FINDR:
A Fast Influential Data Selector for NL2Code Pretraining

Xinliang Frederick Zhang and Lu Wang

Computer Science and Engineering, University of Michigan

AACL 2025

Introduction: Data Selection

Preamble: Large language models (LLMs) have become an exceptionally powerful technology, thanks to the use of enormous pretraining data.
When moving to downstream tasks (e.g., NL2Code), much of this original pretraining data become a distraction at best or a liability at worst.

Introduction: Data Selection

Preamble: Large language models (LLMs) have become an exceptionally powerful technology, thanks to the use of enormous pretraining data.
When moving to downstream tasks (e.g., NL2Code), much of this original pretraining data become a distraction at best or a liability at worst.

Problem Space: Given a large collection of unlabeled examples demonstrating multiple capabilities, how can we effectively and efficiently select a small portion of influential training data?

Introduction: Data Selection for NL2Code

Preamble: Large language models (LLMs) have become an exceptionally powerful technology, thanks to the use of enormous pretraining data.
When moving to downstream tasks (e.g., NL2Code), much of this original pretraining data become a distraction at best or a liability at worst.

Problem Space: Given a large collection of **unlabeled examples** demonstrating multiple capabilities, how can we effectively and efficiently select a small portion of influential training data?

In this project, we focus on the under-studied domain of **Data Selection for NL2Code**

Related Work

- Several Pretraining (PT) data selection methods are studied in the literature

Related Work

- Several Pretraining (PT) data selection methods are studied in the literature
- Data Selection = Feature Extraction + Importance Calculation

Related Work

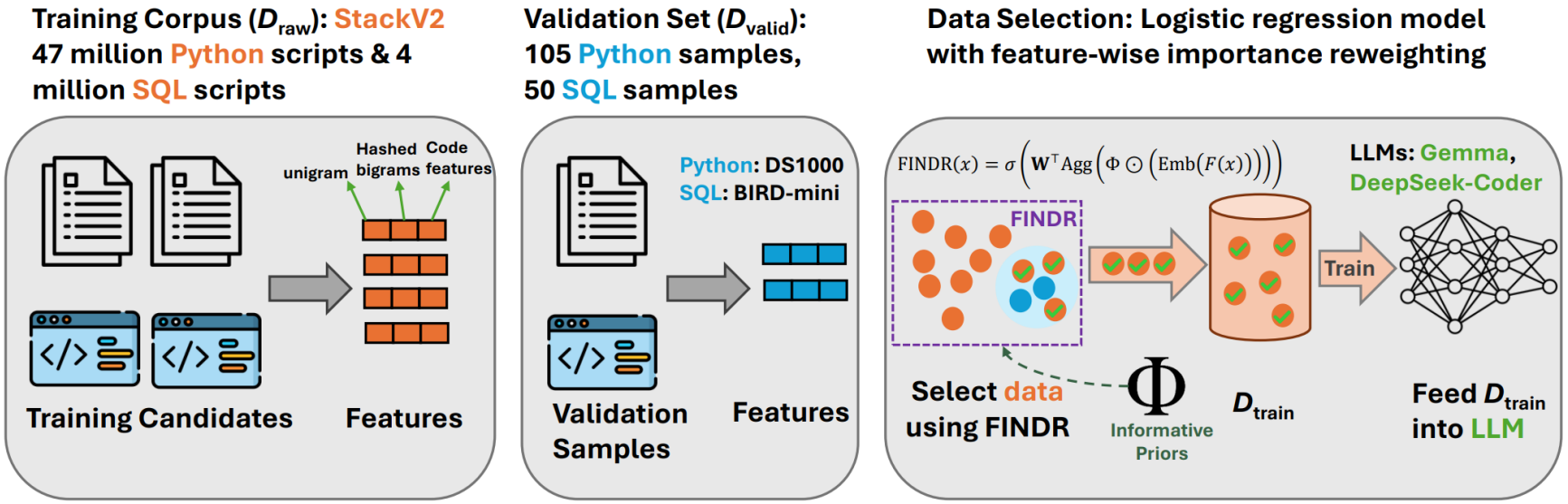
- Several Pretraining (PT) data selection methods are studied in the literature
- Data Selection = Feature Extraction + Importance Calculation

Method	Architecture		Representative issue
	Feature extraction	Importance calculation	
Random	/	/	Uninformed selection
BM25	N-grams	Non-parametric (pairwise comparison)	Selected data being too long
DSIR (*NeurIPS'23*)	Hashed n-grams	Parametric (Naïve Bayes)	Selected data being too short
Quality Classifier (*the Pile, Palm*)	Hashed n-grams	Parametric (logistic regression)	Lack of diversity
Advanced Embedding (BGE; *SIGIR'24*)	Distributed rep	Non-parametric (pairwise comparison)	Lack of scalability
LESS (*ICML'24*)	Gradient feature	Non-parametric (pairwise comparison)	Lack of scalability
External Judge (ChatGPT; *NeurIPS'23*)	/	LLM prompting	Lack of scalability OR fail to capture nuance

Related Work: Limitations & Challenges

- Existing work solely focus on selecting NL (natural language)-only data. It is unclear and unexplored about its efficacy on NL2Code and programming data.
- Advanced methods face scalability issue, while “fast” methods either fail to select data with characteristics similar to the target domain (e.g., length) or lack diversity.

Method: FINDR (Fast INfluential Data Ranker)



Stage 1a: Feature Extraction on Training Candidates

Stage 1b: Feature Extraction on Validation Samples

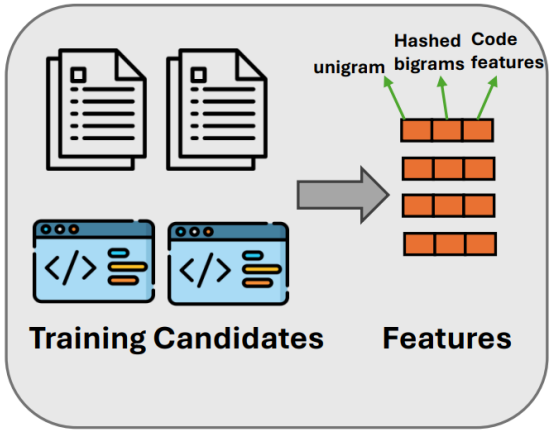
Stage 2: Influence Calculation

Method: FINDR (Fast INfluential Data Ranker)

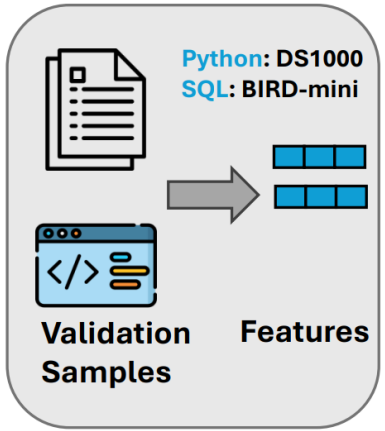
Training Corpus (D_{raw}): StackV2
47 million Python scripts & 4 million SQL scripts

Validation Set (D_{valid}):
105 Python samples,
50 SQL samples

Stage 1 (Feature Extraction)



Stage 1a: Feature Extraction on Training Candidates

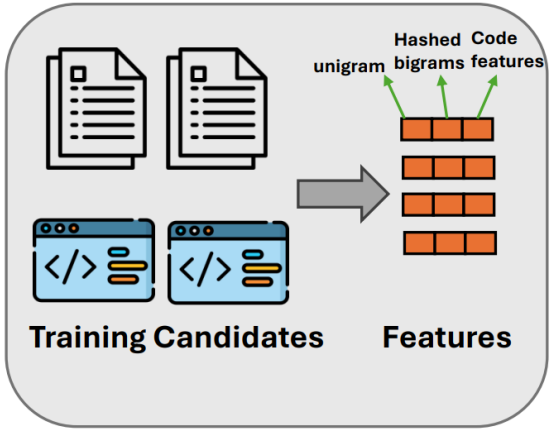


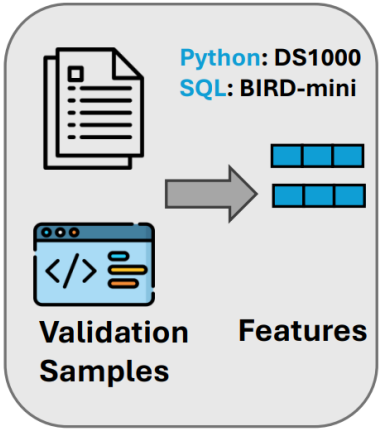
Stage 1b: Feature Extraction on Validation Samples

Method: FINDR (Fast INfluential Data Ranker)

Training Corpus (D_{raw}): StackV2
47 million Python scripts & 4 million SQL scripts

Validation Set (D_{valid}):
105 Python samples,
50 SQL samples

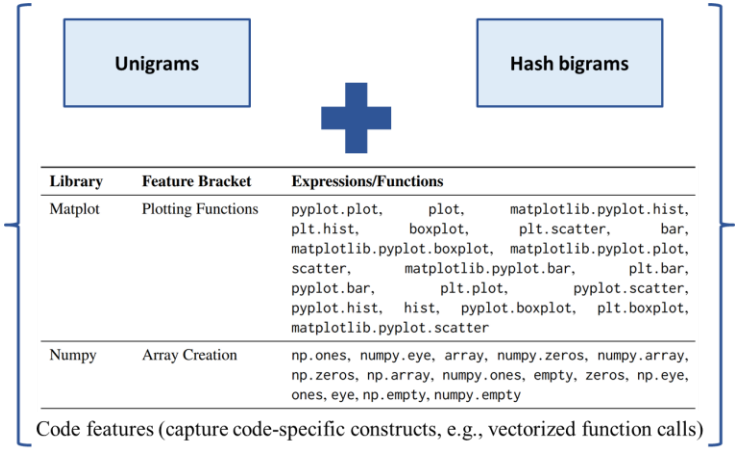




Stage 1a: Feature Extraction on Training Candidates

Stage 1b: Feature Extraction on Validation Samples

Stage 1 (Feature Extraction)



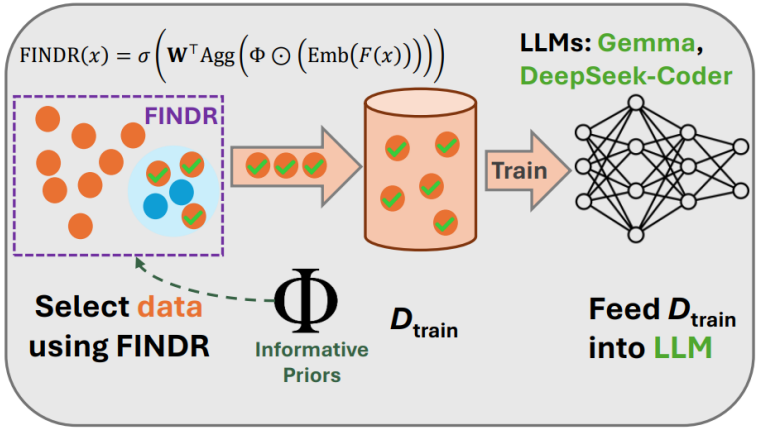
Library	Feature Bracket	Expressions/Functions
Matplot	Plotting Functions	pyplot.plot, plot, matplotlib.pyplot.hist, plt.hist, boxplot, plt.scatter, bar, matplotlib.pyplot.boxplot, matplotlib.pyplot.plot, scatter, matplotlib.pyplot.bar, plt.bar, pyplot.bar, plt.plot, pyplot.scatter, pyplot.hist, hist, 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

Code features (capture code-specific constructs, e.g., vectorized function calls)

Method: FINDR (Fast INfluential Data Ranker)

Stage 2 (Influence Calculation)

Data Selection: Logistic regression model with feature-wise importance reweighting



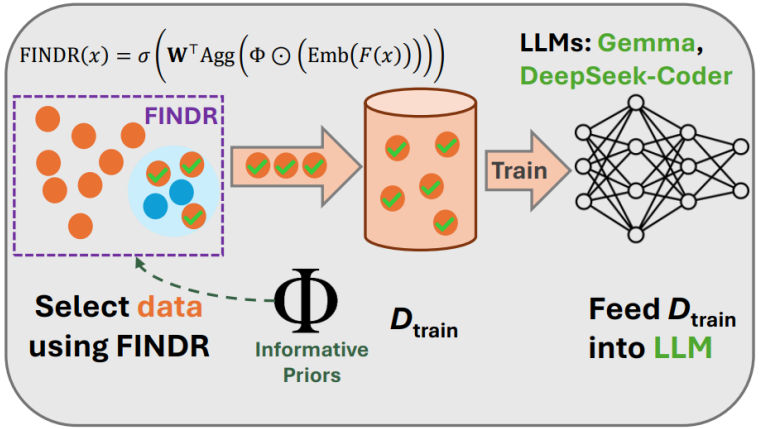
Stage 2: Influence Calculation

Method: FINDR (Fast INfluential Data Ranker)

Stage 2 (Influence Calculation)

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

Data Selection: Logistic regression model with feature-wise importance reweighting



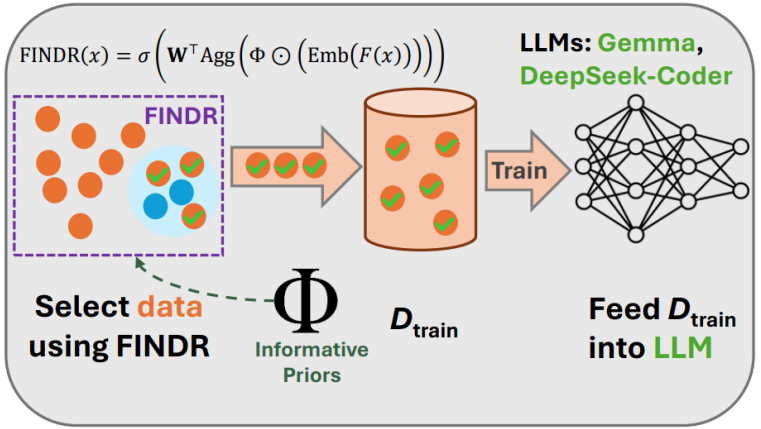
Stage 2: Influence Calculation

Method: FINDR (Fast INfluential Data Ranker)

Stage 2 (Influence Calculation)

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

Data Selection: Logistic regression model with feature-wise importance reweighting



Stage 2: Influence Calculation

Method: FINDR (Fast INfluential Data Ranker)

Stage 2 (Influence Calculation)

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

Feature-wise importance is determined by:

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

Φ' is the frequency-based raw importance for each feature:

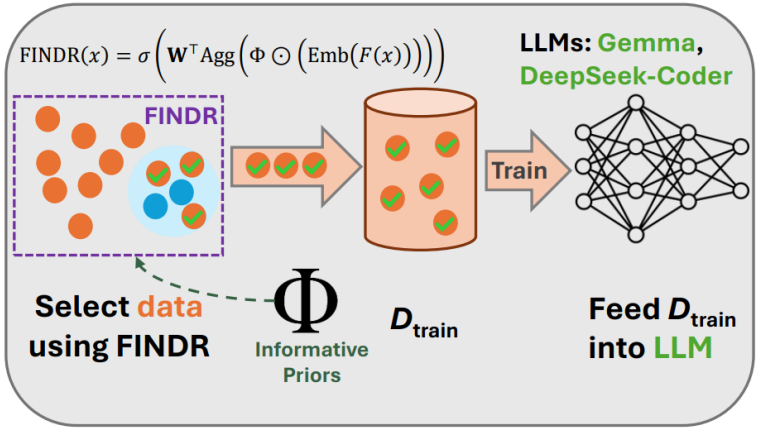
$$\Phi'_{\mathcal{D}}[\mathbf{f}] = \frac{\sum_{j=1}^{n} \mathbf{f}_j}{\sum_{i=1}^{n} \mathbf{1}^\top \mathbf{f}_i}$$

REG is a regularization factor balancing priors vs. uniform weights, controlled by hyper-parameter γ:

$$\mathrm{REG}[\phi] = \gamma(1-\phi) + \phi$$

M caps importance to prevent FINDR learning feature-wise shortcuts.

Data Selection: Logistic regression model with feature-wise importance reweighting



Stage 2: Influence Calculation

Experimental Results

	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

Comparison of FINDR with data selection baselines in the Python domain, measured by Pass@1.

	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**

Comparison of FINDR with data selection baselines in the SQL domain, measured by Execution (EX).

Experimental Results

	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

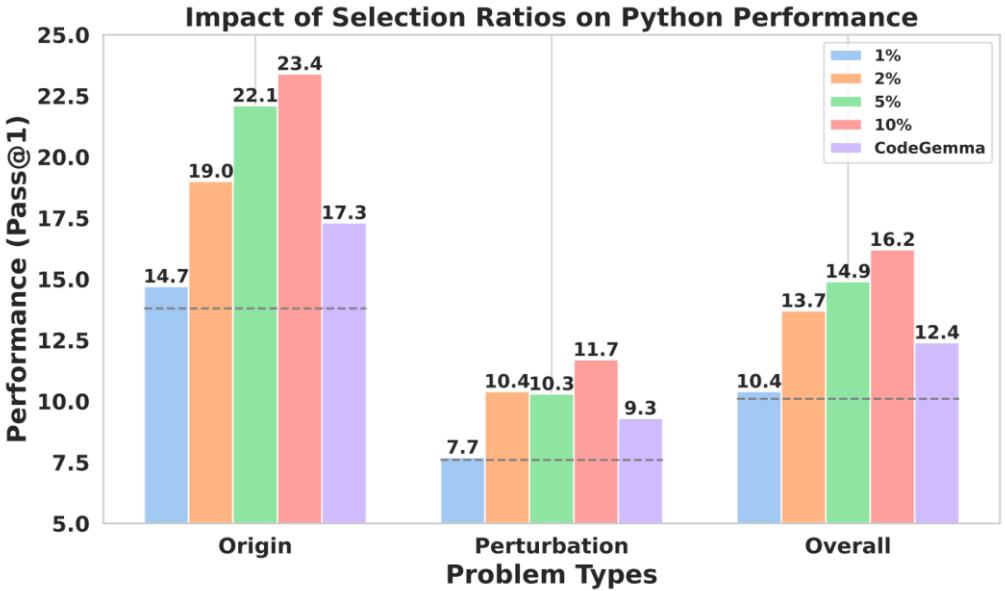
Comparison of FINDR with data selection baselines in the Python domain, measured by Pass@1.

	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**

Comparison of FINDR with data selection baselines in the SQL domain, measured by Execution (EX).

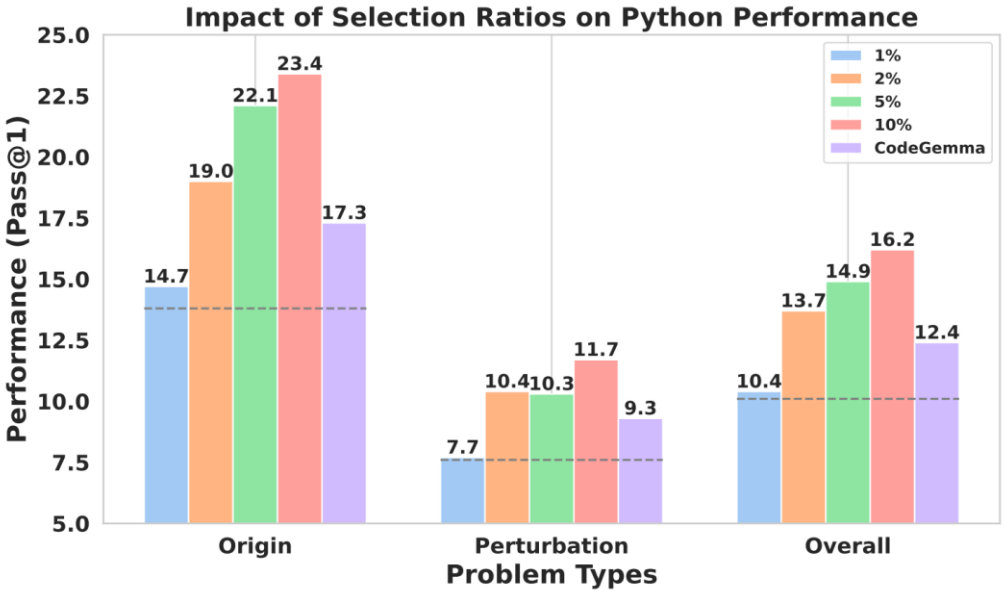
- Random selection often degrades performance.
- FINDR selects on-target data that consistently boosts base models across domains.
- FINDR generally outperforms strong baselines by non-trivial margins.
- FINDR demonstrates superior robustness on “Difficult” examples.
- NL-targeted selectors do not necessarily excel at NL2Code.

Further Study on Selection Ratio:



Gemma results in Python with varying selection ratios.
Dashed lines denote the off-the-shelf Gemma results.

Further Study on Selection Ratio:



Gemma results in Python with varying selection ratios.
Dashed lines denote the off-the-shelf Gemma results.

- Continuously trained models exceed the base model once the ratio reaches 2%.
- Performance generally rises further beyond 2%, albeit with diminishing return.
- Gemma trained on 2% FINDR-selected data surpasses CodeGemma despite its extensive pretraining.
- Similar trend observed in the SQL domain.

Thanks!

Contact: xlfzhang@umich.edu