Selective Self-to-Supervised Fine-Tuning for Generalization in Large Language Models

Sonam Gupta [*], **Yatin Nandwani, Asaf Yehudai, Dinesh Khandelwal, Dinesh Raghu and Sachindra Joshi**
IBM Research

Abstract

Fine-tuning Large Language Models (LLMs) on specific datasets is a common practice to improve performance on target tasks. However, this performance gain often leads to overfitting, where the model becomes too specialized in either the task or the characteristics of the training data, resulting in a loss of generalization. This paper introduces Selective Self-to-Supervised Fine-Tuning (S3FT), a fine-tuning approach that achieves better performance than the standard supervised fine-tuning (SFT) while improving generalization. S3FT leverages the existence of multiple valid responses to a query. By utilizing the model's correct responses, S3FT reduces model specialization during the fine-tuning stage. S3FT first identifies the correct model responses from the training set by deploying an appropriate judge. Then, it fine-tunes the model using the correct model responses and the gold response (or its paraphrase) for the remaining samples. The effectiveness of S3FT is demonstrated through experiments on mathematical reasoning, Python programming and reading comprehension tasks. The results show that standard SFT can lead to an average performance drop of up to 4.4 on multiple benchmarks, such as MMLU and TruthfulQA. In contrast, S3FT reduces this drop by half, i.e. 2.5, indicating better generalization capabilities than SFT while performing significantly better on the fine-tuning tasks.

1 Introduction

Large Language Models (LLMs) have made remarkable progress in recent years, demonstrating impressive capabilities across a wide range of tasks, including question-answering (Rajpurkar et al., 2016), summarization (Nallapati et al., 2016), and more (Brown et al., 2020). Supervised fine-tuning (SFT) of LLMs on task-specific data is a widely used approach to enhance their performance in specialized applications. While fine-tuning improves task accuracy, it can cause the model to overfit to the domain or style present in the training data, potentially limiting its broader applicability. In this work, we focus on exploring how to fine-tune an LLM for a specific task while preserving its general-purpose capabilities.

Task	Write a function to add two lists using map and lambda function.

Gold	def add_list(nums1,nums2):
	  result = map(lambda x, y:x+y, nums1, nums2)
	  return list(result)
	log probability = -84.32

Model Prediction	def add_list(list1, list2):
	  return list(map(lambda x, y:x+y, list1, list2))
	log probability = -36.64

Table 1: An example from the MBPP (Python Programming dataset) (Austin et al., 2021), along with Mistral-7B-Instruct-v0.2's prediction.

SFT relies on gold responses for training. We observe two key challenges when performing SFT on LLMs: (1) in many datasets, model-generated responses—though differing from gold responses—are still valid and acceptable, and (2) the distribution of gold responses often diverges significantly from the model's own response distribution. For instance, consider an example from the MBPP dataset in Table 1. The base model, Mistral-7B-Instruct-v0.2, assigns a log probability of -84.32 to the gold answer. When prompted with the same question, the model generates a response containing the same information as the gold output, but with a much higher log probability of -36.64. This phenomenon is common in generation tasks, where semantically equivalent responses can have widely varying log-likelihood scores. This observation suggests that model-generated responses can align more closely with the model's native distribution, whereas gold responses may lie further apart.

[*]Corresponding author: Sonam.Gupta7@ibm.com

As a result, training exclusively on gold responses risks introducing distributional drift, potentially reducing the model's ability to generalize effectively.

To address this issue, we propose Selective Self-to-Supervised Fine-Tuning (S3FT), a simple yet powerful technique that utilizes model-generated answers for a subset of the training dataset to adapt the model to desirable behaviours while maintaining generalization. S3FT fine-tunes the model on its own generated output for cases where it behaves desirably and on gold output (or its paraphrase) for the remaining data. This approach allows the model to learn from its successes while benefiting from human-labeled data when needed.

In our experiments, we show that S3FT outperform existing approaches, including SFT, on diverse tasks, namely code generation Austin et al. (2021), math problem solving (Cobbe et al., 2021) and reading comprehension (Kwiatkowski et al., 2019). To show that S3FT generalizes better and retains the base model's capabilities, we evaluate on multiple datasets such as MMLU (Hendrycks et al., 2020), TruthfulQA (Lin et al., 2022), and Hellaswag (Zellers et al., 2019). We observe that the drop in performance for S3FT on these benchmarks is smaller than that of existing approaches, demonstrating better generalization capabilities.

2 Proposed Method

Let $\mathcal{M}_\theta$ parameterized by θ be a given large language model. Let $\theta = \theta_0$ be the given model weights obtained after pre-training and instruction tuning the model. We refer to $\mathcal{M}_{\theta_0}$ as the base model. Further, let $\mathcal{T}$ be the new task that we wish to teach the model $\mathcal{M}_\theta$, and let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ be the corresponding training dataset for $\mathcal{T}$. In standard SFT, we backpropagate through the standard Cross Entropy loss over the training dataset, $\mathcal{D}$. However, SFT can cause a degradation of $\mathcal{M}_{\theta_0}$ general capabilities by forcing the model to predict a gold response which is further away from the $\mathcal{M}_\theta$ responses' distribution.

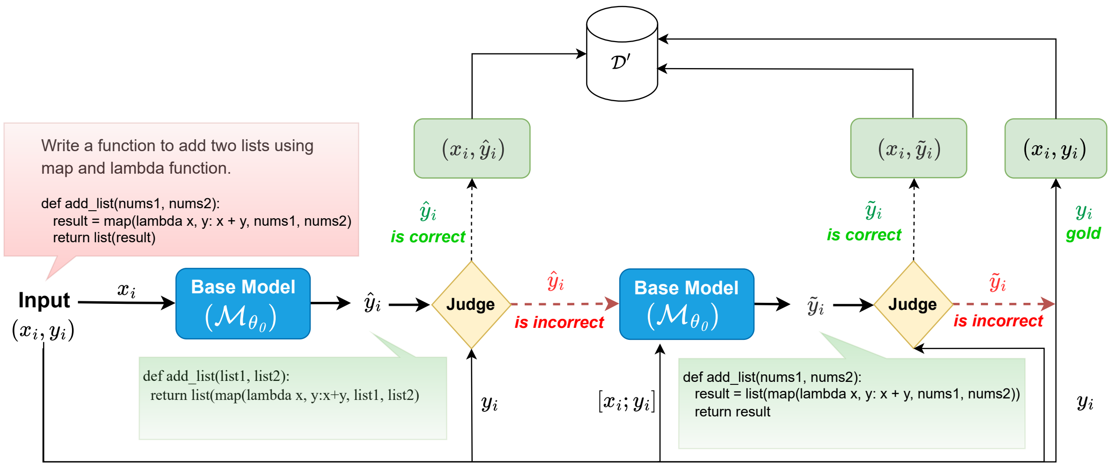
To solve this, our method relies on two key observations. First, for many NLP tasks e.g. machine translation, summarization, reading comprehension *etc.*, the same input x can have multiple valid responses. Nandwani et al. (2020) define this setup as 1oML (one of many learning) and propose various strategies to handle it, albeit for combinatorial problems. Second, teaching the model using its own words can help preserve its own distribution. This can regularizes the model training, helping it to tackle catastrophic forgetting and maintaining its general capabilities. We note that the standard practice of regularization via replay buffer (Hayes et al., 2020), which involves mixing a subset of instruction-tuning dataset with the given task-specific data $\mathcal{D}$ is not always feasible as the instruction-tuning dataset may not be available. These two observations are the basis for S3FT. For each example in the data, we start by generating a prediction $\hat{y}_i = \mathcal{M}_{\theta_0}(x_i)$ with the base model where x_i is the input. If $\hat{y}_i$ is equivalent to y_i, we use $(x_i, \hat{y}_i)$ for model training. If $\hat{y}_i$ is not equivalent, we use $\mathcal{M}_{\theta_0}$ to rephrase the gold answer y_i in its own language, $\tilde{y}_i = \mathcal{M}_{\theta_0}([x_i; y_i])$. If $\tilde{y}_i$ is not equivalent to y_i we use the gold answer, y_i. Figure 1 and the algorithm in Appendix A.4 gives an overview of our approach.

An important component of S3FT is the ability to identify the equivalence of model's prediction or gold paraphrasing to the gold answer. Towards that end, we can use judges that aim to assess the equivalence with $\hat{y}_i$, either by heuristics such as checking the bottom-line agreement of the predicted and gold response or employing a stronger LLM as a judge to measure more semantic equivalence.

3 Experimental Setup

We evaluate and compare the proposed S3FT method with the vanilla SFT and other methods that try to perform fine-tuning while retaining generalization capabilities. We aim to answer the following research questions: **RQ1. In-Domain Performance:** How well does S3FT learn the fine-tuning task compared to baselines when fine-tuned and evaluated on the same dataset? **RQ2. Generalization:** How well does S3FT retain the inherent capabilities of the base model post fine-tuning? **RQ3. Effect of gold response's paraphrasing:** How beneficial is it to use gold paraphrases that are closer to the base model's distribution?

Datasets: We focus on three tasks, improving the mathematical reasoning abilities, basic python programming and reading comprehension skills. For enhancing the mathematical skills we experiment with GSM8K (Cobbe et al., 2021) dataset. This dataset consists of grade school level math word problems and solutions. To boost the Python programming skill, we experiment with MBPP (Austin et al., 2021) dataset which consists of a task description, three test cases, and a code solution for



Write a function to add two lists using map and lambda function.

```
def add_list(nums1, nums2):
   result = map(lambda x, y: x + y, nums1, nums2)
   return list(result)
```

```
def add_list(list1, list2):
  return list(map(lambda x, y:x+y, list1, list2))
```

```
def add_list(nums1, nums2):
    result = list(map(lambda x, y: x + y, nums1, nums2))
    return result
```

Figure 1: **An overview of our proposed approach**: Given the input x_i and its corresponding gold response y_i, we employ the base model $\mathcal{M}_\theta$ to transform y_i such that it is correct but at the same time closer to model's distribution. First, the model predicts the output $\hat{y}$. The judge decides whether the y_i is correct. If true, it becomes part of the training dataset; otherwise, we paraphrase ($[x_i; y_i]$) to obtain $\tilde{y}_i$. The judge evaluates the correctness of $\tilde{y}_i$. If true, we use $\tilde{y}_i$; otherwise, we use y_i as the target response. The resulting dataset $\mathcal{D}'$ is used to train the model.

each example. We experiment with a variant of NQ (Kwiatkowski et al., 2019) dataset as introduced in (Slobodkin et al., 2023) to enhance the reading-comprehension skills. The NQ dataset from (Slobodkin et al., 2023) contains 3800 (context, question, answer) pairs. The questions are categorized into two types: (i) answerable – where the provided context is relevant and contains sufficient information to derive an answer, and ii) unanswerable – where the context lacks the necessary information to answer the question. A detailed description of the datasets, along with the training, validation and testing splits, is provided in the Appendix A.2.

Evaluation Metrics: For GSM8K, we assess the correctness of a generated response by checking if its predicted answer matches the final answer in the gold solution. For MBPP, we evaluate the generated code by executing it against the provided test cases. If the code passes all the test cases, it is considered correct. For NQ dataset, we employ Mistral-instruct-v2 (7B) as a judge. Following (Badshah and Sajjad, 2024), we use a reference-guided prompt to judge the correctness of the generated responses. More details about the LLM judge can be found in appendix A.5. Finally, we report the accuracy of each method for the three datasets.

Human Study on Judges' Accuracy: Since responses in GSM8K and NQ are more open-ended, we conduct a human study to assess the reliability of the judges used for evaluating correctness. For GSM8K, we randomly sampled 50 examples and found that the judge is approximately 96% accurate. Similarly, for NQ, testing on a set of 200 random samples yielded an accuracy of 86%. Additionally, we highlight that improving the quality of the judges could further enhance the evaluation process and improve the overall generalizability of the method.

Base model and Baselines: We experiment with Mistral-instruct-v2 (7B) (Jiang et al., 2023) as our base model. We use three baselines: (1) prompting the base model (see Appendix A.1 for the exact prompt), (2) Supervised Fine-Tuning (SFT), and (3) Self-distill Fine-tuning (SDFT) (Yang et al., 2024). SDFT is a contemporary work that adopts gold answer rephrasing to preserve the generalization of the fine-tuned model.

Implementation Details: For all fine-tuning, we use Low-Rank Adaptation (LoRA) (Hu et al., 2022) with a rank of 8, a scaling factor of 16 and a dropout of 0.1. Please see Appendix A.3 for more details.

4 Results and Discussion

In-Domain Performance Table 2 reports the accuracy of Mistral-Instruct-v2 (7B) fine-tuned over GSM8K, MBPP and NQ datasets. Here, we evaluate the base, SFT, SDFT and S3FT models over the test set corresponding to the training dataset. The base model shows suboptimal performance on all the datasets. Fine-tuning using our method signif-

Method	**GSM8K**	**MBPP**	**NQ**
Base	40.3	22.2	64.7
SFT	53.4	32.8	60.0
SDFT	54.8	35.8	**67.1**
S3FT	**56.9**	**39.4**	**67.1**

Table 2: Performance comparison of various fine tuning techniques over two different tasks using Accuracy(%) as the metric. S3FT achieves the best performance on the fine-tuning task while preserving the generalization to the other tasks.

Dataset	**Model responses**	**Gold paraphrases**	**Gold responses**
GSM8K	49.5%	42.1%	8.4%
MBPP	30.2%	32.4%	37.4%
NQ	66.0%	25.6%	8.1%

Table 3: The proportions of examples used from each data type; Model responses, Gold paraphrases, and Gold responses. We can see that the exact composition depend on the dataset, but the majority of responses are either the model responses or its paraphrasing.

icantly improves the model's performance. S3FT achieves a 2.1% gain on GSM8K, 3.6% gain on MBPP and comparable performance to the state-of-the-art method SDFT on reading comprehension. We note that S3FT significantly outperforms SFT on all three datasets. This improvement stems from a simple yet effective technique of using the base model's responses for training when the base model is correct instead of the gold response. This verifies that S3FT learns the fine-tuning task well outperforming all other baselines (RQ1).

Generalization A major issue with SFT is its tendency to diminish the model's previously learned skills. To demonstrate that S3FT alleviates this issue, we compare the baselines and our method against the base model on a diverse set of publicly available benchmarks. Specifically, we evaluate them on MMLU (Hendrycks et al., 2020), TruthfulQA (Lin et al., 2022), Hellaswag (Zellers et al., 2019) and, Winogrande (Sakaguchi et al., 2019). Table 4 reports our findings.

We observe that irrespective of the task used for fine-tuning, there is a drop in the performance of SFT models across all benchmarks. On average, SFT on Mistral-7B results in an average drop of 4.4, 2.7 and 5.8 when trained using GSM8K, MBPP and NQ respectively. On the other hand, S3FT results in an average drop of only 2.5 when trained on GSM8K and MBPP and a drop of 1.0 when trained on MBPP. This clearly demonstrates that our proposed technique for fine-tuning preserves the base model's capabilities (RQ2) without relying upon any kind of replay buffer which might not even be available in many cases. On the other hand, standard SFT results in overfitting to the training dataset, resulting in catastrophic forgetting of the skills acquired by the base model during pre-training and instruction tuning.

Effect of Gold response's paraphrasing Our findings from Tables 2 and 4 indicate that using the model's response as the target rather than the gold response significantly enhances fine-tuning performance without compromising the model's overall capabilities. Training on gold responses can cause a shift from the original distribution, negatively impacting the model's generalization. Figure 2 illustrates this gap between the distributions of gold responses, gold paraphrases, and the base model's responses. The distribution is plotted using 84 random samples for which the model's prediction and the paraphrases of the gold responses are acceptable. The plot shows that in several cases model responses can be valid and closer to model's own distribution and therefore curating the training data in this way leads to minimal changes in the model parameters while fine-tuning. Thus, the closer the gold response's paraphrases to the model distribution, the better it is (RQ3).

Training Data Proportions Table 3 presents the proportions of examples in which S3FT utilizes base model responses, gold paraphrases, and gold responses across the three datasets. Notably, the base model's responses were deemed acceptable for at least 30% of the training samples. When incorporating gold paraphrases, more than 50% of the training data originates from the model's own responses. This suggests that the base model's outputs play a crucial role in S3FT's success by acting as an effective regularizer, helping to mitigate overfitting in the fine-tuning process.

5 Related Work

Continual learning for language models faces challenges like overfitting and loss of generalization (Yogatama et al., 2019; Zhang et al., 2021). Rehearsal-based methods, such as experience replay (Rolnick et al., 2019) and representation con-

Train Dataset	Method	MMLU	TruthfulQA	HellaSwag	WinoGrande	Average
	Base	58.7	59.6	66.0	74.0	64.6
GSM8K	SFT	57.0	48.0	62.4	73.4	60.2
	SDFT	57.6	51.1	62.6	**73.6**	61.2
	S3FT	**58.2**	**53.7**	**63.2**	73.5	**62.1**
MBPP	SFT	57.5	51.9	64.7	73.6	61.9
	SDFT	57.1	56.6	**65.2**	73.4	63.1
	S3FT	**58.0**	**57.2**	64.9	**74.4**	**63.6**
NQ	SFT	54.4	45.7	63.1	72.0	58.8
	SDFT	55.2	53.9	**65.3**	**72.6**	61.8
	S3FT	**57.0**	**54.1**	64.8	**72.6**	**62.1**

Table 4: Generalization over other benchmarks. 1st row reports the score obtained by prompting the base model Mistral-instruct-v2-7B.

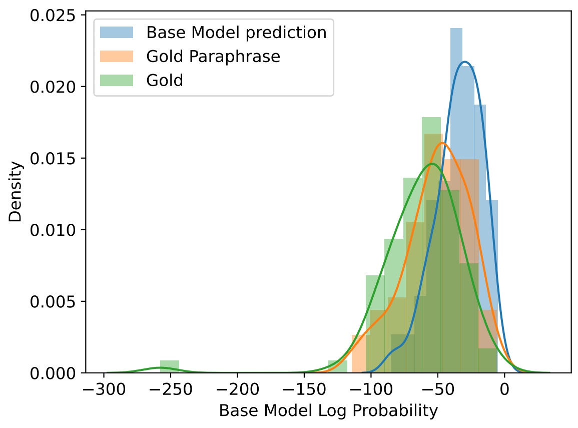


Figure 2: Histogram of the log probability assigned by Mistral-7B-Instruct-v0.2 to the gold responses, paraphrase of gold responses and model's own predictions. The distribution is based on 84 examples from the MBPP training data.

solidation (Bhat et al., 2022), show promise but often depend on real data, which may be scarce. To address this, model-generated responses are used through techniques like self-training (He et al., 2020; Xie et al., 2020) and self-supervised learning (Lewis et al., 2020). However, their effectiveness in continual learning remains underexplored. Current methods focus on real data rehearsal (Scialom et al., 2022; Mok et al., 2023; Zhang et al., 2023), but these can be resource-intensive. In contrast, S3FT avoids storing past data or training extra generative models, making it more data-efficient and practical for real-world use. SDFT (Yang et al., 2024), the closest contemporary work to ours, is thoroughly compared in experiments, where we achieve significantly higher accuracy.

6 Conclusion

In this paper, we present S3FT, a fine-tuning approach that enhances both task-specific performance and generalization across tasks, as shown on benchmarks like MMLU and Truthful QA. S3FT leverages the idea that multiple correct outputs may exist and avoids unnecessary changes by fine-tuning on gold response (or its paraphrase) only when the model's response is incorrect. In future work, we plan to investigate techniques like few-shot prompting for sampling correct outputs that are closer to the model's own distribution to reduce the changes from the base model's weights.

7 Limitations

S3FT requires running model inference on the entire training dataset, identifying correct and incorrect responses, then performing gold rephrasing on incorrect responses and evaluating their correctness. These additional steps introduce a few extra requirements not present in standard SFT. First, S3FT's improved results come at the cost of increased computational demands. Second, it requires a qualified judge. As we show, reliable heuristics can be used to assess response equivalence in mathematical reasoning and Python programming tasks. However, for open-ended tasks such as summarization and translation, no simple heuristics exist. Here, we demonstrate that for reading comprehension, an LLM judge can serve as a reliable evaluator. As LLM judges improve, the usability and applicability of our method can extend to a broader range of tasks.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Sher Badshah and Hassan Sajjad. 2024. Reference-guided verdict: Llms-as-judges in automatic evaluation of free-form text, 2024. *URL https://arxiv.org/abs/2408.09235*.

Loubna Ben Allal, Niklas Muennighoff, Logesh Kumar Umapathi, Ben Lipkin, and Leandro von Werra. 2022. A framework for the evaluation of code generation models. `https://github.com/bigcode-project/bigcode-evaluation-harness`.

Sarthak Bhat, Oleg Sidorov, Ulrich Paquet, and Anirudh Garg. 2022. Representation consolidation for continual learning. In *International Conference on Learning Representations*.

Tom 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. *Advances in neural information processing systems*, 33:1877–1901.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2024. A framework for few-shot language model evaluation.

Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. 2020. Remind your neural network to prevent catastrophic forgetting. In *European Conference on Computer Vision*, pages 466–483. Springer.

Junxian He, Jiatao Gu, Jianfeng Shen, and Marc'Aurelio Ranzato. 2020. Revisiting self-training for neural sequence generation. In *International Conference on Learning Representations*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves 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.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252.

Tanya Mok, Luisa Wellhausen, Hyung Won Choe, and Hannaneh Hajishirzi. 2023. Large language models can be continuously updated without forgetting. *arXiv preprint arXiv:2303.01926*.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çaglar Gulçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Conference on Computational Natural Language Learning*. Association for Computational Linguistics (ACL).

Yatin Nandwani, Deepanshu Jindal, Parag Singla, et al. 2020. Neural learning of one-of-many solutions for combinatorial problems in structured output spaces. In *International Conference on Learning Representations*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. 2019. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*, volume 32.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*.

Thomas Scialom, Thierry Charnois, and Sylvain Lamprier. 2022. Continual learning for large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 5432–5442.

Aviv Slobodkin, Omer Goldman, Avi Caciularu, Ido Dagan, and Shauli Ravfogel. 2023. The curious case of hallucinatory (un) answerability: Finding truths in the hidden states of over-confident large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3607–3625.

Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. 2020. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10687–10698.

Zhaorui Yang, Tianyu Pang, Haozhe Feng, Han Wang, Wei Chen, Minfeng Zhu, and Qian Liu. 2024. Self-distillation bridges distribution gap in language model fine-tuning. *arXiv preprint arXiv:2402.13669*.

Dani Yogatama, Cyprien de Masson d'Autume, Jerome Connor, Tomas Kocisky, Mike Chrzanowski, Lingpeng Kong, Angeliki Lazaridou, Wang Ling, Lei Yu, Chris Dyer, et al. 2019. Learning and evaluating general linguistic intelligence. *arXiv preprint arXiv:1901.11373*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2021. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*.

Zijun Zhang, Yue Wu, Hao Guan, Xinlei Chen, and Yue Zhang. 2023. Continual learning with transformers: Challenges and solutions. *arXiv preprint arXiv:2302.13713*.

```
"<s>[INST] You are an expert in math. Below is a math
question. Write a response that appropriately answers
the question. Your final answer should be an integer
at the end of your response, formatted as: The answer
is {answer}.
Question: {{question}} [/INST]"
```

Figure 3: Prompt used for training the model on GSM8K dataset.

```
<s>[INST] You are an expert Python programmer, and
here is your task: {{task}}. Your code should pass
these tests:

{{testcase_1}}
{{testcase_2}}
{{testcase_3}}

Generate only the python code encapsulated between
[[BEGIN]] and [[DONE]] tags. Do not generate any
explanation. [/INST]
```

Figure 4: Prompt used for predicting the base model response on MBPP training dataset.

A Appendix

A.1 Prompt for Generating the Response

We list the prompts used with mistral-instruct-v2 to generate the base model responses, gold paraphrases and training in this section. For the sake of consistency and fair comparison, the same prompts are used for fine-tuning using SFT, SDFT and S3FT techniques. Figure 4 and Figure 6 present the prompts used for generating the base model prediction and gold paraphrasing for MBPP dataset. To simplify the process of extracting the code in the generated output, we always ask the model to generate only the Python code starting with the string "[[BEGIN]]" and ending in the string "[[DONE]]". For the MBPP dataset, the training prompt is the same as the base model's prediction prompt. Similarly, Figure 5 and Figure 7 present the prompt used for generating the base model response and paraphrasing the gold response. The details of the training prompt are provided in Figure 3.

A.2 Datasets Details

GSM8K: GSM8K is a math word problem dataset comprising of 7473 training examples and 1319 test samples. Each question takes approximately 2-8 steps to solve. Solving these problem require knowledge of basic algebra only. Solutions are human written containing steps in natural language as well. We note that GSM8K does not have a validation set, so we took randomly sampled 150 samples from the dataset as validation.

```
"<s>[INST] You are an expert in math. Below is a math
question. Write a response that appropriately answers
the question. Your final answer should be an integer
at the end of your response, formatted as: The answer
is {answer}.
Question: {{question}}
Great! Let's think step by step. [/INST]"
```

Figure 5: Prompt used for predicting the base model's output on GSM8K dataset.

MBPP: MBPP stands for Mostly basic python programming dataset. As the name suggests, each example of this dataset contains a task description along with three test cases that the code solving the given task should pass. The gold response contains the python code. There are 372 training examples, 90 validation examples and 500 examples in the testing dataset.

NQ: NQ is a content-grounded QA dataset. To increase the complexity of the task, (Slobodkin et al., 2023) augment the NQ dataset with unanswerable queries. Here, the grounding content consists of a single paragraph and the gold answers are short phrases.

A.3 Training Details

Training for all the experiments was carried out on a single A100 (80 GB) GPU. None of the experiments took more than 1.5 hours to train. To generate the base model's responses, we deployed Mistral-7B-Instruct-v0.2using vLLM on a single A100 (80 GB) GPU. It took 1 hour to generate the base model's predictions for GSM8K, 30 minutes to generate the base model's responses for NQ dataset and 15 minutes on MBPP dataset. The entire life cycle, including training data generation, fine-tuning and evaluation, did not take more than 5 hours.

We use a learning rate of $1 \times e^{-4}$. For GSM8K and NQ, we train all the models for 5000 steps, validate after every 500 steps, and select the best checkpoint. For MBPP, we train the models for 1000 steps, validate after every 100 steps, and select the base checkpoint based on the accuracy over the validation set.

For evaluating the GSM8K dataset, we matched the last number of the gold response with the last number extracted from the predicted response. If these answers matched, we considered the gener-

```
<s>[INST] Below are an instruction that describes a task along with a reference answer. Using
the reference answer as a guide, write your own response.

### Instruction:
You are an expert Python programmer, and here is your task: {{task}}. Your code should pass
these tests.

{{testcase_1}}
{{testcase_2}}
{{testcase_3}}

Generate only the python code encapsulated between [[BEGIN]] and [[DONE]] tags. Do not generate
any explanation.

Reference Answer:
{{gold_response}}

Response:[/INST]
```

Figure 6: Prompt used for paraphrasing the gold response of training partition of the MBPP dataset.

```
<s>[INST] You are an expert in math. Below are a reference answer and its corresponding math
question. Refer to the reference answer and write a response that appropriately answers the
question. Your final answer should be an integer at the end of your response, formatted as:
The answer is {answer}.

Reference Answer:
{{gold_response}}

Question:
{{question}}

Response:
Great! Let's think step by step. [/INST]
```

Figure 7: Prompt used for paraphrasing the gold response of training partition of the GSM8K dataset.

```
<s>[INST] You are a helpful assistant acting as an impartial judge. You will be given a
Question, a Reference Answer, and a Provided Answer. Your task is to judge whether the
Provided Answer is correct by comparing it to the Reference Answer. If the Provided Answer is
correct, choose only 'True', otherwise choose only 'False'. Question: {{question}}
Provided Answer:
{{predicted_response}}
Reference Answer:
{{gold_response}}
Decision: [True/False] [/INST]
```

Figure 8: The prompt used for judging the correctness of the responses generated by the model for NQ dataset. Here, question refer to the question that the model is answering, predicted response is the response generated by the fine-tuned model and gold response is the gold answer for the question.

ated response to be correct. For evaluating the Python codes, we use bigcode-evaluation-harness (Ben Allal et al., 2022). LLM-judge's output was parsed to check the correctness of the answers for the NQ dataset. If the judge responded "TRUE", the answer was considered correct; else, if the judge predicted "FALSE", the answer was deemed incorrect. We always used greedy decoding when generating the model responses. Thus, we do a single run of the evaluation and report the numbers. To evaluate the trained models on various benchmarks, the widely popular lm-evaluation-harness (Gao et al., 2024) repository was used.

A.4 S3FT Algorithm

Here we show the algorithm for S3FT.

Algorithm 1 Training Method

```
 1: Input: Base Model $\mathcal{M}_{\theta_0}$
    Data $D = \{(x_i, y_i)\}_{i=1}^{n}$
 2: $D' \leftarrow \{\}$
 3: for each example $(x_i, y_i) \in D$ do
 4:     $\hat{y}_i \leftarrow \mathcal{M}_{\theta_0}(x_i)$
 5:     if $\hat{y}_i = y_i$ then
 6:         $D' \leftarrow D' \cup \{(x_i, \hat{y}_i)\}$
 7:     else
 8:         $\tilde{y}_i \leftarrow \mathcal{M}_{\theta_0}(y_i)$
 9:         if $\tilde{y}_i = y_i$ then
10:             $D' \leftarrow D' \cup \{(x_i, \tilde{y}_i)\}$
11:         else
12:             $D' \leftarrow D' \cup \{(x_i, y_i)\}$
13:         end if
14:     end if
15: end for
16: Train $\mathcal{M}_{\theta_0}$ on $D'$ to obtain updated parameters
    $\theta$
17: Output: Updated model parameters $\theta$
```

A.5 Judges for evaluating the model performance

For tasks like reading comprehension, conventional metrics like BLEU, ROUGE are useful but inadequate to capture the semantics of the generated responses. Therefore, we use LLM-as-a-judge for evaluation. Specifically, inspired by (Badshah and Sajjad, 2024), we use Mistral-instruct-v2 (7B) model as a judge with the prompt shown in Figure 8.