Extended Abstract

Motivation We focus on the task of generating helpful error messages for students taking an introductory Python class. In Code-In-Place, a large Stanford online CS education platform, students are given error message hints using GPT-4o via OpenAI's API when they run their program. Last month, over one million AI hints were generated by users in over 150 countries world wide. However, this is costly and restricts help to those with internet access. Our goal is to develop a small LLM that can produce these error message hints locally on a user's computer.

Method Our approach consists of two components: (1) fine-tuning the 1B parameter Llama 3 Instruct model using Direct Preference Optimization (DPO), and (2) pruning this model with a reinforcement learning (RL) agent while preserving performance. DPO uses pairs of responses (preferred vs. dispreferred) and adjusts the model to increase the log-likelihood of the preferred response, aligning output behavior with desired preferences. For pruning, we use an online advantage actor-critic RL agent that traverses the model one layer at a time. The agent receives a reward based on the pruning ratio and the KL divergence between the output distributions of the original and pruned models on prompts from Code-In-Place. The final pruned model is evaluated on 100 unseen prompts. Its responses are graded by GPT-4o on two criteria: whether it correctly identifies the error (yes/no, where "yes" is ideal) and whether it gives away the answer (yes/no, where "no" is ideal).

Implementation For DPO, we collected approximately 20,000 prompts (the student written code and the error message it yielded), along with the accompanying GPT-4o generated hints that the student was served. We generated alternative hints for each prompt using Llama 3.2 1B Instruct and used GPT-4.1 to create preference pairs given a custom rubric. We then ran DPO on Llama 3.2 1B Instruct to generate our new model. To prune this model, we first performed a sensitivity analysis to identify which components of each layer could be pruned with minimal performance degradation. The actor's state space includes the current layer and the pruning ratios of all prunable parts across layers. At each layer, the actor selects a pruning ratio from a discrete action space. After each layer's pruning, it receives a reward based on pruning efficiency; after the final layer, the reward is updated based on both overall pruning ratio and KL divergence from the original model.

Results We found that using Direct Preference Optimization to fine tune small LLMs lead to significant gains in model performance. In our evaluation, the unpruned DPO model improved in correctly identifying the problem in the code by $\approx 35\%$ on the base model and gave away the answer in the hint $\approx 17\%$ less than the base model. While performance does deteriorate with pruning, the DPO models consistently beat the baseline at every level, with some pruned models remaining competitive with the unpruned baseline. While the DPO model does make some gains, GPT-4o still beats all models on both correctness and not displaying code answers by a large margin.

Discussion While the pruning agent converged on final pruning ratios for all layers without collapsing to a trivial solution, the overall resulting pruned model was underwhelming. When averaged across 300 sample prompts, the performance of our agent-pruned model was in line with that of a benchmark model with the same overall compression ratio applied across all layers equally. In training, it was found that KL divergence is a noisy metric, and it appears challenging for the agent to accurately predict how pruning each layer will influence the terminal KL penalty. Finally, we determined that while DPO does improve performance, in this case, it is likely not effective enough alone to provide a deployable system.

Conclusion We have shown that a pruned, DPO'd model can achieve the same performance as an unaltered model on a specialized task. While the success rate is not high enough for commercial deployment, we believe other finetuning algorithms, better prompting, and larger models are all potential paths to improving the performance on this task. In future work, different (albeit higher cost) ways of measuring output prompt quality may be needed in order to achieve a final pruned model which is better, such as a simple boolean grading of whether or not the LLM output was correct and helpful.

BallPy: Bug Analyzing Local LLM for Python

TJ Jefferson
Department of Computer Science
Stanford University
tjj@stanford.edu

Eli LeChien
Department of Electrical Engineering
Stanford University
lechien@stanford.edu

Ben Pekarek
Department of Electrical Engineering
Stanford University
pekarek@stanford.edu

Abstract

We present a method for generating helpful Python error message hints using a small, locally run LLM, aiming to make educational tools more accessible and cost-effective. We fine-tune a 1B parameter Llama 3 Instruct model using Direct Preference Optimization (DPO) and apply reinforcement learning-based pruning to compress the model while preserving performance. Our DPO-tuned model significantly improves hint quality over the base model, with up to 35% higher correctness and 17% fewer answer leaks. Although pruning reduces performance, some pruned models remain competitive with the unpruned baseline. While not yet suitable for deployment, our results suggest promising directions for building lightweight, offline-capable educational LLMs.

1 Introduction

Large Language Models have become extremely valuable tools, helping users with a broad range of tasks. While frequent innovations and novel applications make this technology exciting, there is a barrier in enabling equitable access to users around the world. In most cases, persistent internet access is a requirement, as server based-inference is the current standard for commercial LLMs. Therefore, there is a need for models which can run locally on a user's own device, as once the model is downloaded, internet is not needed. This, however, presents additional challenges, as local models must be small enough to ensure a fast download and proper execution on a broad range of hardware. In this paper, we focus on one particular task, generating error message hints for beginner Python programmers, and attempt to build a model small enough to be run locally on a wide variety of computers.

1.1 Task

Code in Place is a free online course where over 50,000 students from over 180 countries learn introductory Python [Code in Place (2024); Malik et al. (2023)]. The course platform contains an integrated development environment (shown in Figure 1) where students can write code and solve challenges. When a given student's code has an error in it, the student receives a hint generated by GPT-4o in addition to the standard Python error message. These hints have been shown to have a positive effect on helping students across demographics, however, while much of the IDE is functional without internet, the hint generation is not. Our goal is to create a small LLM such that the error hints can be generated by students locally, avoiding the need for sustained internet access.

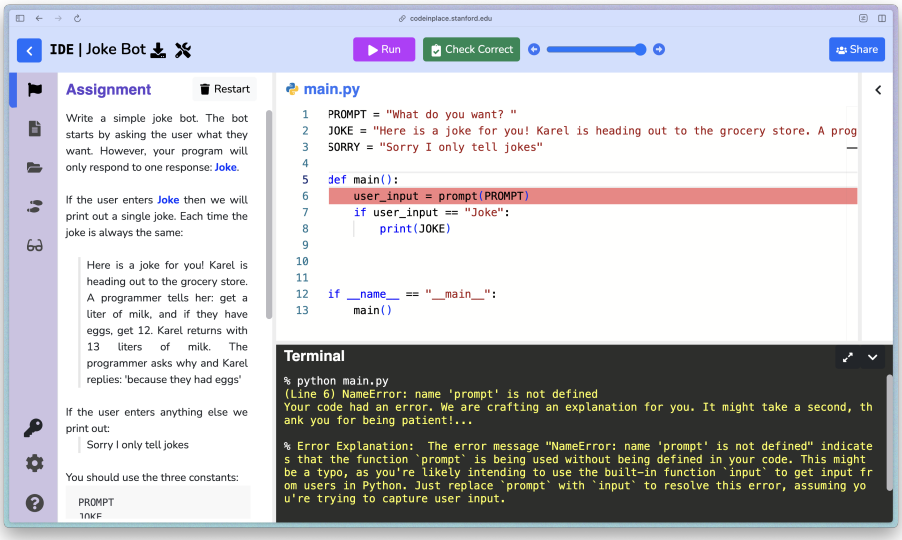

Figure 1: Code-In-Place Development Environment with Sample Error Message

1.2 Approach

We provide a two step approach to developing the model.

1. Fine tune the model using Direct Preference Optimization (DPO) to improve the quality of error hints.
2. Prune the model layer-by-layer with a strategy determined by a reinforcement learning agent.

We attempt to accomplish this using a dataset of approximately 20,000 data points containing student code, the resultant error, and the GPT-generated hints.

2 Related Work

Large Language Models such as those from the GPT and Llama families provide extremely high performance on many useful tasks. That said, smaller models are capable of matching such performance when tuned for a subset of tasks. Many researchers have developed efficient techniques for fine tuning smaller models to improve their performance at specific tasks [Dettmers et al. (2023); Ouyang et al. (2022); Rafailov et al. (2024)]. In this paper, we use one specific reinforcement learning based method put forth by Rafailov et al. (2023), Direct Preference Optimization, to finetune a 1B parameter LLM.

While fine tuning is a powerful tool, we believe that the size of the model can be reduced further. Many researchers have developed methods for compressing, or pruning machine learning models, including LLMs [Saha et al. (2024); DeepSeek-AI et al. (2025)]. In this paper, we will define the pruning task as a reinforcement learning problem. We model our approach based on prior work from He et al. (2018), who proposed an RL system to prune the layers of various neural networks, including VGG, ResNet, and MobileNet. While some researchers have attempted to use reinforcement learning with a limited action space to condense LLMs, there has not been an abundance of research in this area [Li et al. (2024)].

The task we have chosen to focus on is generating helpful error message hints for beginner programmers. In one randomized control trial, Wang et al. (2024) found that LLM generated hints can be helpful for learners from all backgrounds. This finding motivates our research, as decreasing the size of the language model without sacrificing performance could allow for local, less cost intensive deployment, which could benefit learners who might lack access to fast reliable internet.

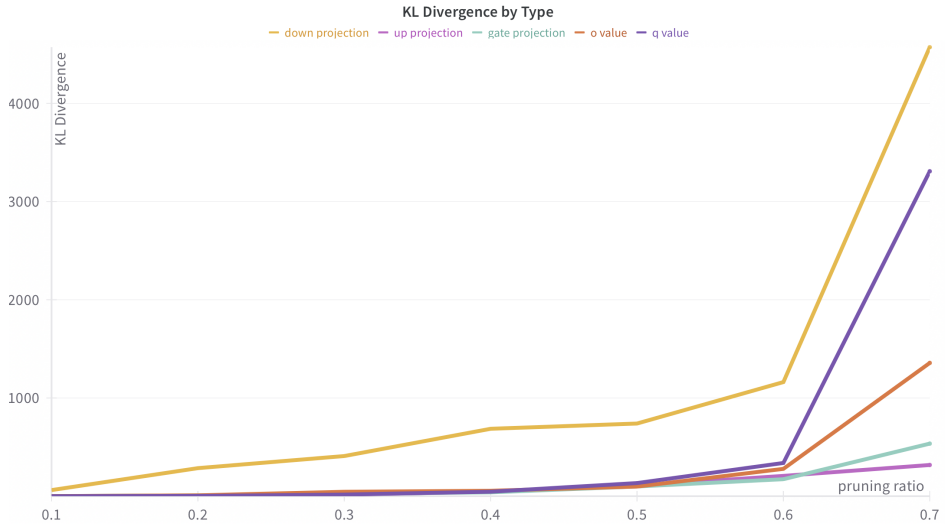

Figure 2: Sensitivity Analysis: KL Divergence when Pruning Different Layers

3 Method

3.1 DPO

We fine-tune our small LLM using Direct Preference Optimization (DPO), which requires pairs of responses labeled by preference. Although our dataset contains 20,000 examples of code, error messages, and AI-generated hints, it does not include the preference pairs needed for DPO training. To generate preference pairs on this scale, we used Llama 3.2 3B Instruct to generate alternative hints for each example, then GPT-4.1 as a judge to choose the preferred hint between the two options. GPT-4.1 was given a rubric emphasizing correctness of the hint (accurately identifying the error), the brevity and clarity of the hint, and the hint not giving away code, or the answer. In addition to generating these preference pairs, we also removed any prompts with student code over 2,500 characters for memory reasons. This amounted to less than 2% of the total dataset.

We ran DPO on a Llama 3.2 1B Instruct model, with 17,648 (90%) training examples and 1,961 test examples (10%). We also used a Low Rank Approximation (LoRA) adapter with a rank of 8, a dropout of 0.1 and an α of 16.

3.2 Pruning Agent

We cast layer-wise pruning as a finite-horizon Markov decision process (MDP) that is solved by an online Advantage Actor–Critic (A2C) agent. Starting from an unpruned *student* model θ_0, the agent visits the L transformer layers in order. At each layer $t \in \{1, \ldots, L\}$ it chooses a discrete pruning ratio for every prunable submodule (query, output, gate and up-projection) and immediately applies magnitude-based unstructured pruning. The episode therefore consists of exactly L steps and terminates with a fully pruned model θ_L. Policy and value networks are lightweight two-layer MLPs with LayerNorm, trained concurrently with the environment.

3.2.1 Defining State and Action Space

State The state vector $s_t \in \mathbb{R}^{4L+1}$ concatenates (i) the *fraction of weights that remain* in every prunable part of *all* layers and (ii) a scalar encoding of the current layer index t. This design allows the agent to reason both about local pruning history and the global budget that remains. The choice to include the query, output, gate, and up submodules in the state space, and thus only prune those parts, was made after performing a sensitivity analysis. In this analysis, one submodule is pruned at a time across all layers of the LLM. The KL divergence between the logit distributions of the base model and pruned model for a subset of prompts is recorded and used as a proxy metric for how well the output of the pruned model matches that of the base model. The pruning ratio and submodule

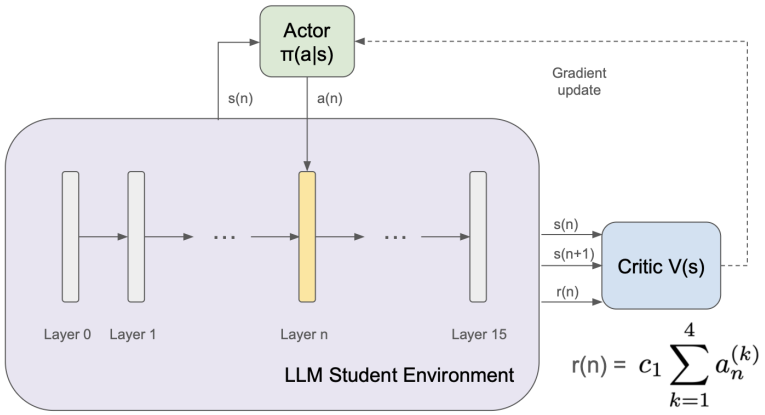

Figure 3: A2C Method Pruning Intermediate Layer n of the LLM

type are varied to see which submodules are most sensitive to unstructured pruning, which manifests as large KL divergence values when the base and pruned models have very different outputs. It can be seen in Figure 2 that pruning the down projection submodule led to much larger distribution shifts than the o, q, up, and gate submodules across all pruning ratios. The key and value submodules had similar behavior, thus why they were excluded from the state/action space.

Action At step t the action is a tuple $a_t = (a_t^{(1)}, \ldots, a_t^{(4)})$, where each component $a_t^{(k)}$ is selected from a discrete set of 31 pruning ratios $\{0.00, 0.01, \ldots, 0.30\}$. The action space therefore has cardinality 31^4. Choosing $a_t^{(k)} = \alpha$ removes the α-quantile (by magnitude) of weights from the corresponding submodule. We choose to use a discrete action space to simplify the pruning task for the agent and more easily facilitate exploration. We experimented with a continuous action space, but found it more difficult to tune hyper parameters for model convergence to non-trivial solutions.

3.2.2 Exploration

We combine two mechanisms to encourage the policy to explore, epsilon greedy random sampling and a policy entropy bonus. The epsilon greedy implementation is standard: with probability ε the agent samples each $a_t^{(k)}$ uniformly; ε decays exponentially from 1.0 to 0.001 with factor 0.999 per episode. The entropy bonus is expressed in the actor loss function with the addition of the $-\lambda_{\text{ent}} \cdot \mathcal{H}(\pi(\cdot \mid s_t))$ term with $\lambda_{\text{ent}} = 0.01$ to discourage premature collapse of the categorical distributions.

3.2.3 Loss Functions

Outside the actor loss entropy bonus, our implementation of the Advantage-Actor Critic method is standard. The critic functions as an estimator of the value function, and the actor operates with the goal of maximizing advantage $A_t = Q_t(s_t, a_t) - V(s_t)$. $Q_t(s_t, a_t)$ is approximated with the TD approximation as $R_t + \gamma V(s_{t+1})$ The actor parameters ϕ and critic parameters ψ are updated after every step using

$$\mathcal{L}_{\text{actor}} \;=\; -\big(\log \pi_\phi(a_t \mid s_t)\big)\, A_t \;-\; \lambda_{\text{ent}}\, \mathcal{H}(\pi_\phi(\cdot \mid s_t))\,, \quad \mathcal{L}_{\text{critic}} \;=\; \big(R_t + \gamma V_\psi(s_{t+1}) - V_\psi(s_t)\big)^2.$$

The advantage A_t uses the discount $\gamma = 0.99$ in the critic update. Gradients are clipped to and optimized with Adam ($\eta_{\text{actor}} = 2 \times 10^{-5}$, $\eta_{\text{critic}} = 1 \times 10^{-4}$).

3.2.4 Reward

We design a shaped reward that aligns with our competing objectives: **LLM size** and **accuracy**. We measure size based on the number of pruned parameters, and accuracy with the KL divergence between the original (teacher) model and the pruned (student) model on a batch of prompts from our

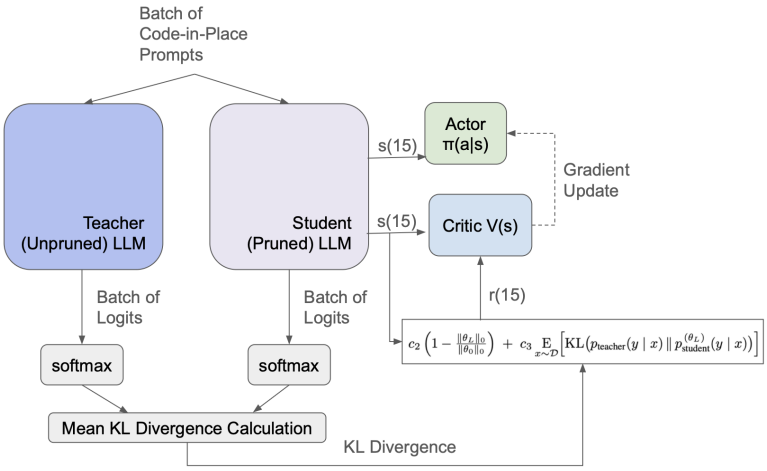

Figure 4: A2C Method Pruning Final Layer (layer 15) of the LLM.

dataset. For every non-terminal step $t < L$ we grant

$$r_t \;=\; c_1 \sum_{k=1}^{4} a_t^{(k)},$$

a reward proportional to the immediate pruning accomplished in that layer. An illustration of a single non-terminal step is shown in Figure 3. After the final layer our reward consists of two global terms:

$$r_L \;=\; c_2 \left(1 - \frac{\|\theta_L\|_0}{\|\theta_0\|_0}\right) \;+\; c_3 \underset{x\sim\mathcal{D}}{\mathrm{E}} \Big[\mathrm{KL}\big(p_{\text{teacher}}(y \mid x) \,\|\, p_{\text{student}}^{(\theta_L)}(y \mid x)\big)\Big],$$

where $\|\cdot\|_0$ counts remaining parameters and the KL term is averaged over a prompt batch. The motivation for only including the KL penalty on the final layer is that the partially pruned model quality is not of interest – only the final model quality matters for the sake of performance on the task. Maximizing r_L therefore encourages the agent to prune aggressively while minimizing overall behavioral drift from the original Llama-3 teacher. An illustration of the terminal step in an episode is shown in Figure 4.

4 Experimental Setup

To evaluate the models, we developed a validation set of 100 prompts (including student code and its corresponding standard error message) and generated hints for all prompts with each of the models we tested. We then used an LLM-as-a-judge system to rate each hint on four dimensions: accuracy of diagnosing the hint (yes/no), helpfulness (score 1-5), clarity (score 1-5), and whether or not the hint gave away the code (yes/no). The LLM (GPT-4.1) received the generated hints along with their corresponding prompts and a rubric specifying the above scoring information.

The RL algorithm described in the methods sections was implemented, with PyTorch being used for the A2C neural networks and Hugging Face's transformers library for the teacher and student LLMs. Built-in methods for conducting L1-unstructured pruning on the model and computing the KL penalty were leveraged to simplify the development process. The pruning agent was trained across 100 epochs through 1400 sample prompts, with the A2C update being conducted after a prompt batch size of 6 was evaluated. This training process was conducted on a NVIDIA A10G Tensor Core GPU. The actor actions for the final epoch were taken as the learned optimal pruning strategy for the prunable parts, and the final model converged on a nontrivial solution.

5 Results

5.0.1 DPO

Fine-tuning the base Llama 3.2 1B model with DPO resulted in a substantial performance improvement. The model's accuracy in identifying code errors increased by 35.7% (from 28% to 38%), while the frequency of hints that gave away the answer dropped by 16.7% (from 66% to 55%). While this is strong improvement compared to the base model, it still performs well below the model currently used for the task. GPT-4o was graded as accurately identifying the error 99% of the time while giving away code approximately 6% of the time. As previously stated, we also scored the models on "helpfulness" and "clarity". We found that both the base model and the DPO model scored similarly on overall helpfulness, averaging scores of 2.81 and 2.84 respectively with GPT-4o averaging 4.9 out of 5. On the evaluation of clarity, we found the base model averaged slightly higher than the DPO model (3.35 as opposed to 3.22) with GPT-4o scoring at 4.92 out of 5.

5.0.2 Pruning

Figure 5 shows the final pruning ratios for Llama-1B-instruct generated by the agent after training using 1400 Code-In-Place examples across 100 epochs. The average pruning ratio of the gate, up, output, and query projection layers are 18%, 17%, 22%, and 19% respectively with an overall compression ratio of 18% for the prunable parameters. This corresponds to pruning 120M parameters in total, which compresses the overall model by 10%. We see some interesting patterns in the pruning strategy. In the gate projection layer, the agent prunes more on the initial and final layers and less in the middle. In the up-projection layer, the agent prunes the more in the early and middle layers. The query and output projection layers appear to be pruned more uniformly, though there are still some layers which are not pruned much.

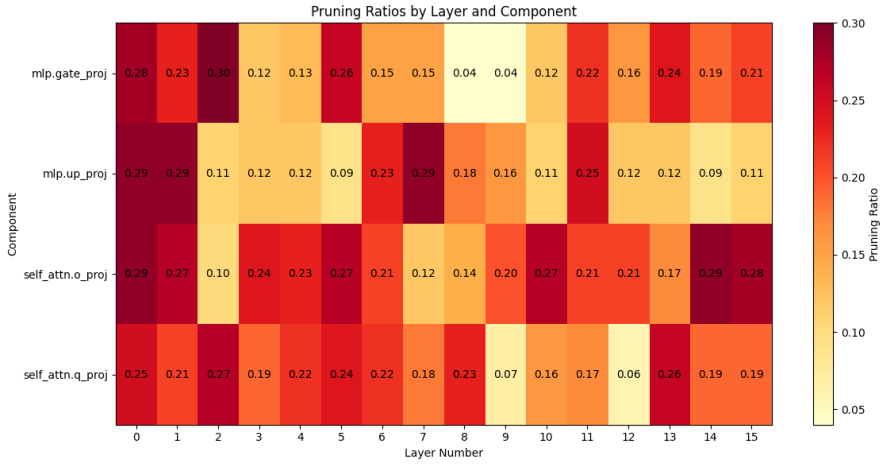

Figure 5: Final Agent-Determined Pruning Ratios For Each Prunable Part

In order to evaluate the pruning strategy, a benchmark pruned model was created. In order to ensure a fair comparison, this model uses the same compression ratio as the agent-pruned model, but instead applies a uniform pruning ratio of 18% to all prunable parts. The agent-pruned and benchmark models were evaluated across 100 samples of prompt batches with a batch size of 3, and the average KL divergence for each sample on both models is shown in Figure 6. We see that both models exhibit similar patterns of KL divergence on each prompt batch, but that the agent-pruned model is always higher than the benchmark (which is unfavorable). The average KL divergence of the agent-pruned model is 2323 while for the benchmark model it is 1168. Overall, this result shows that while the agent's pruning strategy does not ruin the model, the final output quality is worse than that of a trivial pruning strategy.

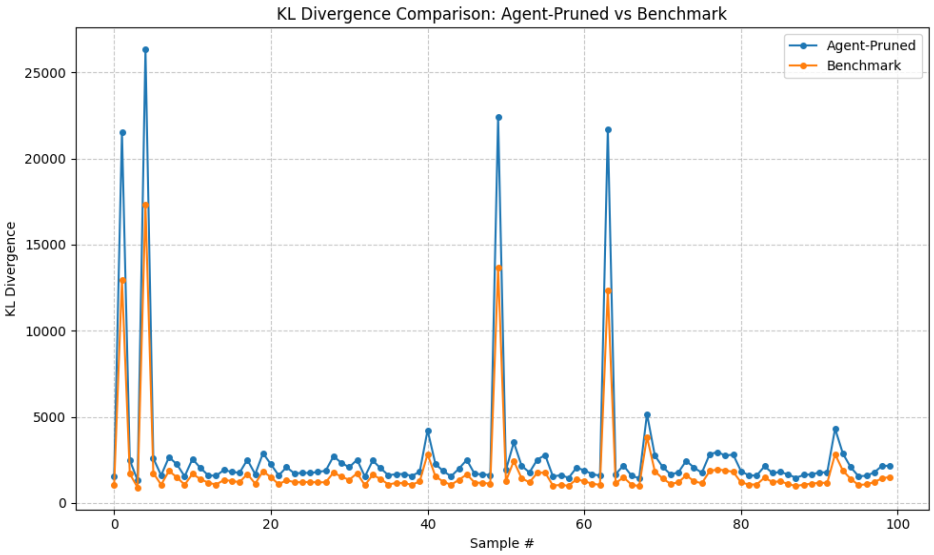

Figure 6: KL Divergence of Agent-Pruned and Benchmark Models on 100 3-Prompt Samples

5.0.3 DPO + Pruning

To assess the robustness of the DPO-finetuned model under compression, we applied the learned pruning strategy at three levels of intensity, resulting in overall parameter reductions of 2.1%, 6.2%, and 10.3%. We applied the same scaled pruning ratios to both the DPO and original models for a fair comparison. Figure 7 summarizes the results. As expected, pruning degrades performance across both models. However, the DPO-finetuned model consistently outperforms the baseline at all pruning levels. Notably, the DPO model pruned by 6.2% matches the unpruned baseline on correctness (27% vs. 28%) while leaking significantly less code (27% vs. 66%). This suggests that DPO not only improves task performance, but also increases resilience to pruning.

At the highest pruning level (10.3%), performance of both models drops sharply. The DPO model identifies errors correctly in fewer than 10% of cases, yet it still maintains a lower code giveaway rate than the baseline. These trends also hold for secondary metrics like helpfulness and clarity: although both models degrade, the DPO model's scores decline more gracefully. In short, while aggressive pruning harms both correctness and hint quality, DPO fine-tuning provides a valuable buffer against this degradation, enabling smaller models to retain more utility.

6 Discussion

While this research did not result in a new way to consistently generate error message hints via a locally run model, it does provide strong motivation for future work on the same task. The improvement of the base model's performance after fine-tuning it with DPO suggests that small models can learn the characteristics necessary to perform on the task. One challenge of our approach was that the data for our DPO training came from only two sources (Llama 3.2 3B Instruct and GPT-4o) and the vast majority of preferred options came from one of the two sources. Improving both the quantity and diversity of data may help to improve the model further. Interestingly, our results do point to the fact that a fine-tuned model will maintain performance at a higher rate than the base model while being pruned. This could potentially mean that a better performing model could be pruned without much of a cost to its performance.

For the RL agent, the results indicate that the agent likely did not explore the action space enough to come up with a truly good pruning strategy. Throughout the development process, we experienced many occurrences of the agent output collapsing (providing a uniform pruning ratio or not pruning at all), and it took many iterations of refining the state-action-reward framework and hyperparameter tuning to achieve non-trivial convergence. We attribute these difficulties to the high amount of noise

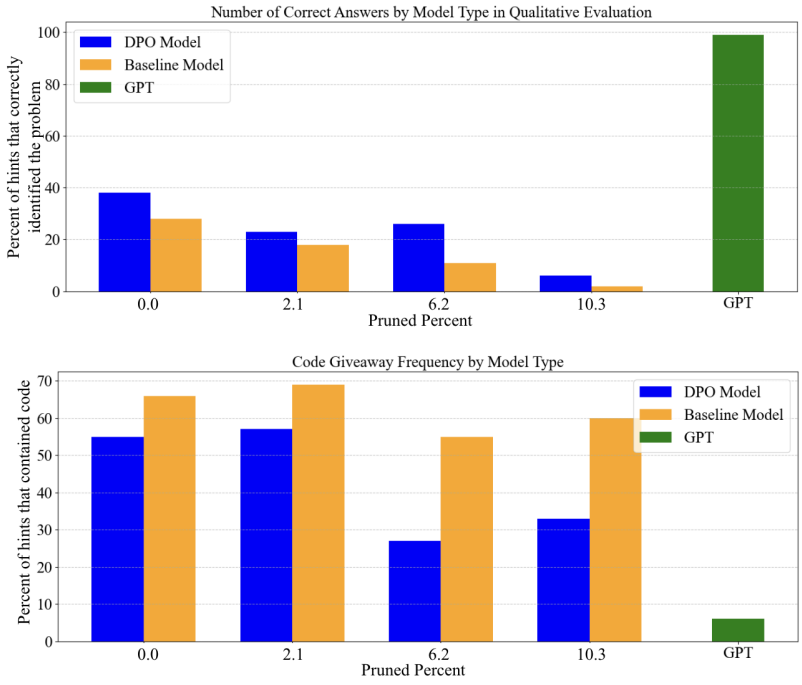

Figure 7: Evaluation results comparing the base model to the DPO model at 4 levels of pruning, along with the model currently used for this task, GPT-4o

present in the KL metric: during training, it appears that it is quite challenging for the agent to predict the KL penalty when trying different pruning strategies. In order to achieve a more meaningful result from the pruning agent, another metric of measuring model quality might be necessary. This could involve again using GPT-4o to grade the outputs in terms of helpfulness and not giving away code, but it's likely that the number of API calls required to achieve this would be time and cost intensive.

7 Conclusion

As previously stated, we did not develop a final model that is adequate for immediate deployment. However, we do show the convergence of an RL pruning agent balancing model size and accuracy and model output quality improvement through DPO.

We anticipate that future work will include diversifying and improving the quality of training data and looking into alternative pruning strategies such as structured pruning. The improvements in performance that we saw, while not strong enough to be actionable, do indicate that there is a potential path to developing a small LLM for error message hints.

8 Team Contributions

- **Ben Pekarek**: Ben setup the teacher/student LLMs for pruning, performed the inital integration of code, added metrics tracking and checkpointing, assisted with trying different RL algorithms and tuning hyperparameters, and trained the final pruning agent.
- **TJ Jefferson**: TJ collected, formatted and preprocessed the data, and designed and ran the DPO fine tuning pipeline. Additionally, he ran the evaluation and benchmarking runs with the finalized pruning strategies. TJ interfaced with the Code In Place lab's A100s as needed.
- **Eli LeChien**: Eli defined the state-action-reward framework, programmed the initial actor-critic algorithm and training loop, assisted with trying different RL algorithms and tuning hyperparameters, and conducted the final evaluation and benchmarking of the agent-pruned model.

Changes from Proposal Contributions were generally in-line with the proposal, though some roles were expanded due to the needs that were not originally forecasted. Overall member contributions were roughly equal.

References

Code in Place. 2024. Code in Place. https://codeinplace.stanford.edu/ [Online; accessed August-2024].

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL] https://arxiv.org/abs/2501.12948

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. arXiv:2305.14314 [cs.LG] https://arxiv.org/abs/2305.14314

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. *AMC: AutoML for Model Compression and Acceleration on Mobile Devices*. Springer International Publishing, 815–832. https://doi.org/10.1007/978-3-030-01234-2_48

Siqi Li, Quan Lu, ning jiang, Jingyang Xiang, Chengrui Zhu, Jiateng Wei, Jun Chen, and Yong Liu. 2024. SparsitySolver: Efficient Reinforcement Learning-based Pruning for LLMs. https://openreview.net/forum?id=zZU69H8tcr

Ali Malik, Juliette Woodrow, Brahm Capoor, Thomas Jefferson, Miranda Li, Sierra Wang, Patricia Wei, Dora Demszky, Jennifer Langer-Osuna, Julie Zelenski, Mehran Sahami, and Chris Piech. 2023. Code in Place 2023: Understanding learning and teaching at scale through a massive global classroom. https://piechlab.stanford.edu/assets/papers/codeinplace2023.pdf.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. arXiv:2203.02155 [cs.CL] https://arxiv.org/abs/2203.02155

Rafael Rafailov, Joey Hejna, Ryan Park, and Chelsea Finn. 2024. From r to Q^*: Your Language Model is Secretly a Q-Function. arXiv:2404.12358 [cs.LG] https://arxiv.org/abs/2404.12358

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 53728–53741. https://proceedings.neurips.cc/paper_files/paper/2023/file/a85b405ed65c6477a4fe8302b5e06ce7-Paper-Conference.pdf

Rajarshi Saha, Naomi Sagan, Varun Srivastava, Andrea J. Goldsmith, and Mert Pilanci. 2024. Compressing Large Language Models using Low Rank and Low Precision Decomposition. arXiv:2405.18886 [cs.LG] https://arxiv.org/abs/2405.18886

Sierra Wang, John Mitchell, and Chris Piech. 2024. A Large Scale RCT on Effective Error Messages in CS1. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1395–1401. https://doi.org/10.1145/3626252.3630764