limit both scale and diversity (Yu et al., 2018; Hendrycks et al., 2021). Therefore, there presents a pressing need for *automated data selection framework for NL2Code* to mitigate these drawbacks and dynamically tailor to users' needs.

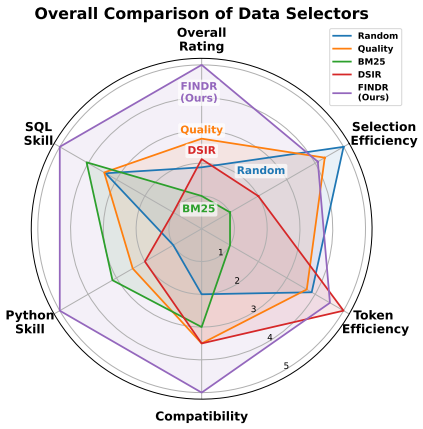

Figure 1: Overall comparison of our proposed FINDR with strong baseline data selectors. FINDR yields the best rating by balancing effectiveness and efficiency. See §A for detailed rubric behind this chart.

[1]Due to budget constraints, we study the pretraining of *small LLMs* ($< 3B$) in this work (see *Limitation* section).

A growing body of work highlights the importance of *data selection* for efficient pretraining or fine-tuning of LLMs (Feng et al., 2022; Chowdhery et al., 2023; Albalak et al., 2024). Two complementary perspectives are typically involved in this space: *data composition* (Soldaini et al., 2024)—deciding the ratio and types of data to include (e.g., NL vs. code)—and *coreset selection* (Phillips, 2016), where the goal is to identify the most important subset of the training data. In this work, we focus on *coreset selection*, where we seek a small subset of pretraining data that leads to performance on par with or better than full-dataset training. Recent advances, such as targeted instruction tuning (Xia et al., 2024), further highlight the impact of selecting the "right" data to cut computational costs and improve target skills.

Despite growing interest in data selection (Xie et al., 2023; Han et al., 2023; Xia et al., 2024), most existing methods remain computationally expensive because they were originally designed for instruction-tuning data (Wang et al., 2024), which might not scale easily to large pretraining corpora. While data selection techniques do exist for NL2Code (e.g., filtering by doc length or repo stars; Cao et al., 2023; Zan et al., 2023), they primarily rely on overly coarse heuristics that overlook subtle differences in examples. Consequently, *scalable and fine-grained data selection in NL2Code are left relatively under-explored*. To address these limitations, we propose an **efficient** data selector, FINDR (Fast INfluential Data Ranker), an extension of logistic regression with *feature-wise importance reweighting*. In addition, to capture code-specific constructs (e.g., vectorized function calls; Nasrabadi et al., 2023), we augment FINDR's hashed n-gram feature extractor with *code-aware feature*.[2] With these fast yet accurate designs, FINDR manages to strike a balance between data selection *effectiveness* and *efficiency*, making it particularly well-suited for large-scale settings.

We perform extensive experiments to evaluate FINDR on NL2Python (Lai et al., 2022) and NL2SQL (Li et al., 2024b) using two distinct small LLM (*sLLM*) families, demonstrating both its *effectiveness* and *efficiency*. Our results reveal that FINDR selects *on-target* data that consistently boosts base models across domains and generally outperforms strong baselines by non-trivial margins. For instance, on Python, FINDR improves the base model by 16% to 36%, and surpasses the SOTA selector, DSIR (Xie et al., 2023), by 10%. While BM25 remains a strong baseline (Xia et al., 2024), FINDR is substantially more efficient, processing 47 million Python files in 3.5 GPU hours compared to BM25's 760 CPU hours. Notably, training Gemma on 2% of data selected by FINDR outperforms CodeGemma by 10%, which consumes 300 times more examples. Moreover, the proposed FINDR exhibits robust generalization as verified in two model families including both general-domain and code-specific sLLMs.

We summarize major contributions as follows:

- We are the *first* to systematically study a suite of data selection methods for NL2Code continued pretraining, and perform comprehensive comparison among them (Figure 1).
- We propose FINDR, an *efficient* data selector to capture nuanced data influence at scale, which integrates code-aware features into hashed n-gram representations, and augments logistic regression with informative priors for feature-wise importance reweighting.
- Experiments on downstream NL2Python and NL2SQL tasks showcase that FINDR boosts base models substantially while efficiently identifying *on-target* data from large-scale pretraining corpus.
- We validate that FINDR robustly outperforms four baselines, including the SOTA selector (Xie et al., 2023), across two sLLMs (DeepSeek-Coder and Gemma) and two languages (Python and SQL) by large margins.

2 Related Work

2.1 Data Selection

Data selection has recently emerged as a fundamental research topic for LLMs (Coleman et al., 2020; Xia et al., 2020; Paul et al., 2021; Sachdeva et al., 2024). Two primary directions are *data composition* (Soldaini et al., 2024), which optimizes the mix of different data sources (e.g., natural language vs. code), and *coreset selection* (Phillips, 2016), which identifies a small, representative subset (the "coreset") that captures the dataset's essential features. While data composition can improve transparency and help inform decision-making process (Gebru et al., 2021; Elazar et al., 2024), ever higher compute spending has attracted increasing attention to coreset selection (Killamsetty et al., 2021; Xia

[2] Although we introduce *code feature* tailored to NL2Code, FINDR can be seamlessly adapted to large-scale unlabeled data by adjusting the custom feature space for other domains.

et al., 2023; Griffin et al., 2024). By focusing on the most *influential* data, coreset selection significantly cuts training costs without degrading performance, benefiting both pretraining (Xie et al., 2023; Han et al., 2023) and instruction tuning (Xia et al., 2024; Wang et al., 2024). In this work, we frame data selection as a **coreset selection** problem, developing **efficient** methods for identifying training subsets that match, or even exceed, the performance of training on manual selections (Feng et al., 2022) or even the full corpus.

Existing data selection approaches mainly fall into three broad categories. The first is *random sampling*, which, despite its simplicity and unbiased nature, results in uninformative selections that fail to represent the target domain (Devlin et al., 2019; Gururangan et al., 2020; Guo et al., 2022). The second relies on *surface-level matching* to make informed but efficient selections, including BM25 (Robertson and Zaragoza, 2009), DSIR (Xie et al., 2023), among others (Jiang and Zhai, 2007; Moore and Lewis, 2010; Du et al., 2022). DSIR, for instance, uses hashed n-gram features to represent documents and applies a Naïve Bayes model for data selection. Yet, they struggle with document-length variations: BM25 favors lengthy documents while DSIR selects overly short ones. Quality Classifier (Brown et al., 2020) is a more robust method, which leverages logistic regression and has become a standard for pretraining data selection (Gao et al., 2021; Chowdhery et al., 2023). However, this approach overlooks per-feature importance. FINDR is a *surface-level matching* method which, inspired by importance weighting for domain adaptation (Shimodaira, 2000; Sugiyama et al., 2007), introduces feature-wise importance reweighting to capture nuances among features.

The third category relies on fine-grained feature representations—such as embeddings (Chen et al., 2023; Wu et al., 2023; Xiao et al., 2024), gradients (Han et al., 2023; Xia et al., 2024), perplexities (Li et al., 2024c) or entropies (Kousar et al., 2025)—often combined with pairwise comparisons. While these methods can capture more subtleties in the data, they typically incur quadratic compute complexity, thus restricted to instruction data selection only (Wang et al., 2024). Recent approaches use ChatGPT (Zheng et al., 2023; Liu et al., 2024) to assess data relevance via prompting, but high API costs limit their scalability, especially when re-runs are needed. In contrast, FINDR offers an informed solution that is *tractable* at the scale needed for unlabeled pretraining data selection.

2.2 Natural Language to Code (NL2Code)

Translating natural language problem description into code (NL2Code) has attracted substantial attention for its potential to enhance developer productivity and democratize software development (Allamanis et al., 2018; Dehaerne et al., 2022; Zan et al., 2023). Early studies approached NL2Code through RNN (Iyer et al., 2016), LSTM (Eriguchi et al., 2016) and CodeBERT models (Feng et al., 2020), often incorporating syntax-aware architectures to capture the structural nature of code (Yin and Neubig, 2017). While these methods mark significant progress over rule-based baselines (Allamanis and Sutton, 2014), they rely on large amounts of labeled language-code pairs, limiting coverage and incurring considerable implementation costs.

Most recent progress stems from LLMs (Chen et al., 2021a; Fried et al., 2023; Guo et al., 2024) (continuously) pretrained on massive *unlabeled code* from GitHub and StackOverflow. These models exhibit strong *zero-* and *few-shot* learning capabilities, often requiring minimal tuning or just prompt engineering to excel at coding tasks (Barke et al., 2023; Zheng et al., 2024a; Zhang et al., 2024b). As model sizes grow, LLMs demonstrate emergent capabilities such as debugging (Kang et al., 2025). Despite showing promise as coding assistants, LLMs can still introduce bugs (Nguyen and Nadi, 2022), indicating a need for further refinement before reaching human-level competence.

While data selection has been increasingly studied for NL generation, it remains *under-explored* for NL2Code. In contrast to NL domains, where selecting instruction data drives sophisticated algorithms, NL2Code datasets are predominantly **unlabeled** (HuggingFace, 2021; Kocetkov et al., 2022). Consequently, current practice for NL2Code is limited to basic filtering techniques to ensure code files are deduplicated, complete, and clean (Chen et al., 2021a; Li et al., 2022; Fried et al., 2023; Nijkamp et al., 2023): remove incomplete or auto-generated files and discard rarely used repos. While these heuristics offer decent opportunities for *filtering out* undesired code, they are not meant for *filtering in* ("finding") the relevant code for a target domain.

Move beyond coarse heuristics, we introduce FINDR, a more fine-grained data selection algorithm *tailored* to NL2Code, while extensible to other *large-scale unlabeled scenarios*. To our knowledge, this is the first systematic study of data

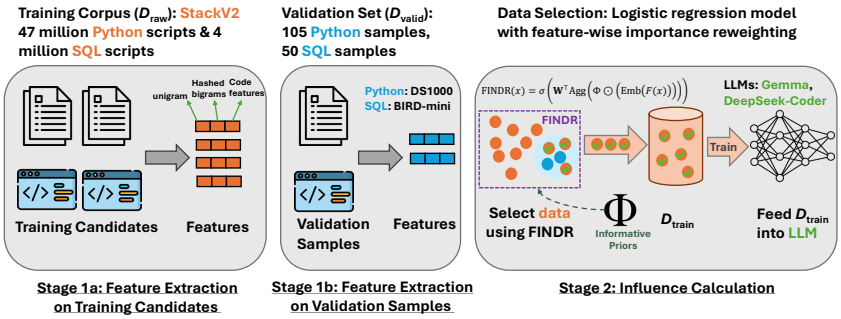

Figure 2: Overview of FINDR. We first extract code-feature-augmented representations (§3.1), and then leverage informative priors to apply feature importance reweighting to compute influence scores (§3.2). The light blue bubble (stage 2) denotes decision boundary.

selection for NL2Code, enabling more efficient and targeted pretraining of sLLMs for code generation.

3 Method: FINDR

Figure 2 provides an overview of the two stages behind FINDR, **feature extraction** and **influence calculation**, specifically instantiated in the NL2Code setting. Here, we aim to select the most influential subset of data, $\mathcal{D}_{train}$, from a large corpus, $\mathcal{D}_{raw}$, for continued pretraining of LLMs.

To begin, we obtain **code-tailored features** by running our feature extractor (§3.1) over the entire training corpus ($\mathcal{D}_{raw}$). Next, we utilize a small set of validation examples, $\mathcal{D}_{val}$, which mirrors the target test data, $\mathcal{D}_{test}$. Indeed, small-scale validation sets have proven effective for model tuning and domain adaptation (Kirstain et al., 2022; Zhou et al., 2023; Zhang et al., 2024c). The stage 1 process is highly flexible: whenever we have a new target dataset ($\mathcal{D}_{test}$) or a shift in domain, we can easily gather a reasonably small validation set to guide the data selection. Hence, FINDR becomes a plug-and-play solution that can seamlessly adapt to evolving requirements.

Stage 2 is where we measure the **influence** of candidate training points with respect to the small validation set ($\mathcal{D}_{val}$), detailed in §3.2. By focusing on the most influential data, we can reduce training time and resource consumption, which is especially critical for LLMs. After computing the influence scores, we feed the selected data into an LLM for **continued pretraining**. Notably, this can be done either *stochastically*—sampling data points based on normalized FINDR scores (with high-scoring samples possibly repeated in $\mathcal{D}_{train}$)—or *deterministically* by selecting the top $k\%$[3] of $\mathcal{D}_{raw}$ to construct $\mathcal{D}_{train}$, ensuring stable coverage of top-scoring examples. We adopt the deterministic approach because, as seen in preliminary experiments, it yields more consistent improvements and simplifies hyperparameter tuning, leaving the stochastic approach for future exploration.

3.1 Feature Extraction (Stage 1)

Considering our goal is to extract features for a massive number of data points in an efficient manner, we choose to trade off some representational expressiveness for higher efficiency. Instead of relying on semantically rich embedding methods (e.g., static (Pennington et al., 2014) and generative (Devlin et al., 2019) embeddings), we adopt n-gram bag-of-words as a practical solution. Inspired by existing work on feature extraction for NL data (Joulin et al., 2017; Xie et al., 2023), we note that using unigrams alone fails to capture subtle surface-level signals, such as word-pair interactions and ordering cues, but enumerating all bigrams is intractable. Hashed bigrams, instead, strike a practical balance by reducing computational overhead while retaining valuable contextual information.

Beyond textual features, we additionally introduce a code-specific representation—**code feature**—that model programming language-specific functions/patterns. For instance, one feature bucket tracks the frequency of NumPy array creation functions (e.g., `numpy.array()`, `numpy.zeros()`). This helps highlight operations that are particularly relevant to coding. Refer to Table A6 for more examples of code-feature buckets. To summarize, our feature extractor encapsulates both lexical and semantic properties of code (i.e., surface-level unigram, hashed bigrams, and abstracted function descriptions), enabling richer representation for subsequent selection steps.

[3] In this paper, we set a fixed $k = 2$ unless otherwise noted, similarly to Xia et al. (2024).

3.2 Influence Calculation (Stage 2)

In order to efficiently compute influence scores from the feature representations introduced in §3.1, we build upon a logistic regression (LR) model. LR has long been favored in large-scale natural language understanding tasks for its simplicity, ease of interpretation, and robust performance (Brown et al., 2020; Chowdhery et al., 2023; Gao et al., 2021). Equation (1) depicts our FINDR score calculation for each data point x, extending prior LR-based models with **feature-wise importance reweighting** and incorporating a novel $\Phi[\cdot]$ function that implements informative priors.

$$\text{FINDR}(x) = \sigma\left(\mathbf{W}^\top \text{Agg}\left(\Phi \odot \left(\text{Emb}\left(F(x)\right)\right)\right)\right) \tag{1}$$

where the parameters $\mathbf{W}$ and $\text{Emb}[\cdot]$ are trainable, while $\text{Agg}[\cdot]$ denotes the aggregation function (i.e., generating document-level representations), and $F(x) \in \mathbb{R}^N$ is the extracted feature for an input data x (described in §3.1).

The **feature-wise importance** score is computed using $\Phi[\cdot] \in \mathbb{R}^N$,[4] formulated as in eq. (2).

$$\Phi[\mathcal{D}_{val}, \mathcal{D}_{raw}] = \min(\text{REG}[\frac{\Phi'_{\mathcal{D}_{val}}[\mathbf{f}]}{\Phi'_{\mathcal{D}'_{raw}}[\mathbf{f}]}], M) \tag{2}$$

where $\Phi'_{\mathcal{D}}[\mathbf{f}]$ is the frequency-based raw importance score (eq. (3)), REG is a regularization factor balancing priors vs. uniform weights (eq. (4)), and M caps scores to prevent feature-wise shortcuts.

Raw Importance Score Calculation. We begin with computing *raw importance scores*,[5] by counting the relative frequency of each individual feature for two sets of data: $\mathcal{D}_{val}$ as the positive set and $\mathcal{D}'_{raw}$[6] as the negative set. Formally,

$$\Phi'_{\mathcal{D}}[\mathbf{f}] = \frac{\sum_{j=1}^{n} \mathbf{f}_j}{\sum_{i=1}^{n} \mathbf{1}^\top \mathbf{f}_i} \tag{3}$$

where $\mathbf{f}_i$ represents the feature vector for the code file i, extracted as in §3.1, and n denotes the size $|\mathcal{D}|$. This frequency-based score calculation strikes a balance between simplicity and scalability, making it especially appropriate for large collections of unlabeled code.

Regularization Component (REG). The REG component serves as a regularization mechanism controlled by the hyperparameter $\gamma \in [0, 1]$. The γ modulates the reliance on *priors* versus *uniform feature weighting* as follows:

- $\gamma = 1$**:** The model reduces to a standard LR model, assigning equal weights to features.
- $\gamma = 0$**:** The model fully leverages priors, allowing nuanced feature distinctions.

$$\text{REG}[\phi] = \gamma(1 - \phi) + \phi \tag{4}$$

where ϕ is instantiated as the difference of raw importance scores between positive and negative sets, as shown in eq. (2).

Furthermore, to address the size imbalance between $\mathcal{D}_{val}$ and $\mathcal{D}'_{raw}$, we introduce a rescaling factor C, and thus replace ϕ with $\frac{\phi}{C}$ in Equation (4). In fact, such imbalances can lead to *skewed feature importance scores*, particularly when smaller sets disproportionately influence the learning process (Henning et al., 2023). In contrast, the factor C ensures comparability between sets of varying sizes. Specifically, we implement two types of rescaling factors: *accumulated feature count difference (AFC)* and *document count difference (DC)*. That is, AFC addresses the size difference based on the total number of feature occurrences, while DC is only concerned about the number of code files.

Capping Scores. In our preliminary studies, we find that certain features, e.g., project-specific variable names, can show up in short bursts, yielding tremendously large importance scores. Those *rare yet inconsequential* tokens will, however, disrupt the training process of FINDR. Thereby, as indicated in eq. (2), we cap each feature's importance score at M to resolve such anomalies and prevent FINDR from picking up unexpected artifacts.

4 Experiments

4.1 Datasets and Evaluation Metrics

Pretraining corpus. We consider **StackV2** (Lozhkov et al., 2024) as $\mathcal{D}_{raw}$ for selecting $\mathcal{D}_{train}$ for our target tasks. StackV2 is a large-scale code corpus of more than 3 billion files in 600+ programming languages, primarily sourced from public GitHub repos. Our focus is on two subsets: Python and SQL. Combined, these subsets comprise approximately 50 million scripts with an average length of 3,412 characters. Specifically, there are 46.64M Python scripts (totaling 300GB) and 3.63M SQL scripts (totaling 40GB).

Evaluation benchmarks. To assess performance and generalizability, we evaluate baselines and

[4] For simplicity, we use $\Phi[\cdot]$ to represent $\Phi[\mathcal{D}_{val}, \mathcal{D}_{raw}]$.

[5] We use "influence" to denote data point-level FINDR score, while "importance" means feature-wise weights.

[6] The construction of $\mathcal{D}'_{raw}$ is detailed in §E.1.

	DeepSeek-Coder						Gemma					
	Origin	Surface	Semantic	Difficult	Perturbation	Overall	Origin	Surface	Semantic	Difficult	Perturbation	Overall
Base Model	19.9	9.2	17.5	6.8	12.0	15.1	13.8	7.2	9.8	4.9	7.6	10.1
Random Selection	20.7	8.6	16.7	4.9	11.0	14.7	15.3	6.6	10.7	**6.2**	8.2	10.9
Quality Classifier	21.2	9.3	15.8	6.2	11.2	15.1	17.0	**10.5**	11.5	**6.2**	9.7	12.5
BM25	22.2	6.6	14.1	5.6	9.5	14.4	**21.0**	7.9	**14.5**	4.9	9.8	**14.2**
DSIR	22.2	9.9	17.5	5.9	12.0	15.9	15.3	5.3	12.4	5.6	8.4	11.1
FINDR (Ours)	**24.2**	**12.2**	**18.4**	**7.1**	**13.3**	**17.5**	19.0	9.2	14.1	**6.2**	**10.4**	13.7

Table 1: Comparison of FINDR with data selection baselines in the Python domain, measured by Pass@1, when training with 2% of selected data. Base model denotes out-of-the-box evaluation without additional training. Per Lai et al. (2022), we conduct 0-shot evaluation, and we report individual results on 4 problem types and the aggregated perturbation set. Best results are **bold**, and informed data selectors that outperform the base model are highlighted on a scale of 5 red shades (color schemes in §B). Overall, FINDR improves base Coder and Gemma by 16% and 36%. Notably, FINDR attains the highest score on perturbed items, showcasing the robustness of FINDR. Efficiency comparison, i.e., data selection efficiency, is provided in Table A4 (*Selection Time* column).

FINDR on two target tasks/domains ($\mathcal{D}_{test}$).[7] In the Python domain, we focus on the still largely unresolved **DS-1000** (Lai et al., 2022), instead of widely studied benchmarks like HumanEval (Chen et al., 2021b) and MBPP (Austin et al., 2021), which have approached saturated performance (Table A8). DS-1000 comprises $1,000$ data science-oriented code generation problems spanning seven scientific computing libraries, e.g., NumPy. For SQL, we adopt the challenging **BIRD** (Li et al., 2024b), which comprises 95 databases across 37 professional areas, and narrows the gap between experimental and real-world settings seen in other benchmarks (Zhong et al., 2017; Yu et al., 2018). In this work, we use its recent derivative, BIRD-miniDev,[8] released in June 2024, which supports diverse database management systems.

Following the literature (Xia et al., 2024), we also hold out a subset of examples as $\mathcal{D}_{val}$ for guiding data selection. Statistics are shown in Table A5.

Evaluation metrics. For Python DS1000 (Lai et al., 2022), we use the **pass@1 accuracy**, which evaluates functional correctness based on test case success and adherence to surface-form constraints (e.g., mandatory use of vectorized operations). For SQL (Li et al., 2024b), we report **Execution (EX)**, which checks if predicted and ground-truth queries produce identical results, and **Soft F1-score**, which measures the similarity between the tables produced by generated and reference SQL queries.

4.2 Experiment Setup

We include recent small LLMs that excel at reasoning and code completion tasks: coding-specialized sLLM, **DeepSeek-Coder-1.3B-base** (hearafter, Coder; Guo et al., 2024), and generalist sLLM, **Gemma-2B** (Mesnard et al., 2024).

As our goal is to study if a selected subset of influential data can boost sLLM performance on the target task, we only perform **continued pre-training** on $\mathcal{D}_{train}$ without additional fine-tuning. We use base versions of the models and employ Llama-Factory (Zheng et al., 2024b) for parallel training with 8 40GB GPUs. We set the context length to $4,096$, gradient accumulation to 32 and per-device batch size to 1. Models are trained for 2 epochs for Python and 3 epochs for SQL. For all other hyperparameters, we keep the default values.

For evaluation, we follow the official evaluation protocols (Lai et al., 2022; Li et al., 2024b), and use greedy decoding with few-shot demonstrations if needed (0-shot for Python and 1-shot for SQL).

FINDR Setup and Training: We build $\mathcal{D}_{\text{FINDR}}$ for training FINDR, using $\mathcal{D}_{val}$ and a sample of $\mathcal{D}_{raw}$ as positive and negative sets. In the feature extractor, we use all unigrams in $\mathcal{D}_{\text{FINDR}}$ and, for bigrams, apply the FNV-1a algorithm (Fowler et al., 2012) to obtain hashed features using $100k$ buckets. The code feature is enabled only for Python, and we semi-automatically define 618 classes (buckets), covering $8,721$ Python functions.[9] The training process of FINDR consists of two stages: first, learning $\Phi[\cdot]$ (§E.2), and then supervised learning on $\mathcal{D}_{\text{FINDR}}$,[10] which updates the randomly initialized parameters $\mathbf{W}$ and $\text{Emb}[\cdot]$ for 10 epochs.

4.3 Data Selector Baselines

We include major *efficient* prtreaining data selection baselines. The simplest baseline is **random selection**, where we randomly sample data from the training corpus. For informed data selection

[7] We rigorously compared nearly 20 benchmarks to determine the most suitable ones for our evaluation (Table A7).

[8] https://github.com/bird-bench/mini_dev

[9] See §D for semi-automatic construction process.

[10] $|\mathcal{D}^{\text{Python}}_{\text{FINDR}}| = 1,000$ and $|\mathcal{D}^{\text{SQL}}_{\text{FINDR}}| = 500$.

	DeeoSeek-Coder				Gemma			
	Easy	Med.	Hard	Overall	Easy	Med.	Hard	Overall
Base	**26.5**	8.8	**2.0**	11.1	12.2	2.8	**3.9**	5.1
Random	18.4	5.7	**2.0**	7.6	11.2	2.8	2.9	4.7
Quality	19.4	3.8	1.0	6.6	**18.9**	2.0	2.9	5.9
BM25	22.5	6.4	**2.0**	8.9	17.9	**3.2**	2.0	6.1
DSIR	4.1	0.0	0.0	0.9	6.6	0.6	0.0	1.8
FINDR	25.5	**11.2**	**2.0**	**12.2**	**18.9**	2.8	**3.9**	**6.6**

Table 2: EM performance in the SQL domain when training with 2% of selected data (F1 in Table A2). Base model denotes out-of-the-box evaluation. Following Li et al. (2024b), we conduct 1-shot evaluation, and we report individual results on 3 problem types. Best results are **bold**, and data selectors superior to Base are highlighted on a scale of 5 red shades. In general, FINDR leads to the best performance across the board.

baselines, we compare with **BM25** (Robertson and Zaragoza, 2009), which is based on word frequency statistics to rank examples to determine how relevant a training document is. Another baseline, LR-based **Quality Classifier**, is widely used for filtering and selecting data from large-scale pretraining corpora (Gao et al., 2021; Chowdhery et al., 2023). We also compare to prior art for selecting unlabelled data, **DSIR** (Xie et al., 2023). It applies n-gram features to weight candidate training data through Naive Bayes formulation, and sample data points accordingly.

We do not compare to instruction data selection approaches, due to extremely *slow* pace and *high* compute cost (Wang et al., 2024), such as representation-(Xiao et al., 2023) and gradient-based (Xia et al., 2024) methods, often exceeding 10k GPU hours to process StackV2 Python subset.

5 Results

5.1 Main Results and Analyses

We present our main results in Table 1 and Table 2 for Python and SQL domains, respectively, compared against baseline approaches. More results are in §C. We summarize five key findings.

1) *Random selection often degrades performance.* It can be tempting to assume that adding more data—no matter how it is sampled—will improve model performance. But in line with pre-LLM findings (Moore and Lewis, 2010; van der Wees et al., 2017), our experiments confirm that randomly chosen data can degrade performance, even compared with using the base model out-of-the-box. Thus, in the absence of an informed data selector, defaulting to the base model is preferred.

2) FINDR *selects on-target data that consistently boosts base models across domains.* As has been shown, training on FINDR-selected data consistently enhances code generation capabilities across all evaluated sLLMs. In particular, overall performance gains range from 16% to 36% on Python and 9% to 29% on SQL. These results suggest that FINDR can indeed select the most influential examples, which works robustly across experimental settings.

3) FINDR *generally outperforms strong baselines by non-trivial margins*. Apart from Gemma model on Python, FINDR performs the best across the board. Interestingly, BM25 sometimes surpasses FINDR, but at a *significant computational cost*: it has the slowest selection pace (Figure 1), and the Python scripts it selects are five times longer (Table A4). As a result, token counts rise substantially, thereby increasing LLM training time. In contrast, FINDR offers a more *balanced* solution for *token-efficiently* capturing the most influential data. Notably, on "Perturbed" Python examples—which mitigate potential data leakage from LLM pretraining—FINDR achieves the highest scores, underscoring its robust performance in settings that rely less on memorized knowledge. For further evidence, see Table A1 and Table A2.

4) FINDR *demonstrates superior robustness on "Difficult" examples.* Beyond the overall performance leap, a key advantage of FINDR is its robustness in handling "difficult" scenarios. On Python's "Difficult" split, FINDR consistently yields the largest improvement relative to other baselines, indicating that its selected examples effectively target the reasoning skills needed for complex code generation. Likewise in SQL, FINDR preserves, and sometimes improves (Table A2), the ability to generate challenging SQL queries, whereas several baselines worsen performance in these tough cases.

5) *NL-targeted selectors do not necessarily excel at NL2Code.* Finally, we note that DSIR (Xie et al., 2023), SOTA method for selecting pure NL data, proves much less effective when adapting to code, especially on the SQL domain. Indeed, as discussed in Xia et al. (2024), we have also observed that DSIR-selected examples are extremely short, thus weakening the code generation capacity of trained models. Furthermore, our finding highlights a notable gap between NL-only and NL2Code data, while our method, FINDR, offers—to the best of our knowledge—the first solution to help bridge the gap.

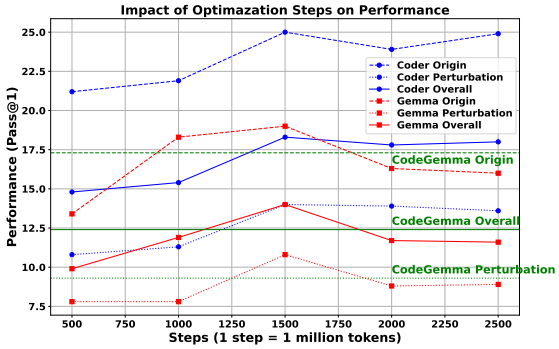

Figure 3: Pass@1 scores of continuously trained sLLMs in the Python domain, in relation to the optimization steps. We identify the performance peaking at around $1,500$ steps, trained with 1.5B *on-target* tokens, and Gemma even outperforming CodeGemma, despite the latter being trained on 500B *general* tokens.

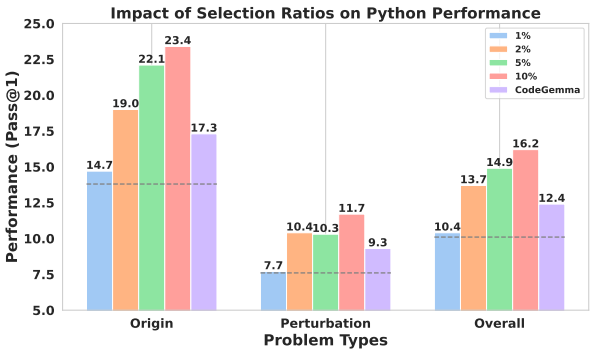

Figure 4: Gemma results in Python domain with varying selection ratios. Dashed line denotes the off-the-shelf Gemma's result. Performance improves steadily with higher selection ratios, with a notable cut-off at 2%, where additional training yields benefits. Complete results refer to Figure A1. We also observe similar trend for the SQL domain, shown in Figure A2.

5.2 Further Study on Data Selection

Data, the essential component in this study, has been the driving force behind ever capable LLMs. Here, we are particularly interested in how LLMs respond to varying *training data*. We address this question under two conditions: (1) varying the **optimization steps** (which translates to the number of training tokens), and (2) varying the **selection ratio** (i.e., the proportion of data selected by FINDR).

Impact of Optimization Steps. We first analyze how increasing the training budget—in terms of optimization steps—impacts final performance. Figure 3 shows performance trends for Coder and Gemma up to $2,500$ steps.[11] Both models improve until about $1,500$ steps (1.5B tokens), after which Coder plateaus and Gemma slightly declines. Thus, $1,500$ steps provides a clear balance of performance gains and training efficiency under the default 2% selection ratio. We also compare Gemma to CodeGemma (Zhao et al., 2024), which benefits from extra 500B tokens of continued pretraining. Despite CodeGemma being a stronger LLM, Gemma continuously trained on 1.5B *on-target* tokens chosen by FINDR effectively closes the gap. This highlights the advantage of informed data selection like FINDR—it consumes just $\mathbf{0.3}\%$ of the CodeGemma training tokens yet outperforms massive-scale training at random.

Impact of Selection Ratio. We next examine how different selection ratios (1%, 2%, 5%, 10%) affect performance, as depicted in Figure 4 and Figure A2. Dashed lines denote the out-of-the-box Gemma's performance. On both tasks, continuously trained models exceed the base model once the ratio reaches 2%. Beyond that threshold, performance generally rises further, albeit with marginal gains from 2% to 5% in certain splits (e.g., Python Perturbation, SQL Hard). Moreover, for Python, starting at 2%, Gemma trained on FINDR-selected data surpasses CodeGemma despite its extensive pretraining. However, no such leap is observed for SQL, even at 10%. This is likely due to the fact that Python's full set (47M scripts) far exceeds SQL's (4M), so 10% of SQL data still translates to fewer samples than 1% of Python.

In summary, our findings reveal that both the size of the training budget (i.e., the number of steps) and FINDR's selection ratio play significant roles in reshaping downstream NL2Code capabilities. More importantly, *informed data selection* can substantially improve performance with only a small fraction of the entire corpus. In the future, we will explore optimal training steps for each selection ratio and further investigate the scaling law for informed data selection.

5.3 Ablation Study of FINDR

We conduct ablation experiments to analyze contributions of design elements in FINDR.

5.4 η Ratio for Φ Estimation

We vary η[12] from 1 to 100 to assess how the sampling size of negatives for *informative priors* es-

[11] Each optimization step processes 1M tokens, so 500–2,500 steps correspond to training on 0.5B–2.5B tokens.

[12] a hyperparameter introduced in the learning process of $\Phi[\cdot]$ (§E.2).

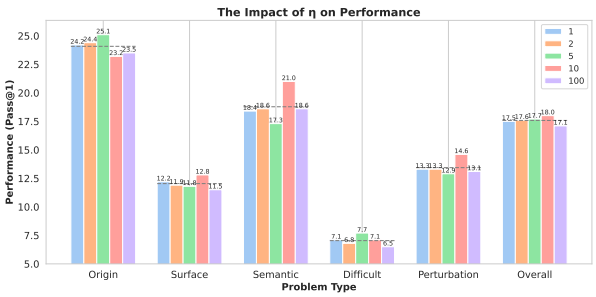

Figure 5: Impact of η (sampling size of negatives for informative priors estimation) on performance in Python domain. We have witnessed a steady performance boost from $\eta = 1$ to 10, but a sharp drop afterwards, likely due to diminishing returns from overly large sampling of negatives.

timation influences downstream performance. Intuitively, a larger set should yield more accurate feature-weight estimates, thus improving downstream selection. Figure 5 confirms this trend for most problem types, recording a steady performance boost from $\eta = 1$ to 10. However, a noticeable drop is observed beyond that point, as evidenced by the lowest overall result when $\eta = 100$, likely due to diminishing returns from overly large sampling of negatives.

Balancing efficiency and accuracy, we adopt $\eta = 1$ in our main experiments. This setting requires minimal computational overhead yet maintains near-peak performance. Future work can explore adaptive strategies for tuning η to improve the *informative priors* estimation.

Code Features and Rescaling Approach. We ablate the *code feature* in feature extractor (§3.1), and test DC-based versus AFC-based *rescaling* approaches (§3.2). As displayed in Table 3, removing code features consistently degrades results across all splits, showing the importance of code-specific features in code-related tasks (Nasrabadi et al., 2023; Jiang et al., 2024). While DC rescaling sometimes performs competitively, AFC generally yields superior results. This reflects the benefit of fine-grained rescaling for data imbalance issues (Henning et al., 2023). Altogether, the ablation studies validate the design choices in FINDR.

6 Conclusion

In this work, we introduce FINDR, an efficient pretraining data selection method based on logistic regression but enhanced with feature importance reweighting. Concretely, we augment hashed n-gram features with *code features* to capture code-specific constructs, then apply informative priors to *reweight* feature importance when computing *influence* scores. Notably, our FINDR is the first data selection algorithm tailored to NL2Code pretraining. Experiments on Python and SQL demonstrate FINDR's superiority over strong baselines and its compatibility across diverse sLLMs. Our further study confirms that a small, influential subset of data can yield significant performance improvements, even outperforming an LLM trained on 300 times more examples.

	Gemma					
	Origin	Surface	Semantic	Difficult	Perturbation	Overall
FINDR	**19.0**	**9.2**	**14.1**	**6.2**	**10.4**	**13.7**
- Code feature	17.4	8.6	11.1	4.6	8.5	12.0
FINDR (*DC* rescaling)	18.3	6.9	11.5	**6.2**	8.7	12.4
- Code feature	**19.0**	**9.2**	12.8	3.7	9.1	12.9

Table 3: Ablation study of FINDR. We find that removing code features consistently degrades results across all splits. Complete results (incl. DS-Coder) see Table A3.

Limitation

GPU resources. The base sLLMs used for continued-pretraining in this work are of 1.3 to 2.5 billions parameters. It is thus more time-consuming than training smaller previous-generation models like BART (Lewis et al., 2020), which in turn results in a significantly higher carbon footprint. Specifically, we train each model on 8 NVIDIA A100 (40GB VRAM) with significant CPU and memory resources. The training time for each model ranges from several hours to 2 days, depending on the configurations.

Meanwhile, due to the limited GPU resources at hand (8 NVIDIA A100-40GB), this work serves as a *pilot study* to rigorously assess the effectiveness of various data selection methods for continued pretraining of small LLMs on the NL2Code task. We will study the scaling effect (i.e., increasing model sizes) in future work, as additional compute becomes available.

Evaluation Domains. In this work, we have included two challenging evaluation benchmarks, aiming to cover a diverse array of code styles and domains. Yet, these two benchmarks cannot comprehensively represent the entire spectrum of the NL2Code space. Indeed, evaluation remains an ongoing challenge in data selection——existing studies typically rely on only 3–4 benchmarks as well (Xia et al., 2024; Li et al., 2024a). In future research, we plan to extend FINDR to more program-

ming languages, e.g., Java and C++, and examine its robustness as new large-scale benchmarks are constructed.

Generalizability of FINDR. In this work, we focus primarily on developing and validating FINDR for NL2Code. As is common in this area (Wang et al., 2024), evaluations are typically performed on the motivating target domains only, leaving the question of generalizability to a broader range of domains for future work. For instance, DSIR (Xie et al., 2023), a SOTA data selector in the natural language (NL) space, performs poorly in the coding space (Table 1, Table 2). Therefore, we plan subsequent work focusing on extending the evaluation of FINDR to non-NL and non-NL2Code domains, while expecting others to also investigate FINDR beyond NL2Code as we have done with the strong baselines (e.g., DSIR) in this work.

References

AI@Meta. 2024. Llama 3 model card.

Alon Albalak, Yanai Elazar, Sang Michael Xie, Shayne Longpre, Nathan Lambert, Xinyi Wang, Niklas Muennighoff, Bairu Hou, Liangming Pan, Haewon Jeong, Colin Raffel, Shiyu Chang, Tatsunori Hashimoto, and William Yang Wang. 2024. A survey on data selection for language models. *Trans. Mach. Learn. Res.*, 2024.

Miltiadis Allamanis, Earl T. Barr, Premkumar T. Devanbu, and Charles Sutton. 2018. A survey of machine learning for big code and naturalness. *ACM Comput. Surv.*, 51(4):81:1–81:37.

Miltiadis Allamanis and Charles Sutton. 2014. Mining idioms from source code. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, pages 472–483. ACM.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Hannah McLean Babe, Sydney Nguyen, Yangtian Zi, Arjun Guha, Molly Q Feldman, and Carolyn Jane Anderson. 2024. Studenteval: A benchmark of student-written prompts for large language models of code. In *Findings of the Association for Computational Linguistics*.

Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded copilot: How programmers interact with code-generating models. 7(OOPSLA1).

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Yihan Cao, Yanbin Kang, Chi Wang, and Lichao Sun. 2023. Instruction mining: Instruction data selection for tuning large language models.

Shubham Chandel, Colin B. Clement, Guillermo Serrato, and Neel Sundaresan. 2022. Training and evaluating a jupyter notebook data science assistant. *CoRR*, abs/2201.12901.

Hao Chen, Yiming Zhang, Qi Zhang, Hantao Yang, Xiaomeng Hu, Xuetao Ma, Yifan Yanggong, and Junbo Zhao. 2023. Maybe only 0.5% data is needed: A preliminary exploration of low training data instruction tuning. *CoRR*, abs/2305.09246.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021a. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie

Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021b. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2023. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113.

Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. 2020. Selection via proxy: Efficient data selection for deep learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Enrique Dehaerne, Bappaditya Dey, Sandip Halder, Stefan De Gendt, and Wannes Meert. 2022. Code generation using machine learning: A systematic review. *IEEE Access*, 10:82434–82455.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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, Minneapolis, Minnesota. Association for Computational Linguistics.

Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten P. Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen S. Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V. Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. 2022. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 5547–5569. PMLR.

Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. 2023. Classeval: A manually-crafted benchmark for evaluating llms on class-level code generation. *Preprint*, arXiv:2308.01861.

Yanai Elazar, Akshita Bhagia, Ian Magnusson, Abhilasha Ravichander, Dustin Schwenk, Alane Suhr, Evan Pete Walsh, Dirk Groeneveld, Luca Soldaini, Sameer Singh, Hannaneh Hajishirzi, Noah A. Smith, and Jesse Dodge. 2024. What's in my big data? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to-sequence attentional neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 823–833, Berlin, Germany. Association for Computational Linguistics.

Yukun Feng, Patrick Xia, Benjamin Van Durme, and João Sedoc. 2022. Automatic document selection for efficient encoder pretraining. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9522–9530, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.

Glenn Fowler, Landon Noll, Kiem-Phong Vo, and Donald E. Eastlake 3rd. 2012. The FNV Non-Cryptographic Hash Algorithm. Internet-Draft draft-eastlake-fnv-03, Internet Engineering Task Force. Work in Progress.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. 2023. Incoder: A generative model for code infilling and synthesis. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang,

Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2021. The pile: An 800gb dataset of diverse text for language modeling. *CoRR*, abs/2101.00027.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna M. Wallach, Hal Daumé III, and Kate Crawford. 2021. Datasheets for datasets. *Commun. ACM*, 64(12):86–92.

Brent A. Griffin, Jacob Marks, and Jason J. Corso. 2024. Zero-shot coreset selection: Efficient pruning for unlabeled data. *CoRR*, abs/2411.15349.

Chengcheng Guo, Bo Zhao, and Yanbing Bai. 2022. Deepcore: A comprehensive library for coreset selection in deep learning. In *Database and Expert Systems Applications - 33rd International Conference, DEXA 2022, Vienna, Austria, August 22-24, 2022, Proceedings, Part I*, volume 13426 of *Lecture Notes in Computer Science*, pages 181–195. Springer.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming - the rise of code intelligence. *CoRR*, abs/2401.14196.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online. Association for Computational Linguistics.

Xiaochuang Han, Daniel Simig, Todor Mihaylov, Yulia Tsvetkov, Asli Celikyilmaz, and Tianlu Wang. 2023. Understanding in-context learning via supportive pretraining data. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12660–12673, Toronto, Canada. Association for Computational Linguistics.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Sophie Henning, William Beluch, Alexander Fraser, and Annemarie Friedrich. 2023. A survey of methods for addressing class imbalance in deep-learning based natural language processing. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 523–540, Dubrovnik, Croatia. Association for Computational Linguistics.

Junjie Huang, Chenglong Wang, Jipeng Zhang, Cong Yan, Haotian Cui, Jeevana Priya Inala, Colin Clement, and Nan Duan. 2022. Execution-based evaluation for data science code generation models. In *Proceedings of the Fourth Workshop on Data Science with Human-in-the-Loop (Language Advances)*, pages 28–36, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.

HuggingFace. 2021. Github code dataset. https://huggingface.co/datasets/codeparrot/github-code.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2073–2083, Berlin, Germany. Association for Computational Linguistics.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *CoRR*, abs/2310.06825.

Jing Jiang and ChengXiang Zhai. 2007. Instance weighting for domain adaptation in NLP. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 264–271, Prague, Czech Republic. Association for Computational Linguistics.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *CoRR*, abs/2406.00515.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, Valencia, Spain. Association for Computational Linguistics.

Sungmin Kang, Bei Chen, Shin Yoo, and Jian-Guang Lou. 2025. Explainable automated debugging via large language model-driven scientific debugging. *Empir. Softw. Eng.*, 30(2):45.

KrishnaTeja Killamsetty, Xujiang Zhao, Feng Chen, and Rishabh K. Iyer. 2021. RETRIEVE: coreset selection for efficient and robust semi-supervised learning.

In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 14488–14501.

Yuval Kirstain, Patrick Lewis, Sebastian Riedel, and Omer Levy. 2022. A few more examples may be worth billions of parameters. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 1017–1029, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. 2022. The stack: 3 tb of permissively licensed source code. *Preprint*.

Humaira Kousar, Hasnain Irshad Bhatti, and Jaekyun Moon. 2025. Pruning-based data selection and network fusion for efficient deep learning. *CoRR*, abs/2501.01118.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-Tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2022. Ds-1000: A natural and reliable benchmark for data science code generation. *ArXiv*, abs/2211.11501.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Dongyue Li, Ziniu Zhang, Lu Wang, and Hongyang R. Zhang. 2024a. Scalable fine-tuning from multiple data sources: A first-order approximation approach. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 5608–5623, Miami, Florida, USA. Association for Computational Linguistics.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024b. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. 2024c. From quantity to quality: Boosting LLM performance with self-guided data selection for instruction tuning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7602–7635, Mexico City, Mexico. Association for Computational Linguistics.

Yujia Li, David H. Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with alphacode. *CoRR*, abs/2203.07814.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. 2024. What makes good data for alignment? A comprehensive study of automatic data selection in instruction tuning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian J. McAuley, Han Hu, Torsten Scholak, Sébastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, and et al. 2024. Starcoder 2 and the stack v2: The next generation. *CoRR*, abs/2402.19173.

Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Cristian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, and et al.

2024. Gemma: Open models based on gemini research and technology. *CoRR*, abs/2403.08295.

Robert C. Moore and William Lewis. 2010. Intelligent selection of language model training data. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 220–224, Uppsala, Sweden. Association for Computational Linguistics.

Morteza Zakeri Nasrabadi, Saeed Parsa, Mohammad Ramezani, Chanchal Roy, and Masoud Ekhtiarzadeh. 2023. A systematic literature review on source code similarity measurement and clone detection: Techniques, applications, and challenges. *J. Syst. Softw.*, 204:111796.

Nhan Nguyen and Sarah Nadi. 2022. An empirical evaluation of github copilot's code suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22, page 1–5, New York, NY, USA. Association for Computing Machinery.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. Codegen: An open large language model for code with multi-turn program synthesis. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. 2021. Deep learning on a data diet: Finding important examples early in training. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 20596–20607.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Jeff M. Phillips. 2016. Coresets and sketches. *CoRR*, abs/1601.00617.

Stephen E. Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

Noveen Sachdeva, Benjamin Coleman, Wang-Cheng Kang, Jianmo Ni, Lichan Hong, Ed H. Chi, James Caverlee, Julian J. McAuley, and Derek Zhiyuan Cheng. 2024. How to train data-efficient llms. *CoRR*, abs/2402.09668.

Hidetoshi Shimodaira. 2000. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90:227–244.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Raghavi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Harsh Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Pete Walsh, Luke Zettlemoyer, Noah A. Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. 2024. Dolma: an open corpus of three trillion tokens for language model pretraining research. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 15725–15788. Association for Computational Linguistics.

Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert Müller. 2007. Covariate shift adaptation by importance weighted cross validation. *J. Mach. Learn. Res.*, 8:985–1005.

Marlies van der Wees, Arianna Bisazza, and Christof Monz. 2017. Dynamic data selection for neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1400–1410, Copenhagen, Denmark. Association for Computational Linguistics.

Jiahao Wang, Bolin Zhang, Qianlong Du, Jiajun Zhang, and Dianhui Chu. 2024. A survey on data selection for LLM instruction tuning. *CoRR*, abs/2402.05123.

Shiqi Wang, Li Zheng, Haifeng Qian, Chenghao Yang, Zijian Wang, Varun Kumar, Mingyue Shang, Samson Tan, Baishakhi Ray, Parminder Bhatia, Ramesh Nallapati, Murali Krishna Ramanathan, Dan Roth, and Bing Xiang. 2022. Recode: Robustness evaluation of code generation models.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Raghavi Chandu, David Wadden, Kelsey MacMillan, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. 2023a. How far can camels go? exploring the state of instruction tuning on open resources. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Zhiruo Wang, Shuyan Zhou, Daniel Fried, and Graham Neubig. 2023b. Execution-based evaluation for open-domain code generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1271–1290, Singapore. Association for Computational Linguistics.

Shengguang Wu, Keming Lu, Benfeng Xu, Junyang Lin, Qi Su, and Chang Zhou. 2023. Self-evolved diverse data sampling for efficient instruction tuning. *CoRR*, abs/2311.08182.

Mengzhou Xia, Antonios Anastasopoulos, Ruochen Xu, Yiming Yang, and Graham Neubig. 2020. Predicting performance for natural language processing tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 8625–8646. Association for Computational Linguistics.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. 2024. LESS: selecting influential data for targeted instruction tuning. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Xiaobo Xia, Jiale Liu, Jun Yu, Xu Shen, Bo Han, and Tongliang Liu. 2023. Moderate coreset: A universal method of data selection for real-world data-efficient deep learning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding. *Preprint*, arXiv:2309.07597.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2024, Washington DC, USA, July 14-18, 2024*, pages 641–649. ACM.

Sang Michael Xie, Shibani Santurkar, Tengyu Ma, and Percy Liang. 2023. Data selection for language models via importance resampling. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Yichen Xu and Yanqiao Zhu. 2022. A survey on pretrained language models for neural code intelligence. *CoRR*, abs/2212.10079.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada. Association for Computational Linguistics.

Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang, and Tao Xie. 2024. Codereval: A benchmark of pragmatic code generation with generative pre-trained models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*, pages 37:1–37:12. ACM.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Wang Yongji, and Jian-Guang Lou. 2023. Large language models meet NL2Code: A survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7443–7464, Toronto, Canada. Association for Computational Linguistics.

Shudan Zhang, Hanlin Zhao, Xiao Liu, Qinkai Zheng, Zehan Qi, Xiaotao Gu, Yuxiao Dong, and Jie Tang. 2024a. NaturalCodeBench: Examining coding performance mismatch on HumanEval and natural user queries. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 7907–7928, Bangkok, Thailand. Association for Computational Linguistics.

Xinliang Frederick Zhang, Nicholas Beauchamp, and Lu Wang. 2024b. Narrative-of-thought: Improving temporal reasoning of large language models via recounted narratives. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 16507–16530, Miami, Florida, USA. Association for Computational Linguistics.

Xinliang Frederick Zhang, Carter Blum, Temma Choji, Shalin Shah, and Alakananda Vempala. 2024c. ULTRA: Unleash LLMs' potential for event argument extraction through hierarchical modeling and pairwise self-refinement. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 8172–8185, Bangkok, Thailand. Association for Computational Linguistics.

Heri Zhao, Jeffrey Hui, Joshua Howland, Nam Nguyen, Siqi Zuo, Andrea Hu, Christopher A. Choquette-Choo, Jingyue Shen, Joe Kelley, Kshitij Bansal, Luke Vilnis, Mateo Wirth, Paul Michel, Peter Choy, Pratik Joshi, Ravin Kumar, Sarmad Hashmi, Shubham Agrawal, Zhitao Gong, Jane Fine, Tris Warkentin, Ale Jakse Hartman, Bin Ni, Kathy Korevec, Kelly Schaefer, and Scott Huffman. 2024. Codegemma: Open code models based on gemma. *CoRR*, abs/2406.11409.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue.

2024a. OpenCodeInterpreter: Integrating code generation with execution and refinement. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 12834–12859, Bangkok, Thailand. Association for Computational Linguistics.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. 2024b. LlamaFactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 400–410, Bangkok, Thailand. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. LIMA: less is more for alignment. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*.

A Rubrics for Radar Plot

In order to produce the overall comparison radar plot (Figure 1), we consider five distinct metrics.

- **Selection Efficiency:** This metric is derived from the *Selection Time* column in Table A4. Specifically, we first apply add-one smoothing to all raw values, followed by a logarithmic transformation, and is finally rescaled to a range of 1 to 5 using min-max normalization. Note, for all raw values (unit: hours), they represent the entire time cost of selecting 2% of data from the Python set of StackV2, including parameter learning and inference (i.e., data selection) stages. For the random baseline, the time cost is trivial, so we assign 0.
- **Token Efficiency:** This metric is directly derived from the *Character Count* column in Table A4, and is rescaled to a range of 1 to 5 using min-max normalization. The raw value indicates the average number of characters in a Python script within the selected data.
- **Compatibility:** TThe purpose of this metric is to assess whether a data selection approach can boost performance for both evaluated models, and whether it exhibits any model preference or bias. This metric is relatively subjective, and the scores we assign to each approach are displayed in Table A4. For example, both Quality Classifier and BM25 yield improvements when the underlying model is Gemma, but no improvement is observed for Coder. Therefore, we rate their compatibility as moderate (3). However, since Quality classifier does not hurt the performance in the Python domain, we slightly increase its score to 3.5.
- **Python Skill:** This metric is directly derived from Table 1, and is rescaled to a range of 1 to 5 using min-max normalization.
- **SQL Skill:** This metric is directly derived from Table 2, and is rescaled to a range of 1 to 5 using min-max normalization.

Finally, we derive the **overall rating** by placing equal emphasis on both efficiency and effective/performance dimensions. The efficiency dimension encompasses the selection and token efficiencies, while the other three metrics are grouped under the effective/performance dimension. At the authors' discretion, we assign the five metrics weights of $[1, 1, 0.5, 0.75, 0.75]$, considering that compatibility is a relatively subjective metric. We then compute a weighted sum of per-metric ranks for each data selection approach, which is subsequently rescaled to a range of 1 to 5 using min-max normalization.

B Color Scheme

For all data selectors (including both FINDR and baseline approaches), we highlight them on a scale of 5 red shades based on the relative improvements over the off-the-shelf base models. We design the following scheme to color Table 1, Table 2, Table A1 and Table A2:

- if the relative gain is in the range of $(0\%, 5\%]$, the value is highlighted in pale pink.
- if the relative gain is in the range of $(5\%, 15\%]$, the value is highlighted in pink.
- if the relative gain is in the range of $(15\%, 30\%]$, the value is highlighted in rose-pink.
- if the relative gain is in the range of $(30\%, 50\%]$, the value is highlighted in rose-red.
- if the relative gain is over 50%, the value is highlighted in dark red.

C Supplementary Main Results

Due to space limitation in the main text, this section supplements §5.1.

Python. Table A1 presents results where features for validation data are extracted from the *complete* script, including both *context* and *answer*, whereas for Table 1, features are extracted solely from the *context*. Overall, the performance difference between the *context-only* and *complete* script settings is minimal. Therefore, for all the other experiments performed in the Python domain, we extract features using only the context for two reasons: (1) Context is shorter than the complete script, making feature extraction more efficient, and (2) incorporating solutions requires extensive human annotation, which limits scalability.

SQL. As discussed in §4.1, we adopt two evaluation metrics for the SQL domain: EM and F1. Table A2 complements Table 2 by presenting performance in terms of F1. Note, for all SQL domain experiments, features for validation data are only extracted from the *answer*. This choice is based on our observation that using *context-only* or *complete*

	DeepSeek-Coder						Gemma					
	Origin	Surface	Semantic	Difficult	Perturbation	Overall	Origin	Surface	Semantic	Difficult	Perturbation	Overall
Base Model	19.9	9.2	17.5	6.8	12.0	15.1	13.8	7.2	9.8	4.9	7.6	10.1
Random Selection	20.7	8.6	16.7	4.9	11.0	14.7	15.3	6.6	10.7	6.2	8.2	10.9
Quality Classifier	**23.6**	9.9	17.9	6.8	12.4	16.8	19.9	7.9	12.4	**7.4**	9.7	13.6
BM25	22.8	6.6	15.4	5.6	10.1	15.0	**23.1**	5.3	15.0	6.2	9.7	**14.9**
DSIR	21.0	9.2	**18.8**	6.8	12.6	15.9	16.4	7.9	12.0	4.9	8.8	11.7
FINDR (Ours)	23.3	**12.5**	**18.8**	**7.4**	**13.7**	**17.4**	20.1	**9.2**	**16.2**	3.7	**10.6**	14.3

Table A1: Comparison of FINDR with strong data selection baseline approaches in the Python domain, measured by Pass@1, when training with 2% of selected data. Base model denotes out-of-the-box evaluation without additional training. In contrast to Table 1, features of validation examples are extracted from the complete script (i.e., context and answer). Following Lai et al. (2022), we conduct 0-shot evaluation, and we report individual results on 4 problem types and the aggregated perturbation set. Best results are **bold**, and informed data selectors that outperform the base model are highlighted on a scale of 5 red shades (see color schemes in §B). Overall, FINDR improves over base Coder and Gemma by 15% and 39%, respectively. Notably, FINDR achieves the highest score on perturbed items, showcasing the robustness of FINDR.

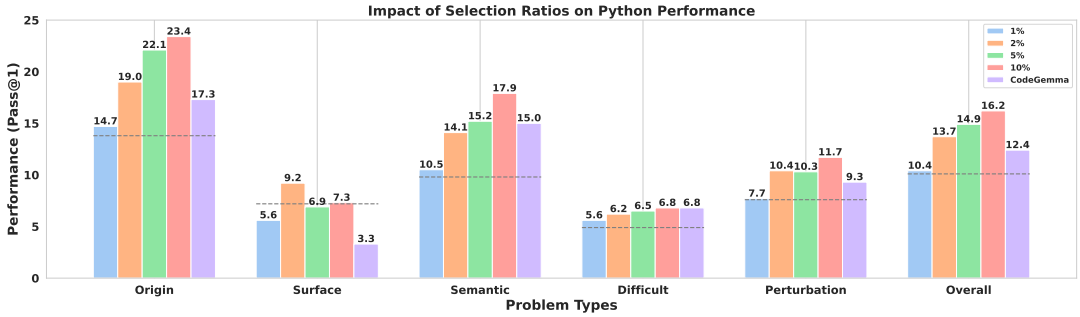

Figure A1: Gemma results in Python domain with varying selection ratios. Dashed line denotes the off-the-shelf Gemma's result. Performance improves steadily with higher selection ratios, with a notable cut-off at 2%, where additional training leads to better performance than base Gemma and CodeGemma.

	DeeoSeek-Coder				Gemma			
	Simple	Moderate	Challenging	Overall	Simple	Moderate	Challenging	Overall
Base	**28.4**	10.7	3.3	12.8	13.3	3.6	4.6	6.0
Random	19.1	6.5	**4.1**	8.7	12.8	3.4	4.2	5.6
Quality	19.8	4.4	1.5	7.1	**18.5**	2.7	3.3	6.3
BM25	24.0	7.9	3.3	10.3	**19.6**	**4.3**	4.1	**7.6**
DSIR	4.1	0.1	0.0	1.0	7.0	0.7	0.8	1.8
FINDR	27.5	**11.7**	2.7	**13.1**	**20.0**	4.1	**4.9**	**7.7**

Table A2: F1 performance comparison in the SQL domain when training with 2% of selected data. Base model denotes out-of-the-box evaluation. Following Li et al. (2024b), we conduct 1-shot evaluation, and we report individual results on 3 problem types. Best results are **bold**, and data selectors superior to base are highlighted on a scale of 5 red shades. In general, FINDR leads to the best performance across the board. EM results refer to Table 2.

scripts results in inferior performance and higher results variance.

D Construction of Code Feature Buckets

To construct a comprehensive set of code feature buckets, we leverage powerful LLMs such as GPT-3.5.[13] Using the Numpy library as an example, we prompt GPT-3.5 to generate major function categories and their corresponding expressions. This process is performed three times with varying output sizes: 10, 50, and 70 classes. We merge the results, removing only duplicated expressions while retaining all unique classes. This procedure is repeated for each of the seven libraries. In total, we compile 618 feature classes encompassing 8,721 distinct Python expressions. Examples of these feature buckets are presented in Table A6.

E More details of Informative Priors (Φ)

The informative priors (Φ) are intended to capture global beliefs about the relative importance of different features. In an ideal scenario, Φ would be independent of any specific experience. However, in practice, Φ inevitably depends on the target domain. Moreover, generating an exhaustive list of importance scores for all features in the training set is impractical—especially since the training set is dynamic, and even if it were fixed, manually assigning these scores would be prohibitively labor-intense. To this end, we decide to estimate Φ in an on-demand fashion using Equation (2).

[13] https://chat.openai.com/; gpt-35-turbo-16k-0613, training data up to Sept. 2021

	DeepSeek-Coder						Gemma					
	Origin	Surface	Semantic	Difficult	Perturbation	Overall	Origin	Surface	Semantic	Difficult	Perturbation	Overall
FINDR	**24.2**	**12.2**	18.4	**7.1**	13.3	**17.5**	**19.0**	**9.2**	**14.1**	**6.2**	**10.4**	**13.7**
- Code feature	22.5	11.2	18.8	6.8	13.1	16.8	17.4	8.6	11.1	4.6	8.5	12.0
FINDR (*DC* rescaling)	23.4	10.6	18.6	6.8	12.9	17.0	18.3	6.9	11.5	**6.2**	8.7	12.4
- Code feature	22.2	11.9	**20.5**	6.5	**14.0**	17.2	**19.0**	**9.2**	12.8	3.7	9.1	12.9

Table A3: Ablation study of FINDR. We find that removing code features consistently degrades results across all splits. Meanwhile, using DC rescaling approach generally hurts the performance, in comparison with the default AFC.

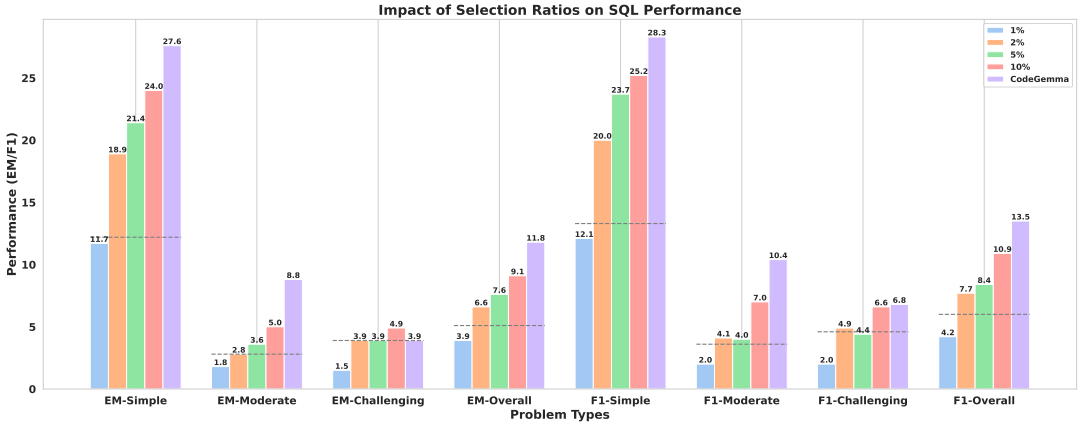

Figure A2: Gemma results (EM and FI) in SQL domain with varying selection ratios. Dashed line denotes the off-the-shelf Gemma's result. Performance improves steadily with higher selection ratios, with a notable cut-off at 2%, where additional training leads to better performance than base Gemma.

	Selection Time	Character Count	Compatibility
Random	0	3,410	2.0
Quality	2	3,855	3.5
BM25	760	10,788	3.0
DSIR	145	533	3.5
FINDR (Ours)	3.5	1,762	5.0

Table A4: Raw statistics for radar plot (Figure 1), and efficiency comparison (i.e., selection time) among select baselines and FINDR. The unit for selection time is *hours*, including parameter learning and inference stages. The detailed usage of each column is documented in §A.

E.1 Construction of $\mathcal{D}'_{raw}$

To address the large size disparity between $\mathcal{D}_{val}$ and $\mathcal{D}_{raw}$, and mitigate shortcut learning (see §3.2), we introduce a reduced negative set, $\mathcal{D}'_{raw}$. Specifically, $\mathcal{D}'_{raw}$ is a small subset of $\mathcal{D}_{raw}$ that serves as a proxy for negative examples. To stabilize subsequent supervised learning stage of FINDR, we ensure that all negative instances in $\mathcal{D}_{\text{FINDR}}$ are contained within $\mathcal{D}'_{raw}$. We further introduce a hyperparameter η, which defines the size of $\mathcal{D}'_{raw}$ as as a multiple of the negative set in $\mathcal{D}_{\text{FINDR}}$. That is, $|\mathcal{D}'_{raw}| = \eta|\mathcal{D}^{-}_{\text{FINDR}}|$. By tuning η, we can balance data coverage against computational overhead.

E.2 Learning Process of Φ

After obtaining $\mathcal{D}'_{raw}$, we combine it with $\mathcal{D}'_{val}$ (serving as the positive set) to learn Φ. The learning process of Φ is detailed in §3.2. Once Φ is learned, the corresponding parameters are frozen in the subsequent supervised learning stage.

E.3 Default Setting of Φ

Our FINDR method introduces three core hyperparameters in Φ:

- γ**:** Balances the contribution of the priors vs. uniform weighting ($\gamma = 0.75$ by default).
- M**:** Caps the maximum importance score for each feature ($M = 3$ by default).
- η**:** Controls the ratio of negative samples between the training set and the Prior estimation set ($\eta = 1$ by default, due to efficiency and representativeness).

Benchmarks	$\lvert\mathcal{D}\_\text{val}\rvert$	$\lvert\mathcal{D}\_\text{test}\rvert$	Splits	$\lvert C\rvert$	$\lvert A\rvert$	#Shot	#Tasks	Domain
DS1000 (Lai et al., 2022)	105	895	452 (105)/152/234/162	2,857	141	0	7	Python
BIRD-miniDev (Li et al., 2024b)	50	450	148 (50)/250/102	4,270	201	1	12	SQL

Table A5: Statistics of evaluation benchmarks. $\lvert\mathcal{D}\_\text{val}\rvert$ and $\lvert\mathcal{D}\_\text{test}\rvert$ denote the size of validation and test sets. Splits represent the fine-grained data splits by problem types, as seen in Table A1 and Table 2. That is, there are *Origin, Surface, Semantic* and *Difficult* in the Python domain, and *Simple, Moderate* and *Challenging* in the SQL domain. We also ensure that all validation data are sampled from the simplest category, as indicated by parentheses, allowing for the study of LLM generalizability and true intelligence. $|P|$ and $|A|$ denote the average length of *context* (C) and *answer* (A). For #Shot, we follow the official practice in respective benchmarks (Lai et al., 2022; Li et al., 2024b). #Tasks represent the number of libraries (Python) and subjects (SQL) included in each benchmark.

Library	**Feature Bracket**	**Expressions/Functions**
Matplot	Plotting Functions	`pyplot.plot`, `plot`, `matplotlib.pyplot.hist`, `plt.hist`, `boxplot`, `plt.scatter`, `bar`, `matplotlib.pyplot.boxplot`, `matplotlib.pyplot.plot`, `scatter`, `matplotlib.pyplot.bar`, `plt.bar`, `pyplot.bar`, `plt.plot`, `pyplot.scatter`, `pyplot.hist`, `hist`, `pyplot.boxplot`, `plt.boxplot`, `matplotlib.pyplot.scatter`
Numpy	Array Creation	`np.ones`, `numpy.eye`, `array`, `numpy.zeros`, `numpy.array`, `np.zeros`, `np.array`, `numpy.ones`, `empty`, `zeros`, `np.eye`, `ones`, `eye`, `np.empty`, `numpy.empty`
Pandas	Input/Output	`to_csv`, `pd.read_json`, `pandas.read_csv`, `pd.read_sql`, `pandas.to_csv`, `pandas.read_html`, `pandas.read_sql`, `read_csv`, `read_html`, `read_json`, `pandas.read_json`, `pd.to_csv`, `read_sql`, `pd.read_html`, `pd.read_csv`, `pandas.read_excel`, `pd.read_excel`, `read_excel`
PyTorch	Math Operations	`torch.log`, `torch.cos`, `add`, `torch.sub`, `pow`, `sub`, `torch.sqrt`, `exp`, `sin`, `cos`, `sqrt`, `mul`, `div`, `torch.sin`, `torch.exp`, `torch.mul`, `log`, `torch.pow`, `torch.add`, `torch.div`
SciPy	Data Structures	`scipy.sparse.dok_matrix`, `coo_matrix`, `scipy.sparse.coo_matrix`, `scipy.sparse.bsr_matrix`, `scipy.sparse.lil_matrix`, `sparse.lil_matrix`, `sparse.dok_matrix`, `lil_matrix`, `scipy.sparse.csc_matrix`, `bsr_matrix`, `sparse.coo_matrix`, `csc_matrix`, `sparse.csc_matrix`, `sparse.bsr_matrix`, `dok_matrix`
Sklearn	Model Selection	`sklearn.model_selection.KFold`, `model_selection.GridSearchCV`, `StratifiedKFold`, `sklearn.model_selection.GridSearchCV`, `cross_val_score`, `model_selection.train_test_split`, `KFold`, `sklearn.model_selection.cross_val_score`, `train_test_split`, `model_selection.StratifiedKFold`, `sklearn.model_selection.train_test_split`, `GridSearchCV`, `sklearn.model_selection.StratifiedKFold`, `model_selection.KFold`, `model_selection.cross_val_score`
TensorFlow	Tensor Manipulation	`tf.constant`, `tf.Variable`, `concat`, `tf.concat`, `Variable`, `constant`, `tf.reshape`, `reshape`, `transpose`, `tf.transpose`

Table A6: Example feature brackets. For each library in DS1000, we show one bracket with associated expressions.

Benchmark	Size	#PL
HumanEval (Chen et al., 2021b)	164	Python
HumanEval+ (Liu et al., 2023)	164	Python
MBPP (Austin et al., 2021)	974	Python
MBPP+ (Liu et al., 2023)	378	Python
Spider (Yu et al., 2018)	8,034	SQL
BIRD-Dev (Li et al., 2024b)	500	SQL
ODEX (Wang et al., 2023b)	945	Python
CoderEval (Yu et al., 2024)	460	Python, Java
ReCode (Wang et al., 2022)	1,138	Python
StudentEval (Babe et al., 2024)	1,749	Python
BigCodeBench (Zhuo et al., 2024)	1,140	Python
ClassEval (Du et al., 2023)	100	Python
NaturalCodeBench (Zhang et al., 2024a)	140	Python, Java
LiveCodeBench (Jain et al., 2025)	713	Python
DSP (Chandel et al., 2022)	1,119	Python
ExeDS (Huang et al., 2022)	534	Python
DS-1000 (Lai et al., 2022)	1,000	Python

Table A7: Overview of major NL2Code benchmarks. We pick BIRD-DEV and DS-1000 based on the following rationales: 1) we chose Python for it being widely used in NL2Code benchmarks, and we adopted DS-1000 due to the unsaturated performance (also see Table A8); 2) we included SQL, particularly BIRD-Dev, because of the introduced challenges due to its complexity and recency (2024), helping mitigate data contamination issues; and 3) we excluded Java, as the Java partitions of NaturalCodeBench only contains 70 problems, which seemed insufficient to reveal models' true performance. For each benchmark, we report its size and programming languages covered.

Model	Human Eval (%)	MBPP (%)
DeepSeek-Coder-1.3B (Guo et al., 2024)	65.9	65.3
DeepSeek-Coder-6.7B (Guo et al., 2024)	74.4	74.9
GPT-4 Turbo (OpenAI, 2023)	85.4	85.7

Table A8: Model performance on two widely adopted NL2Code benchmarks, Human Eval and MBPP. As explained in §4.1, we do not use these two since they have approached saturated performance. Results are reported as of 04/03/2025.