Diverse Code Generation with Large Language Models

Zhenbang Feng
jasonfen@usc.edu

Melody Gui
melodyg@usc.edu

David Haolong Lee
dhlee@usc.edu

Dhruv Tarsadiya
tarsadiy@usc.edu

Shirley Yu
yushirle@usc.edu

Abstract

Code generation has been an exciting and interesting application of large language models (LLM). While researchers have tried to increase the robustness and accuracy of the generated code, prior work in quality diversity (QD) optimization suggests that structurally and semantically diverse programs may obtain better performance in downstream code generation tasks. To this end, we combine insight from prompt optimization and reinforcement learning (RL) to search for human-readable prompts that elicit diverse and high-accuracy programs from an LLM. Our empirical experiments indicate that our method empowers LLM to generate more diverse code that achieves higher accuracy than sequentially sampling an LLM. Our work sheds insight into the role of diversity in high-quality LLM generation.

1 Introduction

As computer science students, one of the most exciting applications of large language models (LLM) is code generation. LLM has impressive code generation capabilities and can solve many programming problems (Jiang et al., 2024; Rozière et al., 2024). A recent study demonstrates that repeatedly prompting (with the same prompt) a high-temperature LLM increases the number of programming problems that the LLM can solve (Brown et al., 2024). This is because, while the prompt remains the same, the high-temperature parameter induces slight variations in the code generated by the LLM. *The key insight to our project is that stronger variations (or increased diversity) among the generated code can improve the LLM's capacity to solve programming problems.*

This insight is echoed in quality diversity optimization (QD), where the goal is to produce a set of *high-performing* solutions that are also *diverse*. QD algorithms are powerful in domains where sub-optimal solutions act as "stepping stones" that mitigate convergence to local optima (Lee et al., 2024). A classic example is the problem of training an agent to reach a target position in a deceptive maze (Lehman and Stanley, 2011). There, directly minimizing the distance between the agent's final position and a target goal causes the agent to get stuck. On the other hand, the problem can be solved by ignoring the objective of directly reaching the target and instead attempting to find a diverse range of agents, each of which reaches a different region of the maze.

We utilize the exploration power of QD algorithms to empower LLM to generate diverse solutions to coding problems. However, we encountered difficulties in utilizing QD algorithms for prompt optimization as the dimensionality of the embeddings is too high. Thus, we considered reinforcement learning (RL) for learning prompts. RL works well in non-differentiable settings and can handle complex spaces through trial and feedback. We treat the prompt as a parameterized action space and learn how to adjust it to improve accuracy and diversity through Proximal Policy Optimization (PPO) (Schulman et al., 2017). PPO can find prompts that produce more accurate and diverse code solutions by repeatedly evaluating the rewards and using them to guide its next actions.

A problem with optimizing the continuous prompt embeddings is the deviation from natural language. This means that the LLM has to be queried in a way that bypasses the encoding layer, limiting our options to open-weight LLMs. However, performing discrete optimization on the vocabulary of the LLM is extremely difficult (Wen et al., 2024). To address this issue, we leverage the work of (Wen et al., 2024), which enables us to perform continuous optimization on the embeddings while maintaining the readability of the prompt.

We report our experimental results on two baseline algorithms, batched and sequential LLM sampling. Sequential sampling exhibits superior performance over batched sampling on all recorded metrics, including accuracy and diversity metrics (Fig. 4). We show that our RL methods induce better perturbations than the baselines and outperform them in code accuracy and diversity. We provide the code for our algorithm and experiment at https://github.com/LLM-QD.

2 Related Work

We describe the related work on prompt optimization and the code similarity metric CodeBLEU.

2.1 Quality Diversity Optimization

QD problems assume an objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and a vector-valued feature function $\boldsymbol{\phi} : \mathbb{R}^n \rightarrow \mathbb{R}^m$. The goal is to find a set of solutions S that collectively maximize the objective function $f(\cdot)$ and whose features $\boldsymbol{\phi}(\cdot)$ are sufficiently diverse. Formally, for solution $\boldsymbol{\theta} \in \mathbb{R}^n$, QD assumes an *objective* function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and m *feature* functions, jointly represented as a vector-valued function $\boldsymbol{\phi} : \mathbb{R}^n \rightarrow \mathbb{R}^m$.

Various methods address the QD problem by relaxing it finding an *archive* (i.e., a set) $\mathcal{A}$ of representative solutions. A nominal work utilizing this insight is **Covariance Matrix Adaptation MAP-Annealing (CMA-MAE)** (Fontaine and Nikolaidis, 2023), a state-of-the-art QD algorithm that has shown superior performance on many QD domains. The key idea of CMA-MAE is to optimize archive improvement with Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen, 2016). CMA-ES is a state-of-the-art derivative-free optimizer that maintains a population of solutions represented by a multivariate Gaussian $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

During each iteration, CMA-MAE draws λ solutions from CMA-ES. Based on the *rankings* (rather than the raw objectives) of the solutions, CMA-ES adapts the covariance matrix $\boldsymbol{\Sigma}$ to regions of higher-performing solutions. The solutions are ranked based on how much they improve the archive, e.g., solutions that found new cells in the archive are ranked high, while those that were not added at all are ranked low. The ranking is further annealed by an *archive learning rate* α. As the learning rate increases, CMA-MAE will focus more on solutions that explore the feature space over solutions that optimize the objective function. With this ranking, CMA-ES adapts the Gaussian to sample solutions that will further improve the archive.

2.2 Prompt Optimization

Prior works (Brown et al., 2020) have shown that prompting or instruction tuning is a powerful tool to improve the task-adaptive capabilities of pre-trained language models. Programmatically generating suitable task-specific text prompts through optimization in the continuous embedding space faces issues of interpretability and portability across various models (Khashabi et al., 2022). To overcome these issues, we shall utilize intuitions from Wen et al. (2023) to optimize in the discrete token space.

2.3 CodeBLEU

CodeBLEU (Ren et al., 2021) is an evaluation metric originally devised for code synthesis. It's based on comparing the synthesized code with the ground-truth code. CodeBLEU absorbs the n-gram matching algorithm used by its predecessor, BLEU (Papineni et al., 2002), and further compares the abstract syntax trees (AST) and data flow of the two programs to capture structural and semantic similarities. To this end, the CodeBLEU score is expressed as a weighted sum of four components as shown in Eq. 1.

$$\begin{aligned}\mathrm{CodeBLEU} &= \alpha \cdot \mathrm{BLEU} + \beta \cdot \mathrm{BLEU}_{\mathrm{weight}} \\ &+ \gamma \cdot \mathrm{Match}_{\mathrm{ast}} + \delta \cdot \mathrm{Match}_{\mathrm{df}}\end{aligned} \quad (1)$$

where $\mathrm{Match}_{\mathrm{ast}}$ is the syntactic AST match and $\mathrm{Match}_{\mathrm{df}}$ is the semantic data-flow match.

$\mathrm{Match}_{\mathrm{ast}}$ counts the number of matching subtrees between the AST of the two programs, denoted T_A and T_B (Eq. 2).

$$\mathrm{Match}_{\mathrm{ast}} = \frac{\mathrm{Match}(T_A, T_B)}{\mathrm{Count}(T_B)} \quad (2)$$

where $\mathrm{Match}(T_A, T_B)$ is the number of subtrees in T_A that matches with subtrees in T_B and $\mathrm{Count}(T_A)$ is the total number of subtrees in T_A.

Similarly, $\mathrm{Match}_{\mathrm{df}}$ measures the number of matching subgraphs between the data flow graphs of the two programs, denoted DF_A and DF_B, with normalized variable names (Ren et al., 2021).

$$\mathrm{Match}_{\mathrm{df}} = \frac{\mathrm{Match}(\mathrm{DF}_A, \mathrm{DF}_B)}{\mathrm{Count}(\mathrm{DF}_B)} \quad (3)$$

where $\mathrm{Match}(\mathrm{DF}_A, \mathrm{DF}_B)$ is the number of matched data-flows between the two programs and $\mathrm{Count}(\mathrm{DF}_B)$ is the total number of data-flows in program B.

For our experiments, we are interested in utilizing $\mathrm{Match}_{\mathrm{ast}}$ and $\mathrm{Match}_{\mathrm{df}}$ to capture the structural and semantic diversity of our code, respectively. More details are provided in Sec. 4.1.

3 Research Questions

In this section, we describe the research questions of this project. A central goal of this project is to build an algorithm based on LLM and QD that generates high-performing and diverse solutions to coding problems.

R1: *Can LLMs be combined with Prompt Optimization to generate high-performing and diverse solutions to coding problems?*

Furthermore, we want to motivate our algorithm by showing that increased diversity correlates with higher-quality code. This is supported by literature in QD, where searching for diversity helps mitigate local optima (Lehman and Stanley, 2011).

R2: *Does diverse code generation enable LLMs to solve more coding problems?*

4 Metrics and Evaluation

To answer **R1** and **R2**, we measure the diversity and functional correctness of the generated programs for each algorithm.

4.1 Structural and Semantic Diversity

The definition of diversity is an important anchor in this project. How do we quantify diversity in a set of programs? Programs can exhibit structural and semantical differences, capturing the difference in program syntax and program logic, respectively. To this end, we say that two programs are **structurally diverse** if their ASTs are different and that they are **semantically diverse** if their underlying logic is different.

We utilize the CodeBLEU (Ren et al., 2021) library to measure the semantic and structural diversity between a pair of code solutions. We define the structural diversity and semantic diversity of two programs as the inverse of $\mathrm{Match}_{\mathrm{ast}}$ and $\mathrm{Match}_{\mathrm{df}}$, respectively.

$$\mathrm{StructDiv} = 1 - \mathrm{Match}_{\mathrm{ast}} \tag{4}$$

$$\mathrm{SemDiv} = 1 - \mathrm{Match}_{\mathrm{df}} \tag{5}$$

4.2 Average and Maximum Accuracy

The accuracy of each generated program is the percentage of passed test cases. While average accuracy represents the overall quality of the generated solutions, in code generation settings we may only care about the most accurate solution, so we also note the maximum accuracy of the generated programs.

5 Experiment Settings

We will compare our algorithm, to the baseline **sequential LLM sampling**, using the metrics and evaluation described in Sec. 4. Each algorithm is expected to output $m = 4$ Python programs for each programming problem.

5.1 Dataset

Following prior work (Brown et al., 2024), we will use the CodeContests dataset (Li et al., 2022), a competitive programming benchmark. It consists of programming problems with text descriptions and test cases to evaluate our generated codes. CodeContests consists of a versatile set of problems collected from competitive programming websites such as Aizu, AtCoder, CodeChef, Codeforces, and HackerEarth.

For our experiments, we will only evaluate our models on Codeforces[1] dataset in the test set of CodeContests. Codeforces problems include more precise difficulty scores and include more test cases (Li et al., 2022). We removed all problems that were missing test cases and evaluated our algorithms on the first 50 Codeforces problems with non-empty test cases.

5.2 Base LLM Model

We leverage Meta's instruction-tuned Llama 3 model with 8 billion parameters (AI@Meta, 2024) as our base model.

5.3 Baseline

We evaluated the following two baselines and compared our proposed algorithm with the stronger one.

For **batched LLM sampling**, we prompt the LLM to generate $m = 4$ distinct solutions in one single response for each problem. For **sequential LLM sampling**, we prompt it m times with the same prompt used by Brown et al. (2024). The exact prompts are included in Appendix A. For both baselines, we set the parameters `temperature` to 0.6 and `max_tokens` to 4096.

Fig. 4 shows the average and maximum passing rate and the structural and semantic diversity for each problem. The metrics are averaged over ten trials with 50 problems per trial, and the bars are

[1] https://codeforces.com

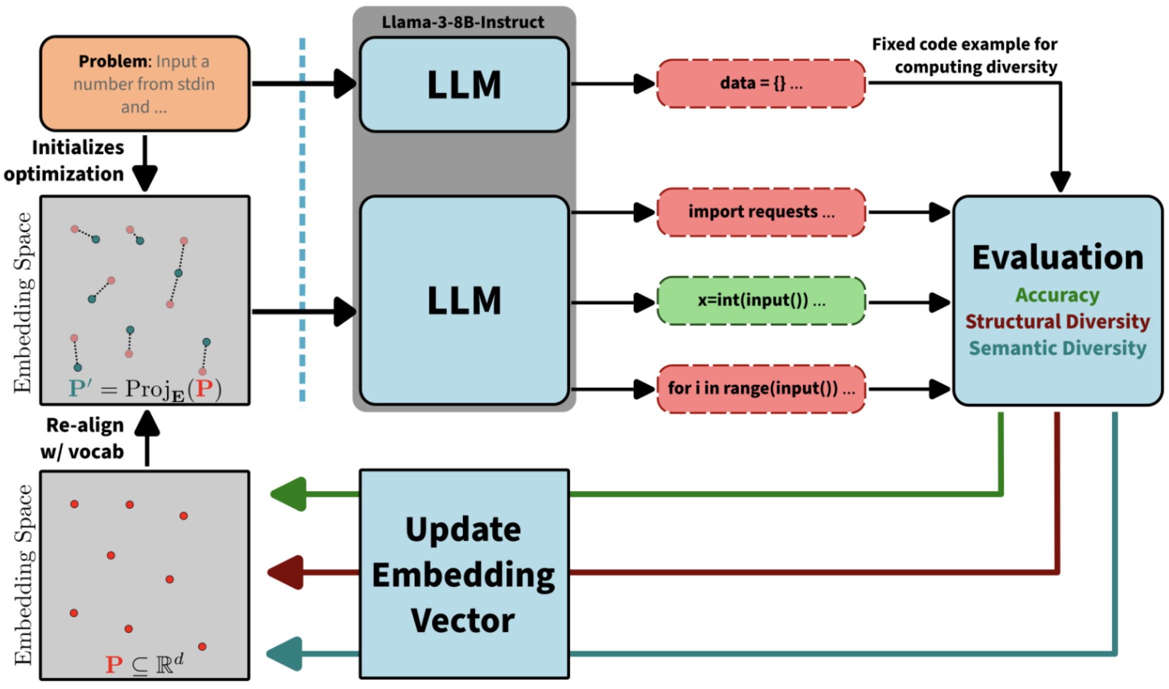

Figure 1: Framework of our proposed method: Prompt learned through repeated iteration of optimization and projection

sorted based on the models' performance. Additionally, the mean for each metric is indicated with a horizontal red line.

Our preliminary results demonstrate that sequential sampling slightly outperforms batched sampling in terms of both accuracy and diversity. Compared to batched sampling, sequential sampling achieves a higher mean and a higher maximum on every metric. Both methods achieved low test case accuracy but demonstrated decent diversification capabilities, achieving structural and semantic diversity around 50%. Henceforth, we use sequential sampling as our baseline for comparison with our proposed method.

6 Proposed Methods

We propose a modular framework, Large Language Model with Quality and Diversity (LLMQD), that optimizes the input prompt in discrete token space to increase diversity and accuracy, which is illustrated in Fig. 1. LLMQD utilizes an RL or QD algorithm to update the prompt's embedding vector based on diversity and accuracy. Inspired by Wen et al. (2023), LLMQD re-aligns the embedding to the vocabulary with a nearest neighbors projection function. This way, we maintain interpretable natural language prompts after every update to the embedding.

A central challenge is that the metrics described in Sec. 4 are non-differentiable, and thus cannot be directly optimized. Thus, we leverage derivative-free optimizers like PPO from RL and CMA-ES from QD. The optimizer interacts with an evaluation function that measures how well a particular APrompt leads to solutions that pass test cases and exhibit diversity.

6.1 Alignment with Vocabulary

To create a hard prompt from continuous embeddings, we utilize the work of Wen et al. (2024) to project each token to its nearest vocabulary token in the embedding space. This is done by computing the dot products between the candidate embedding and the vocabulary embeddings and searching for the nearest neighbor. This nearest-neighbor projection ensures that the prompt remains within the model's vocabulary, allowing us to prompt the LLM directly, without bypassing the embedding layer.

6.2 Optimizing with CMA-MAE

We experiment with leveraging CMA-MAE from QD to elicit diverse output from LLM. As illustrated in Fig. 2, CMA-MAE samples new embeddings from CMA-ES, records those prompts in an

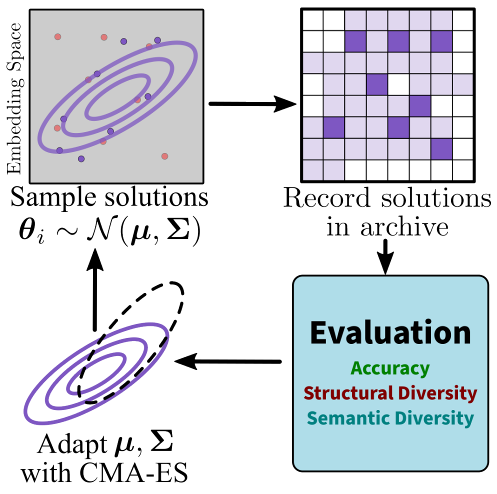
Embedding Space
Sample solutions $\theta_i \sim \mathcal{N}(\mu, \Sigma)$
Record solutions in archive
Evaluation
Accuracy
Structural Diversity
Semantic Diversity
Adapt μ, Σ with CMA-ES

Figure 2: An algorithm that leverages CMA-MAE to search for prompts that elicit diverse outputs from LLMs.

archive, and evaluates the accuracy and diversity of the code generated by those prompts. Importantly, all the embeddings generated by CMA-ES are projected to tokens in the vocabulary. Then, CMA-ES adapts the multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ that it generates samples from.

We encounter memory constraints when implementing this algorithm with prompt optimization. CMA-ES maintains a covariance matrix $\boldsymbol{\Sigma}$ that has quadratic space complexity in terms of the dimensionality of the samples. Since we are sampling embeddings of a prompt, the dimension of the samples is a product of the embedding dimension and the length of the prompt. This far exceeded the memory capacity of the machines we were using and we were unable to conduct experiments on this method.

We propose several ideas to address this limitation in discussion.

6.3 Optimizing with PPO

We leveraged PPO to iteratively augment k additional tokens, denoted as `APrompt`, that are appended to the original prompt. The `APrompt` is initialized with random embeddings and alters the output of the LLM in subtle ways during optimization. After each iteration, the `APrompt` is re-aligned with the vocabulary by projecting each token to its nearest token in the vocabulary in the embedding space.

We define the **state space** (s) as the embedding space, the action space (a) consists of continuous perturbations on the embeddings, and the reward function (r) is a weighted combination of average accuracy and structural and semantic diversity evaluated on the code generated by the LLM with the updated prompt (Eq. 6).

$$r = \alpha \cdot \text{Acc} + (1-\alpha)\left(\frac{\text{StructDiv} + \text{SemDiv}}{2}\right) \tag{6}$$

We choose $\alpha = \frac{2}{3}$ to focus more on optimizing the accuracy scores.

PPO learns a policy $\pi(\mathbf{a}|\mathbf{s})$ that maps states s to actions a. In this setting, the policy is parameterized by a neural network that takes the flattened embeddings of the APrompt as input and outputs a continuous action distribution. We use MLP-Policy, a multilayer perceptron that processes continuous inputs and outputs continuous actions. As the training progresses, PPO refines the policy to produce perturbations that move the APrompt embeddings toward configurations yielding higher rewards. This process optimizes the APrompt embeddings and leads the base LLM to generate code solutions with better accuracy and diversity.

7 Results

We apply the sequential sampling and RL approach to programming problems from the CodeContests dataset. For each problem, both methods generate $m = 4$ code solutions, on which we evaluate the mean accuracy, maximum accuracy, mean structural diversity, and mean semantic diversity. The metrics are recorded in Table 1.

Analyzing the box plot on Fig. 3, we observe an improved performance across both diversity and accuracy metrics for our proposed method compared to the baseline. The improvement in mean accuracy shows that the programs generated by our method passed more test cases on average. There is also

	Baseline	Our (RL)
Mean Accuracy	0.060	0.139
Max Accuracy	0.092	0.200
Mean Structural Div.	0.207	0.417
Mean Semantic Div.	0.228	0.458

Table 1: Average results across problems

a significant jump in max accuracy, which implies that the number of test cases passed by the best-performing programs was higher for our method. Additionally, our method produces programs that are more structurally and semantically diverse from each other than the programs generated by the baseline.

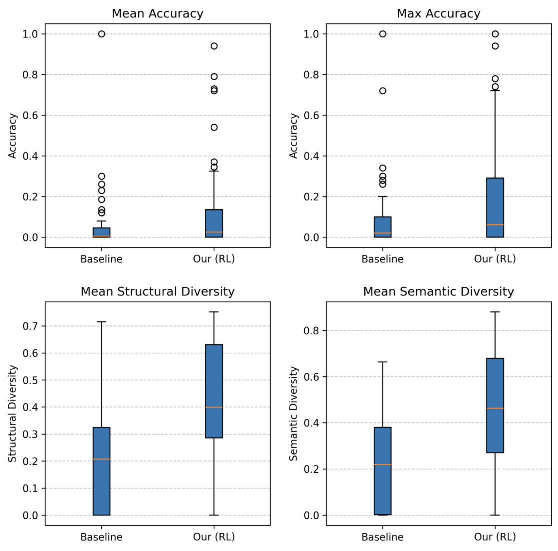

Figure 3: Box plots comparing sequential sampling and our (RL) method on mean accuracy, max accuracy, mean structural diversity, and mean semantic diversity

8 Conclusion

We leverage insight from QD to introduce a novel method that elicits diverse and high-quality code generation from LLM via prompt optimization. We experiment with our framework with QD and RL algorithms, achieving improved performance with PPO.

Although our initial attempt with QD failed, we can try to address the limitations by having QD optimize on a single token of the prompt instead of the entire prompt. Moreover, we can combine our RL method with QD by using an algorithm that adapts PPO to QD in RL domains (Batra et al., 2023).

We hope to extend our work to more general domains beyond code generation. Our work sheds insight into how eliciting diverse behaviors from LLM may improve performance. This may be applied to more human-LLM collaboration settings, where the expectations of the human are often ambiguous and hard to determine. In such cases, offering diverse options to the human may improve satisfaction and streamline collaboration. However, diversity might be difficult to quantify in these cases so we will require a robust metric based on human evaluation.

References

AI@Meta. 2024. Llama 3 model card.

Sumeet Batra, Bryon Tjanaka, Matthew C Fontaine, Aleksei Petrenko, Stefanos Nikolaidis, and Gaurav Sukhatme. 2023. Proximal policy gradient arborescence for quality diversity reinforcement learning. *arXiv preprint arXiv:2305.13795*.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *Preprint*, arXiv:2407.21787.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Matthew Fontaine and Stefanos Nikolaidis. 2023. Covariance matrix adaptation map-annealing. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 456–465.

Nikolaus Hansen. 2016. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*.

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

Daniel Khashabi, Xinxi Lyu, Sewon Min, Lianhui Qin, Kyle Richardson, Sean Welleck, Hannaneh Hajishirzi, Tushar Khot, Ashish Sabharwal, Sameer Singh, and Yejin Choi. 2022. Prompt waywardness: The curious case of discretized interpretation of continuous prompts. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3631–3643, Seattle, United States. Association for Computational Linguistics.

David H Lee, Anishalakshmi Palaparthi, Matthew C Fontaine, Bryon Tjanaka, and Stefanos Nikolaidis. 2024. Density descent for diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 674–682.

Joel Lehman and Kenneth O Stanley. 2011. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223.

Yujia Li, David 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. *Science*, 378(6624):1092–1097.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2021. Codebleu: a method for automatic evaluation of code synthesis. In *Proceedings of the 2021 AAAI Conference on Artificial Intelligence*, pages 716–724.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

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

Yuxin Wen, Neel Jain, John Kirchenbauer, Micah Goldblum, Jonas Geiping, and Tom Goldstein. 2023. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. *Preprint*, arXiv:2302.03668.

Yuxin Wen, Neel Jain, John Kirchenbauer, Micah Goldblum, Jonas Geiping, and Tom Goldstein. 2024. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. *Advances in Neural Information Processing Systems*, 36.

A Input Prompts

For our experiments, we use the following prompt from Brown et al. (2024):

```
Q: Write python code to solve the following coding problem that obeys the constraints and passes the example test cases. The output code needs to read from and write to standard IO. Please wrap your code answer using ```: [problem description]
```

B Baseline Results

We illustrate our baseline comparison results in Fig. 4.

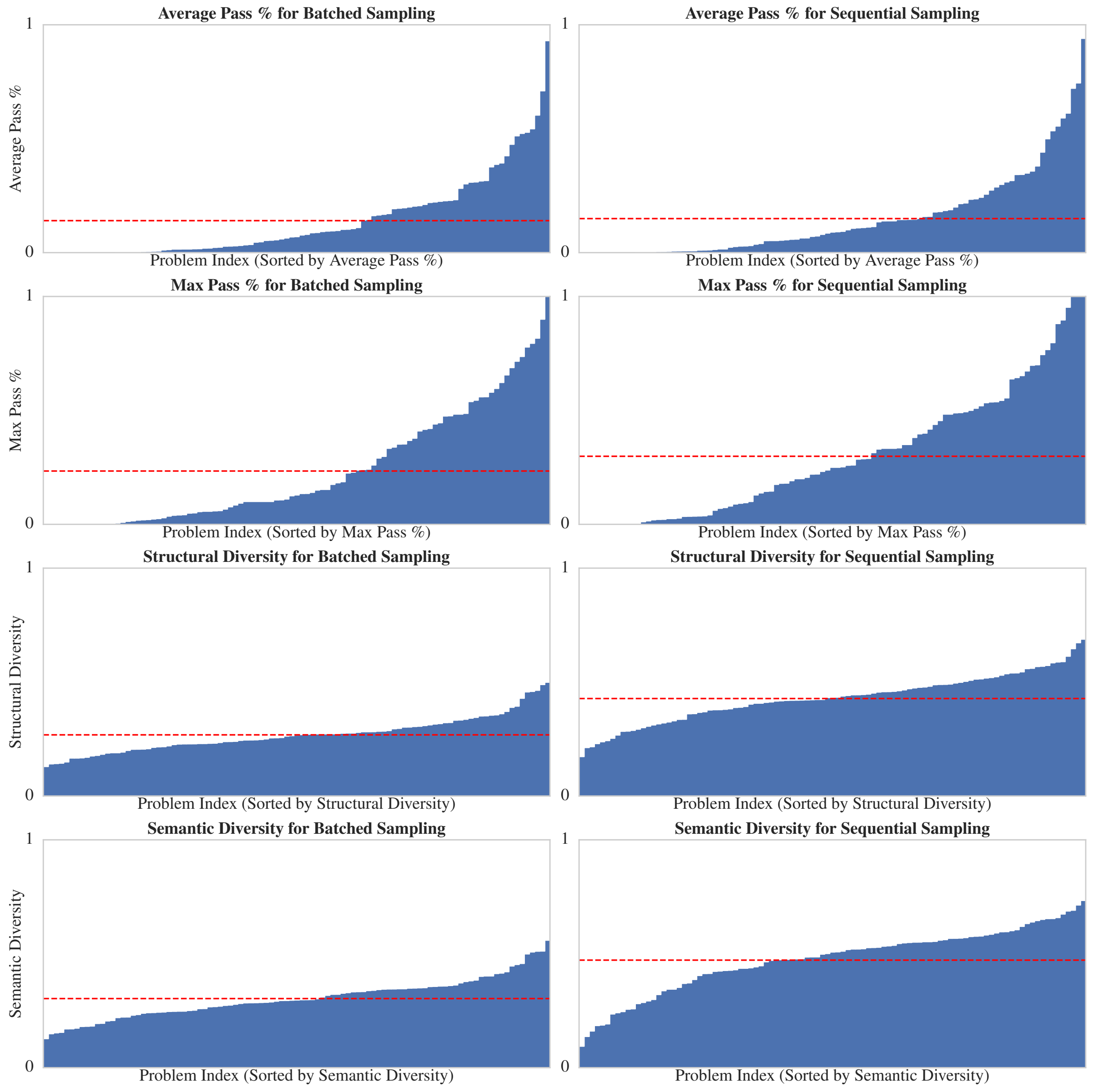

Figure 4: Accuracy and Diversity metrics of batched sampling and sequential sampling. The x-axes are problem index sorted by the performance in each metric, resulting in a non-decreasing bar plot. The red line indicates the mean performance for each metric.