

Investigating the Impact of Persona-Based System Prompts During SFT of Code LLMs

Stanford CS224N {Custom} Project

Tushar Aggarwal

Institute For Computational and Mathematical Engineering
Stanford University
tushar53@stanford.edu

Chenglei Sei (Mentor)

Department of Computer Science
Stanford University
clsi@stanford.edu

Abstract

System prompts are traditionally treated as critical anchors for guiding Large Language Model (LLM) behavior. While recent evidence suggests that persona-based prompting often fails to improve performance during zero-shot inference [1], the role of these prompts during Supervised Fine-Tuning (SFT) has remained under-explored. This paper investigates whether persona-conditioned SFT provides a superior steering mechanism compared to standard instruction tuning. We fine-tune the *deepseek-coder-6.7b-base* model [2] on the InstructCoder dataset [3] using a range of system prompts, from expert personas to arbitrary distractors. Our results show that while SFT improves Pass@1 on HumanEval from 49.4% to 69.3% [4], the model achieves near-identical performance regardless of the persona quality. We conclude that models learn to prioritize task-specific logic over system prompts during SFT, exhibiting significant cross-persona generalization.

1 Key Information to include

- Mentor: Chenglei Sei
- External Collaborators (if you have any): No
- Sharing project: No
- Late Days (if you are using any): No

2 Introduction

The primary interface for interacting with LLMs is the prompt, and within that interface, assigning a "persona" to the system is a ubiquitous strategy used to steer model behavior. From "expert programmers" to "helpful assistants," these roles are designed to contextualize the model's task. However, the true efficacy of these personas remains a subject of intense debate. This project addresses a fundamental problem in LLM alignment: do system-level personas actually enhance the capabilities of a model, or are they merely stylistic veneers that the model learns to ignore?

This problem is particularly important and difficult because of the black-box nature of fine-tuning. While we know that Supervised Fine-Tuning (SFT) significantly improves instruction-following, it is unclear if the model internalizes a persona as a functional expert mode or if it simply maps the input-output pairs of the training data. If personas are effectively disregarded by the model, then the extensive effort spent on prompt engineering during the SFT phase—crafting high-quality, task-specific roles may be a redundant and inefficient use of human and computational resources.

Current methods largely rely on zero-shot persona prompting during inference. However, recent empirical research [1] has demonstrated that these inference-time personas often fail to yield significant

gains in objective performance benchmarks. However, existing methodologies have not sufficiently explored whether "baking" a persona into the model weights via SFT provides a more robust steering mechanism than zero-shot prompting alone.

In this work, we investigate the impact of persona-conditioned SFT using the *deepseek-coder-6.7b-base* model [2]. Our approach involves fine-tuning the model on 108K code-editing samples while varying the quality and nature of the system prompts—ranging from carefully crafted expert roles to arbitrary and nonsensical distractors. Our key insight is the discovery of "persona agnosticism": while SFT provides a massive performance boost (increasing HumanEval Pass@1 scores [4] from 49.4% to 69.3%), the specific choice of persona has negligible impact. We find that the model learns the underlying task mechanics so effectively that it becomes invariant to the system prompt, even demonstrating high performance when presented with contradictory personas during inference. These results suggest a simplified path for future SFT, prioritizing raw data quality over complex persona engineering.

3 Related Work

Persona Prompting and System Roles: The practice of assigning roles to LLMs (e.g., "You are an expert") is widespread, yet its empirical basis is contested. The primary inspiration for this study is the work of [1], which conducted a large-scale evaluation of system prompts during zero-shot inference. They found that standard personas often fail to improve and can sometimes degrade model performance. Our work extends this inquiry from the inference phase to the training phase, investigating whether weight-level updates during SFT can succeed where zero-shot prompting fails.

Code LLMs and Instruction Tuning: The development of code-specific models has seen rapid advancement through models like DeepSeek-Coder [2], NextCoder [5] and QwenCoder [6]. These models often leverage specialized instruction-tuning techniques. For instance, WizardCoder [7] utilized the Evol-Instruct method to generate complex synthetic data, while InstructCoder [3] introduced a dedicated dataset for code-editing tasks. While these papers focus on the quality and complexity of the *instructions* and *outputs*, they typically treat the *system prompt* as a static, optimized constant. Our research addresses this gap by treating the system prompt as a variable to be analyzed.

Limitations of Current Literature: Existing literature effectively benchmarks how data scaling and instruction complexity improve Pass@1 scores on benchmarks like HumanEval [4]. However, there is a lack of promising analysis regarding "persona invariance", the degree to which a fine-tuned model becomes robust to (or ignores) its system-level framing. By replicating SFT across diverse persona conditions, we aim to demonstrate that the instruction-following signal in code-editing datasets is powerful enough to make the complex persona engineering unnecessary.

4 Approach

Our approach investigates the sensitivity of the fine-tuning process to the semantic content of system prompts. We formalize the SFT task as a conditional sequence generation problem where the model must learn to disregard varying system-level noise in favor of task-specific execution.

4.1 Model Architecture and Task Formalization

We utilize the *deepseek-coder-6.7b-base* model [2], a Transformer-based decoder-only model. Given a dataset $D = \{(P_{sys}, X_{instr}, Y_{code})\}$, where P_{sys} is the system prompt (persona), X_{instr} is the user instruction, and Y_{code} is the target Python code, the model is trained to minimize the standard language modeling objective:

$$\mathcal{L}(\theta) = - \sum_{t=1}^{|Y|} \log P_{\theta}(y_t | P_{sys}, X_{instr}, y_{<t}) \quad (1)$$

In this setup, we vary P_{sys} across three experimental conditions:

1. **Expert Persona:** A carefully crafted prompt defining the model as a world-class AI programmer.

2. **Arbitrary Persona:** A prompt defining a role completely unrelated to coding (e.g., "You are a donkey").
3. **Contradictory Persona:** A prompt that instructs the model to perform poorly or provide buggy code.

4.2 The Fine-Tuning Pipeline

The fine-tuning script was built using the official WizardLM fine-tuning script [7] as a reference. The custom data-collator, and persona-injection logic are our original implementations. For evaluation, we used the bigcode evaluation harness [8]. The custom data-collator, persona-injection logic, and the integration of vLLM [9] support for fast inference into the bigcode evaluation harness are our original implementations.

4.3 Baseline

Our primary baseline is the zero-shot performance of the *deepseek-coder-6.7b-base* model [2]. This baseline represents the model’s intrinsic capabilities without any task-specific alignment, providing a point of origin to measure the absolute performance gain attributable to our Supervised Fine-Tuning (SFT) process.

5 Experiments

This section details the empirical setup used to evaluate the impact of system prompts on code-editing performance, including dataset specifications, evaluation metrics, and training configurations.

5.1 Data

We utilize the **InstructCoder** dataset [3], which consists of 108K high-quality Python code-editing samples. Each sample is curated to reflect real-world programming tasks where a developer provides a natural language instruction to modify existing code. To precisely define the task, the model is presented with an input tuple (P_{sys}, X_{instr}) and must generate a code block Y_{code} .

- **System Prompt:** You are a {persona}, you help people in solving their questions.
- **User Instruction:** [Natural language instruction from InstructCoder]

SFT Data Point Example:

System: You are a **donkey**, you help people in solving their questions.
User: Modify the `process_data` function to include a try-except block for `ValueError`.
Output: [Executable Python code block]

5.2 Evaluation method

The primary evaluation metric is Pass@1 on the **HumanEval** benchmark [4]. HumanEval consists of 164 handwritten programming problems. For each problem, the model generates a single completion. The completion is considered passed only if it passes all associated unit tests. Pass@1 is defined as the fraction of problems solved.

To facilitate large-scale testing across multiple persona conditions, we utilized the **bigcode-evaluation-harness** [8]. As a technical extension, we integrated **vLLM** [9] to serve the model during evaluation, which increases the speed of inference by efficiently utilizing the resources.

5.3 Experimental details

To test the limits of persona invariance, we fine-tuned the model using 4 distinct personas. During the evaluation phase, we significantly expanded this scope to 15 diverse personas, ranging from domain experts to nonsensical roles like Donkey, to observe if the model’s task performance remains robust

across a wide stylistic spectrum. All experiments were conducted using the *deepseek-coder-6.7b-base* model [2]. Fine-tuning was performed over 2 epochs to ensure consistency across all the experiments and also avoid overfitting.

- **Optimizer:** AdamW [10] with weight decay.
- **Learning Rate:** 2×10^{-5} with a **WarmupLR** scheduler (0.05 warmup ratio).
- **Infrastructure:** Training was executed on NVIDIA A100 GPU.
- **Hyperparameters:** Batch size of 32 and a maximum sequence length of 4096 tokens.

5.4 Results

Figure 1 illustrates the training loss over the course of 2 epochs. We observed that the loss decreases steadily within each epoch, with a significant, sharp drop occurring at the transition between epoch 1 and epoch 2. The sudden drop in training loss at the end of the first epoch is primarily caused by the model’s second pass over the 100K samples in the InstructCoder dataset. Having already established a baseline for task-specific logic during the first epoch, the model uses the second pass to rapidly refine its weights and correct high-magnitude errors.

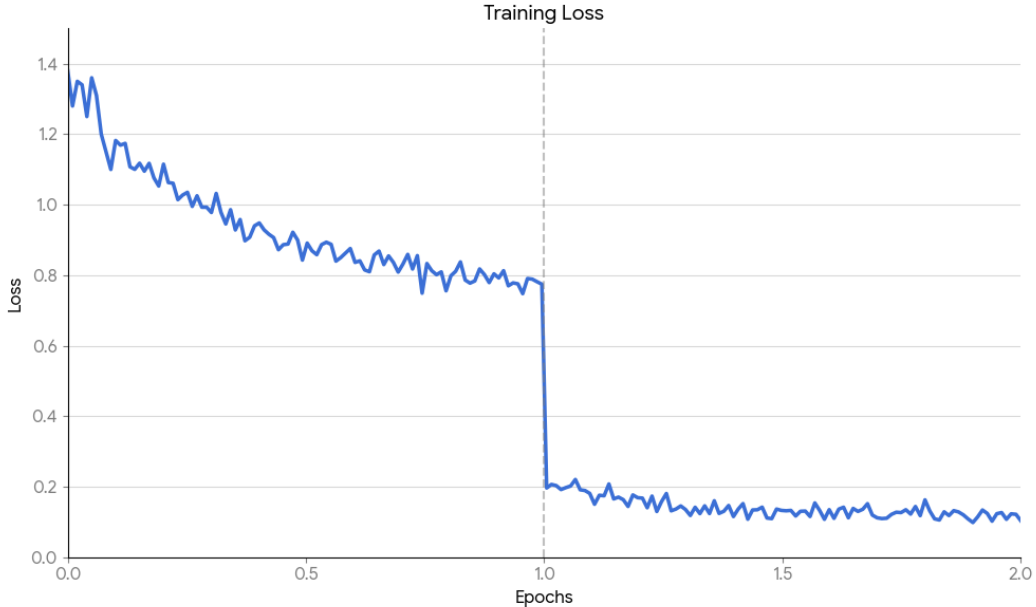


Figure 1: Training Loss for one of the SFT experiments.

The quantitative results of our experiments are summarized in Figure 2. Our fine-tuning pipeline achieved a 69.3% Pass@1 score on HumanEval, representing a substantial 19.9% absolute improvement over the base model’s zero-shot performance of 49.4%. This significant jump confirms that our SFT setup effectively aligns the model for code-editing tasks.

Commentary: These results were highly unexpected. While we anticipated a strong performance boost from SFT, we hypothesized that the quality of the system prompt would play a decisive role in the final accuracy. Instead, the results achieved with arbitrary and even nonsensical personas (e.g., "Donkey") were nearly identical to those achieved with the carefully crafted "Expert AI" persona.

This tells us that the model’s performance gain is driven almost entirely by the weight-level updates associated with the instruction-following task, rather than the framing of the persona. The model appears to be learning to disregard the system prompt during the fine-tuning process when a strong task-specific signal is present. This finding suggests that complex persona engineering may be an over-optimization in the context of supervised fine-tuning for functional tasks like coding.

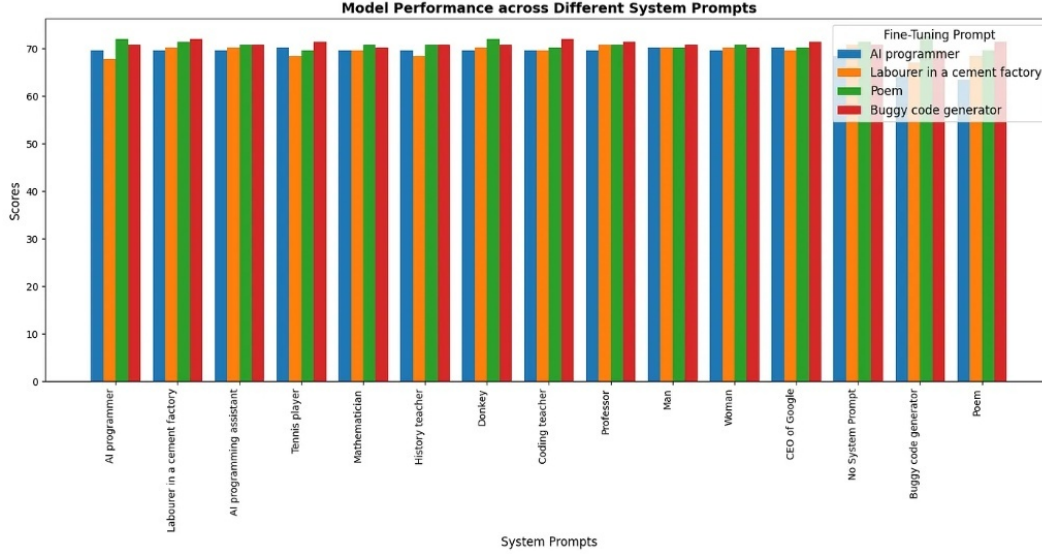


Figure 2: The x-axis shows the prompt that was used for inference and the y-axis are the scores on HumanEval.

6 Analysis

6.1 Qualitative Evaluation

To understand our system, we inspected the generated outputs when the model was prompted with diverse system personas. In Figure 1, we contrast the output of the model when prompted with an "Expert AI Programmer" persona versus a "Donkey" persona.

Despite the drastic semantic difference in the system prompt, the generated Python code is syntactically and logically identical. This suggests that the model's internal representation of the "code-editing" task, learned from the *InstructCoder* [3] dataset, creates a strong constraint that overrides the stylistic or persona-based conditioning of the system prompt. The model effectively treats the persona as background noise, focusing its weights on the functional requirement of mapping the user instruction to the correct code change.

6.2 Limitations

A critical limitation of this analysis is the scope of our model selection. This study was conducted exclusively on the *deepseek-coder-6.7b-base* model [2]. This is a relatively small, code-specific model. It is possible that larger, more generic models (e.g., 70B+ parameters) possess the capacity to balance both task-specific precision and persona-based style steering simultaneously. The observed persona-agnosticism may be a byproduct of the model's specialized training for code, where syntactic correctness is prioritized over semantic style.

Additionally, our findings suggest that a "Mixture of Prompts" strategy, where the model is trained on a diverse variety of personas simultaneously might force the model to generalize better across stylistic contexts, potentially preventing the persona-agnostic behavior we observed. By diversifying the training-time system prompts, we may be able to create a model that retains its coding capabilities while simultaneously becoming more responsive to persona-based steering during inference.

7 Conclusion

In this work, we investigated the role of system prompts during the Supervised Fine-Tuning (SFT) of Large Language Models. Our central achievement was the development and validation of an SFT pipeline that improved the HumanEval Pass@1 score of our baseline model from 49.4% to 69.3%. Our results provide strong evidence that persona engineering (the practice of meticulously crafting

system prompts) is not a primary driver of performance in the SFT regime. Instead, we found that the model internalizes the task-specific logic of the dataset so thoroughly that it becomes invariant to the system-level persona.

References

- [1] Mingqian Zheng, Jiaxin Pei, Lajanugen Logeswaran, Moontae Lee, and David Jurgens. When “a helpful assistant” is not really helpful: Personas in system prompts do not improve performances of large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15126–15154, Miami, Florida, USA, November 2024. Association for Computational Linguistics.
- [2] 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. Deepseek-coder: When the large language model meets programming – the rise of code intelligence, 2024.
- [3] Kaixin Li, Qisheng Hu, Xu Zhao, Hui Chen, Yuxi Xie, Tiedong Liu, Qizhe Xie, and Junxian He. Instructcoder: Instruction tuning large language models for code editing, 2024.
- [4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde 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, Josh 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. Evaluating large language models trained on code, 2021.
- [5] Tushar Aggarwal*, Swayam Singh*, Abhijeet Awasthi, Aditya Kanade, and Nagarajan Natarajan. Nextcoder: Robust adaptation of code lms to diverse code edits. In *ICML 2025*, July 2025.
- [6] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-coder technical report, 2024.
- [7] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations*, 2024.
- [8] Loubna Ben Allal, Niklas Muennighoff, Logesh Kumar Umapathi, Ben Lipkin, and Leandro von Werra. A framework for the evaluation of code generation models. <https://github.com/bigcode-project/bigcode-evaluation-harness>, 2022.
- [9] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.
- [10] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.