SELF–BOOST: Boosting LLMs with Iterative Self-Generated Data

Anonymous ACL submission

Abstract

Instruction finetuned large language models (LLMs) have shown impressive performance solving a diverse range of natural language processing (NLP) tasks involving classification and reasoning. However, this can be particularly challenging in low-data regimes. Recent methods have shown boosting via iterative full finetuning to be an effective method to augment the training data by using the incorrect examples to generate synthetic data using a teacher LLM. However, data generation at scale using a teacher LLM can be costly, and full finetuning can be computationally expensive. To address this, we introduce SELF–BOOST, an iterative data augmentation and instruction finetuning strategy that has no external dependence on any teacher models. SELF–BOOST uses parameter efficient finetuning (PEFT) with Llama 3 8B to instruction finetune a model using the seed data, uses the *same* model to generate examples similar to the misclassifications, and also the *same* model to verify and filter the generated examples. Our experiments show that performance on TREC, GSM8K, and CaseHOLD improves by 21.6%, 5.6% and 1.3% respectively, when compared to our baseline.

1 Introduction

LLMs have made significant advancements across diverse benchmarks, operating based on custom directives or example-based prompts. However, real-world applications reveal limitations in their adaptability to specialized domains and memory capacity for long prompts. Fine-tuning can be effective but requires substantial training data.

Recently, Lee et al. (2024) introduced a novel strategy, LLM2LLM, for improving the performance of pre-trained LLMs using boosting. This approach leverages a teacher LLM to iteratively augment a seed dataset with synthetic data, specifically addressing areas where the model underperforms. During each iteration, the teacher LLM generates new training examples that target the student model's errors for further refinement. This method has demonstrated significant improvements across various NLP tasks, illustrating that targeted, iterative training can effectively enhance model performance even in settings with low seed data. However, the LLM2LLM approach employs expensive teacher models and full finetuning. This leads to high computational costs and are expensive to scale; for instance, the token cost for GPT-4 can range from $10 to $30 per million tokens.

To address this, we introduce SELF-BOOST, a cost-effective alternative that enhances LLMs through iterative self-generated data. We draw inspiration from the analogy of a motivated learner who is able to identify their own weaknesses, come up with more practice questions that exemplify these weaknesses, and use them for additional practice to get better. Similarly, SELF–BOOST enhances a model through a cyclic process involving instruction fine-tuning, evaluation, data augmentation, and verification over multiple iterations. It identifies errors, generates new examples, and rigorously verifies them using task-specific prompts and majority voting. Only high-quality examples are retained for subsequent iterations, effectively enhancing model performance without extensive data collection.

Our key contributions are as follows:

1. Introducing SELF–BOOST, utilizing the model itself for generating and verifying data, reducing computational costs.

2. Demonstrating success using PEFT techniques on a single GPU.

3. Introducing a novel self-verification process and running ablations to show its value in efficiency and scalability.

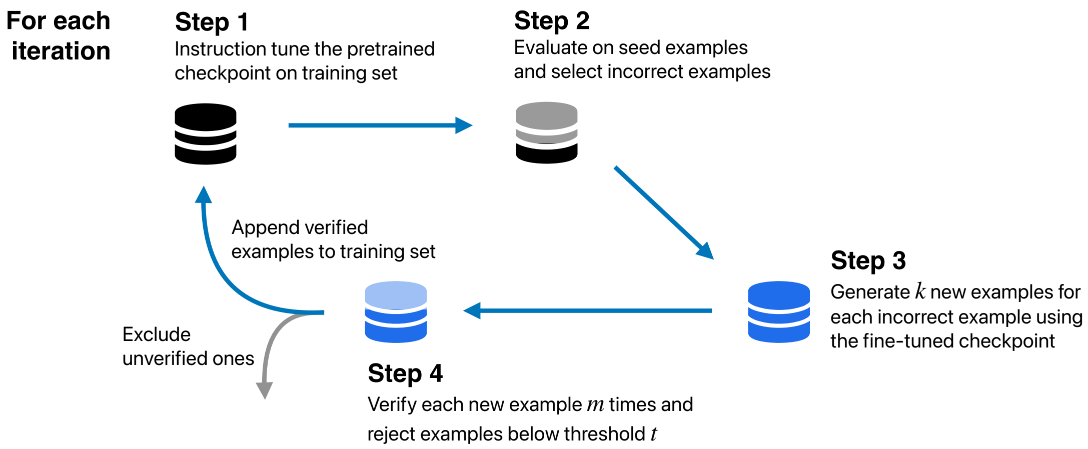

Figure 1: An visual explanation of our method, SELF–BOOST. At each iteration, we instruction finetune the pretrained model using the current training dataset and evaluate the seed examples using the trained data. The finetuned model is used to generate examples similar to the incorrect seed examples, and verify that they are consistent. The verified examples are added back to the training dataset, and the cycle continues.

4. Conducting experiments showing the effectiveness of SELF–BOOST, with increasing boosting cycles significantly improving model performance.

2 Related Work

In this section, we present a literature review of prior work that is relevant to our method.

Boosting: Freund and Schapire (1997) propose boosting method to improve learning algorithms by combining weak classifiers. The boosting algorithm calls this weak learner repeatedly, and each iteration generates a new weak prediction rule. Finally the algorithm combines all the weak rules into a single one that is likely to achieve higher accuracy. Wang et al. (2023a) propose a Chain-of-Knowledge (CoK) prompting method that applies a boosting-style algorithm to improve the reasoning capabilities of large language models. The CoK method applys a boosting-style algorithm to improve LLMs' reasoning capabilities through iterative rethinking and knowledge re-weighting, allowing CoK to boost reasoning performance on reasoning tasks like commonsense QA, arithmetic, and symbolic reasoning.

LLM2LLM: As our baseline, Lee et al. (2024) introduce an innovative approach to enhance pretrained LLMs using a teacher LLM to iteratively augment a seed dataset with synthetic data targeting the model's weaknesses. At each round, the teacher LLM generates new, targeted training examples based on the student's errors, which are used for further training. This cycle continues, iteratively enhancing the student's ability to handle previously challenging examples. This method shows significant improvements but relies on access to a strong teacher model and full finetuning.

Self-Verify: Weng et al. (2023) propose a self-verification method that leverages LLMs' inherent ability to self-verify, enhancing their reasoning without additional verifier training. It employs a two-step process: forward reasoning and backward verification, using True-False Item Verification and Condition Mask Verification.

Self-Consistency: Wang et al. (2023b) present a method called Self-Consistency, which improves LLMs' reasoning on complex tasks using a three-step approach: chain-of-thought prompting, sampling multiple reasoning paths, and evaluating consistency to choose the final answer.

Self-Instruct: In the Self-Instruct method (Wang et al., 2023c), the authors propose an automatic data generation and fine-tuning process for LLMs. Initially, instruction data is defined to generate tasks that an LLM can understand and perform. The LLM then creates new instructions, generates instances, filters low-quality content, and is fine-tuned on the resulting high-quality data to enhance its ability to follow instructions accurately. The Self-Instruct approach is designed to significantly reduce the dependency on human-annotated data, leveraging the LLM's own generative capabilities to improve its performance on a broader range of instructional tasks.

Instruction Finetuning (FLAN): Wei et al. (2022) introduce a method called instruction finetuning, which is designed to boost the zero-shot learning abilities of language models. This approach involves finetuning a pre-trained language model with 137 billion parameters across an array of datasets. This process enhances the model's ability to perform on tasks it hasn't seen before. The resultant model undergoes evaluation through a series of unseen tasks, demonstrating its improved zero-shot learning capabilities. Also, it shows that instruction tuning leads to better performance on tasks naturally verbalized as instructions, and it is less effective on tasks directly formulated as language modeling, like commonsense reasoning and co-reference resolution.

3 Methodology

3.1 Baseline

Following the terminology defined by Lee et al. (2024), we assume we are given an instruction finetuned LLM model $\mathcal{M}^0$ that is pretrained and instruction finetuned on some large corpus. In all of our experiments, $\mathcal{M}$ is an instruction tuned Llama 3 8B model, the most recent iteration of the Llama model family (Touvron et al., 2023) released by Meta [1]. We also have access to D^0, a domain specific seed dataset for which pre-trained or finetuned performance is unsatisfactory either due to the complex domain, lack of data availability, or inability to collect more data. To improve on pre-trained model performance, in our baseline we instruction finetune $\mathcal{M}^0$ with D^0 using low-rank adaptation (LoRA) (Hu et al., 2021), which is a PEFT method that can achieve near full-finetuning performance with a fraction of trainable parameters. LoRA adapters with rank $r = 16$, $\alpha = 32$, and dropout 0.1 are added to all the linear layers. We obtain the trained model $\mathcal{M}^1$ by training for 3 epochs with a learning rate of 3×10^{-4} and an AdamW (Loshchilov and Hutter, 2019) optimizer. $\mathcal{M}^1$ is then evaluated on an unseen test dataset D' from the same domain.

3.2 Self–Boost

We outline the approach for our method Self–Boost in Algorithm 1. The steps involved in each iteration i are briefly described below.

Instruction Finetuning: We instruction tune $\mathcal{M}^0$ on data D^i for 3 epochs with the same training hyperparameters as the baseline method outlined in Section 3.1 to obtain the finetuned model $\mathcal{M}^i$. Ablation studies from Lee et al. (2024) show that finetuning $\mathcal{M}_0$ from scratch shows significant improvements in performance when compared to continuous finetuning on $\mathcal{M}^{i-1}$ due to overfitting.

[1] https://llama.meta.com/llama3/

Evaluation: We evaluate $\mathcal{M}^i$ on the original seed data D^0. An exact match of an extracted answer is used to identify the training examples in D^0 that are incorrectly predicted by $\mathcal{M}^i$. The extraction process varies by task. For example, for GSM8K, only text after the "####" string is considered, and for TREC and CaseHOLD, everything after the assistant role header is considered.

Data Augmentation: We sample k times from $\mathcal{M}^i$ by conditioning on a task-specific prompt to generate k similar examples for each incorrect example in D^0. Ablation studies from Lee et al. (2024) show that generating examples using previously generated examples could propagate errors in data augmentation. Since this is likely true in our setting as well, we only generate samples using only the incorrect seed examples.

Verification: To ensure that the new training data is of high quality, for each newly generated example, we condition $\mathcal{M}^i$ on a task-specific prompt and sample m times to verify if the provided reasoning and answer are correct for the generated question. This approach is inspired by chain-of-thought reasoning (Wei et al., 2023), self-verify (Weng et al., 2023), and self-consistency (Wang et al., 2023b), which show that complex reasoning admits different paths of thinking, and correct reasoning processes tend to have greater agreement in the final answer that incorrect processes. Using a minimum threshold t, we take a majority vote of the m verifications and reject the examples that do not meet the threshold.

Iteration: Finally, the verified, generated examples are added back to the dataset D^i to obtain the augmented dataset D^{i+1}. This process is repeated for n iterations until we get the final model $\mathcal{M}^n$.

4 Experimental Setup

In this section, we outline the experimental setup for our results. In addition to demonstrating empirical results, we also perform an extensive ablation study to understand the settings that work well for our framework. All experiments are performed on either a single NVIDIA RTX 4090 or A10 GPU. The prompts used for example generation and verification for each of the benchmarks can be found in Appendix A.

Algorithm 1 Given a small seed dataset D^0, SELF–BOOST uses the same model $\mathcal{M}^i$ at each step to finetune with an augmented dataset, evaluate training examples, generate new examples that are similar to the incorrect one, and verify that the new examples are consistent. The new examples are added back to the dataset and used to finetune the model in the next iteration.

procedure SELF–BOOST($\mathcal{M}^0$, D^0)
  $i \leftarrow 0$
  while $i < n$ **do**
    $\mathcal{M}^i \leftarrow$ Finetune($\mathcal{M}^0$, D^i)
    $E^i \leftarrow$ Evaluate($\mathcal{M}^i$, D^0) ▷ Evaluate on seed data
    $W^i \leftarrow$ Filter(E^i,D^0) ▷ Keep wrong answers
    $A^i \leftarrow$ {Generate$_1$($\mathcal{M}^i$, W^i), ..., Generate$_k$($\mathcal{M}^i$, W^i)} ▷ Self-augment
    $C^i \leftarrow$ Majority Vote{Verify$_1$($\mathcal{M}^i$, A^i), ..., Verify$_m$($\mathcal{M}^i$, A^i)} ▷ Ensure self-consistency
    $D^{i+1} \leftarrow D^i + C^i$ ▷ Append to data
    $i \leftarrow i + 1$
  end while
  Evaluate M^n
end procedure

4.1 Methods

We compare our method SELF–BOOST to the baseline method of a single iteration of finetuning without any data augmentation, similar to the evaluation methodology by Lee et al. (2024). However, unlike their evaluation, for our method we present results of $\mathcal{M}^n$ (the final model) instead of $\mathcal{M}^*$ (the best performing model on a validation held-out set), as we believe that this is a more robust measure of the method.

4.2 Benchmarks

To evaluate the improvement that our framework offers over the baseline, we present the performance of both methods on three benchmarks: GSM8K (Cobbe et al., 2021), TREC (Li and Roth, 2002) and CaseHOLD (Zheng et al., 2021), all of which involve very different tasks. GSM8K is an open-ended generation task involving complex reasoning over grade school math problems. TREC is a 6-way classification task involving assigning the intent of a question to a category such as abbreviation, entity, or location. CaseHOLD is multiple choice task involving determination of a court's holding on a cited case. To evaluate the methods in a low data setting, we sample a seed dataset of 1 - 10% (uniformly sampled) of the training data for GSM8K, 1.1 - 2.2% of training data for TREC (10-20 instances per class), and 0.1 - 0.5% of training data for CaseHOLD.

4.3 Hyperparameters

In our ablation studies, we experiment with a few hyperparameters and settings that our method offers. Specifically, we explore generation sample size $k \in \{1, 3, 5\}$, verification sample size $m \in \{1, 3\}$, number of boosting iterations $n \in \{3, 5, 10\}$, and whether verification is enabled.

4.4 Measurement

For all benchmarks, we measure and report the exact match accuracy between the label and parsed predicted response for the entire test set. For ended-tasks such as GSM8K, the parsing involves the extraction of just the final answer. For classifications tasks like TREC and CaseHOLD, the class is extracted.

5 Results

5.1 Main Results

We apply the SELF–BOOST framework on the newly released instruction tuned Llama 3 8B model on various benchmarks, including GSM8K (Cobbe et al., 2021), TREC (Li and Roth, 2002) and CaseHOLD (Zheng et al., 2021). Due to limited computing resources, we present results for a subset of sample rates from 0.5% to 10% to emulate a low-data regime and test the efficacy of our framework.

We present the accuracy of $\mathcal{M}^{10}$ on the test split of each benchmark dataset after 10 iterations of SELF–BOOST, as well as the baseline method (as outlined in Section 3.1) in Table 1. On 1.1% of the TREC dataset, the baseline achieves 69% accuracy.

With our SELF-BOOST framework, the accuracy goes up to 90.6% (improved by 21.6%) by iteratively generating 118 additional examples based on the seed examples. With double the available data of 2.2% of TREC, our method still achieves 93.3% accuracy, improved by 8.8%. These are hard examples for which the model failed to give the correct answer. We only generate new examples similar to examples in the seed training set, according to the ablation study in (Lee et al., 2024) which suggests that generating using non-seed examples may cause drifting, introduce more inaccurate examples, and eventually hurt the model's performance. Other than generating new data, we also found verifying the generated data could help further boost the performance, which we will discuss later in Section 5.3. We observe a similar trend for GSM8K and CaseHOLD, though with smaller improvements than on TREC. We further analyze these results in Section 5.4.

5.2 Effect of Boosting Iterations

We compare the performance of our method by varying the number of boosting iterations n. The results are shown in Figure 2. For both TREC and GSM8K, we observe that increasing the number of iterations allows the performance to continue improving. We hypothesize that this is because at each iteration, we are augmenting the training dataset with generated examples that are similar to training examples that the model got wrong. This leads to the difficult patterns and concepts being weighed more in the loss during the forward pass, and LoRA parameters being optimized towards better understanding these patterns and concepts in the backward pass. This could allow the model to generalize better to similar unseen examples. We also hypothesize that the improvements in performance across iterations could be due to the fact that at each iteration i, model $\mathcal{M}^i$ is trained on a richer dataset D^i, and thus has the ability to generate better quality examples than the model from previous iteration $\mathcal{M}^{i-1}$. A possible ablation to validate this hypothesis, which we leave to our future work, is to compare performance with using model $\mathcal{M}^0$ to generate the examples at each iteration instead.

5.3 Effect of Verification

We show the results of some ablations exploring the effects of verification as part of our method.

Presence of Verification: For both TREC and GSM8K, we enable verification with $m = 1, t = 1.0$ (i.e. we sample once and reject the generated example if the answer is different). We observe that when verifications are enabled (*Verifications*), there is a drop in the test accuracy of $\mathcal{M}^n$ as compared to when verifications are disabled (*No Verifications*). We hypothesize that this is due to the fact that the total number of training examples is lower when verifications are enabled, since inconsistent examples are filtered out. To account for this, we test a setting in which the number of generated examples per incorrect training example is increased from 1 to 3 (*Verifications* + *More Generations*). This setting results in the best performance. This ablation reveals an interesting tradeoff between the quantity and quality of training examples. Clearly, both quantity and quality are necessary to improve performance. These results are shown in Figure 3.

Consistency in Verifications: We also test the effect of increasing consistency in verification by varying the number $m \in \{1, 3\}$, which is the number of times each generated example is verified by the model. Both settings use $t = 1.0$, meaning that if any of the verifications are unsuccessful, the example is rejected. Intuitively, we would assume that a higher m in this setting would mean that there is greater agreement in verifications, resulting in higher quality examples being retained, and that the higher quality in training examples would results in improved performance on the test set. However, we observe that this is true only for GSM8K but not for TREC. Although increasing m led to fewer examples being generated in both GSM8K and TREC, it led to better performance for GSM8K but poorer performance for TREC. We hypothesize that tradeoff between quantity of examples and the quality of examples may vary based on the type of task. For a simpler classification task such as TREC, it is possible that having more examples, even if they are not completely accurate, could be helpful in allowing the model to learn simpler patterns such as the possible output classes in the classification task. However, for more complex open-ended tasks such as GSM8K, the addition of inaccurate examples may confuse the model and result in poorer reasoning, leading to worse performance. These results are shown in Figure 4.

In Figure 5, we show an example of our verification process with $m = 3$ and $t = 1.0$, as well as the reasoning provided by the sample which was not verified. In this case, the other samples missed out the fact that the three inline skates were col-

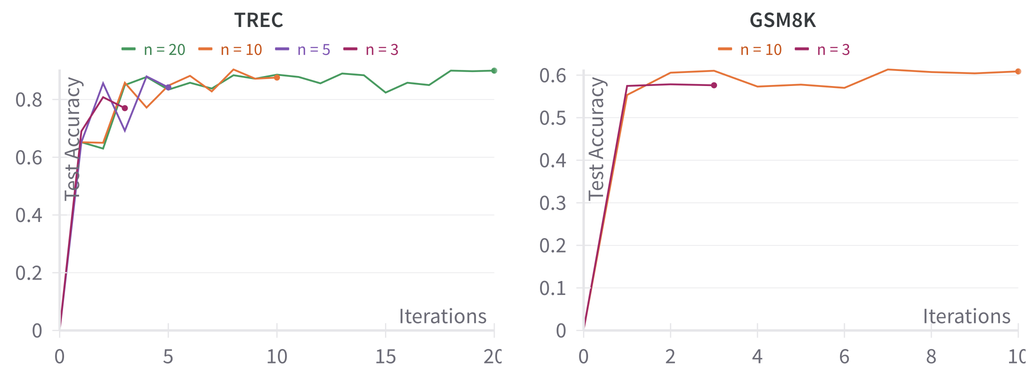

Figure 2: Our ablations show that continued boosting improves performance. With continued boosting, the number of generated examples that similar to incorrect training examples increases, allowing the model to learn more difficult patterns.

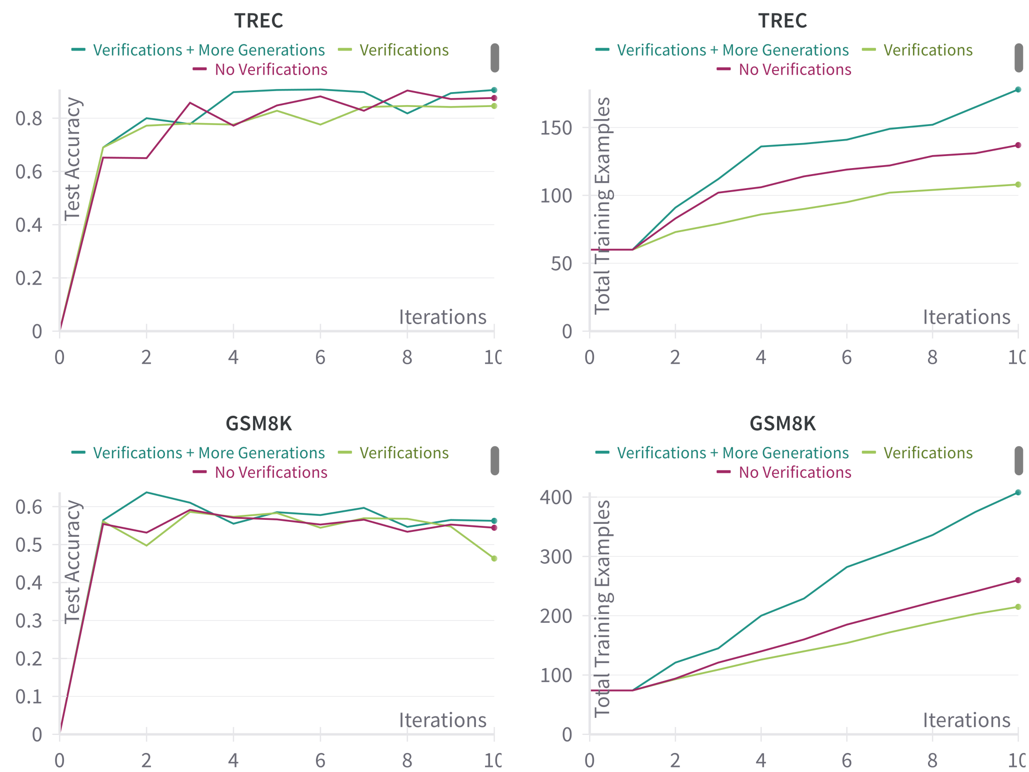

Figure 3: Our ablations show that simply adding verification is insufficient to improving performance, as there is a risk of insufficient training data. Additional generated examples are also required to ensure there is enough data.

lectively 3/4 the price of the roller skates (due to this word *each* being missing in the question), but the last verification was able to successfully identify this. In tricky cases like this, leveraging more verifications results in a higher chance to filter out wrong examples to maintain the high quality of the training set.

5.4 Effect of More Data

Our ablations also show that the SELF-BOOST framework is most effective in low-data settings, since it can self-generate new examples based on the wrong examples and self-verify the quality of the generated examples. This is best illustrated by experiments on TREC – even though we achieve a higher test accuracy on 2.2% sample rate than on 1.1%, the accuracy gain compared with baseline is much more significant on 1.1%, the lower

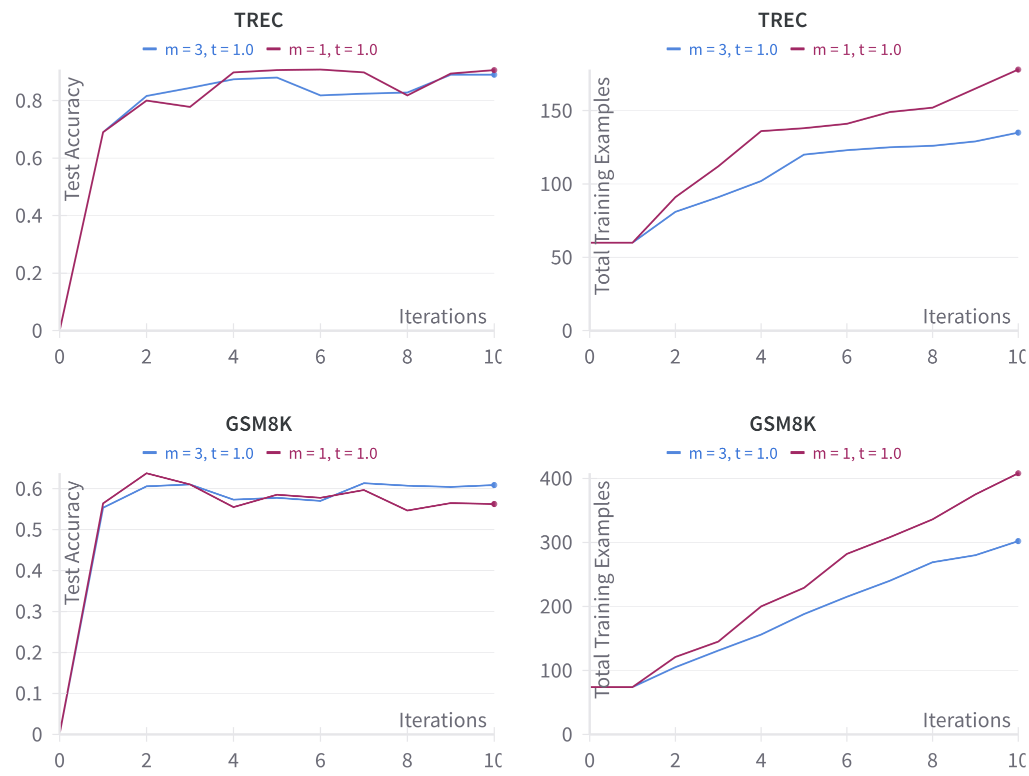

Figure 4: Our ablations show that the tradeoff of data quantity and quality may be task dependent. For simpler tasks like TREC, more training data matters more than data quality. For complex reasoning tasks such as GSM8K, data quality is much more important.

Dataset	% Data	# Seed Examples	# Augmented	Test Accuracy (%)		
				Pretrained	**Baseline**	**SELF–BOOST**
TREC	1.1%	60	118	0.0	69.0	90.6 **(+21.6)**
TREC	2.2%	120	131	0.0	84.8	93.6**(+8.8)**
GSM8K	1%	74	302	0.0	55.3	60.9 **(+5.6)**
GSM8K	10%	747	888	0.0	63.9	59.1 **(-4.8)**
CaseHOLD	0.1%	45	32	1.4	61.5	62.0**(+0.5)**
CaseHOLD	0.5%	204	114	1.4	68.9	70.2 **(+1.3)**

Table 1: The results of our method, SELF–BOOST, compared to zero-shot and baseline performance on the test sets of TREC, GSM8K, CaseHOLD. The columns *% Data* and *# Seed Examples* describe the amount of seed data available at the start of boosting. The column *# Augmented* describes the total number of examples generated during the boosting process. Pretrained performance is evaluated on $\mathcal{M}^0$ without any instruction tuning. Baseline performance is evaluated on $\mathcal{M}^1$ after finetuning with only the original seed examples. SELF–BOOST performance is evaluated on $\mathcal{M}^{10}$ after 10 iterations of boosting. Overall, we observe that test accuracy improves with SELF–BOOST.

subsample rate, as shown in Table 1. This shows that when SELF-BOOST can generate enough examples (in TREC's case, for both subsample rates our method has more augmented examples than the seed training set), it can largely boost the finetuning accuracy.

For both subsample rates on CaseHOLD, our method generates fewer augmented examples than the original seed training set, which could explain why the accuracy gains on CaseHOLD are less significant than on TREC. In our experiments, due to the large number of tokens in CaseHOLD, we only try to generate 5 new examples per wrong example. We hypothesize that even higher numbers of generations per wrong example (e.g. 10, 50) could improve performance further. TREC and CaseHOLD are both multi-classification task benchmarks, while GSM8K is a quite challenging

```
For newly generated example:
Maria buys one set of roller skates for $120 and three sets of inline skates for 3/4rds of the price of the roller skates. How much does she pay total?', 'output': 'First find the cost of one set of inline skates: $120 * 3/4 = $<<120*3/4=90>>90\nThen find the total cost of the inline skates: $90/set * 3 sets = $<<90*3=270>>270\nThen add that amount to the cost of the roller skates to find the total cost: $270 + $120 = $<<270+120=390>>390\n#### 390
CORRECT: CORRECT
CORRECT: CORRECT
NOT VERIFIED (2 / 3.0): it got 1/3 verifications wrong:
#Judgement: WRONG
The student's answer is incorrect because they added the cost of the inline skates to the cost of the roller skates, but they did not take into account the fact that the cost of the inline skates is already three times the cost of the roller skates. Therefore, the total cost is actually $120 + $90 = $210, not $390.
```

Figure 5: Multiple verifications help filter out incorrectly generated examples. In this example, the question is missing the word *each*, which leads to a different answer that the one in the generated example.

math reasoning task benchmark. On GSM8K, we see an unusual accuracy drop on a higher sample rate with 10%. We hypothesize that this is because the 10% subsampling provides 10x more examples in the seed training set, and the quality of our self-generated and self-verified examples by Llama 3 8B is poorer than the human-labeled GSM8K examples in the seed dataset. For challenging reasoning tasks like GSM8K, adding more, poor quality examples to the finetuning process may hurt the accuracy when given enough seed examples. For such tasks, we might need a teacher model with stronger reasoning capability to replace the self-generation and self-verification steps.

6 Future Work

Building upon our results for SELF-BOOST, we propose a few directions for future research.

Hybrid Teachers Models: Our current method leverages student models as the teacher models and performs self-augmentation. The combination of SELF-BOOST with other successful methodologies like LLM2LLM could prove to be beneficial. For example, employing a low-cost teacher model to guide the self-boosting process might strike a balance between computational efficiency and the robustness of generated examples, providing a more refined dataset for training.

Teacher Model Capabilities: Another potential direction is to thoroughly investigate the impact of teacher models' varying capabilities on the quality of data generation, and the consequent performance of student models. This analysis will involve systematically varying the complexity and instructional capacity of teacher models to see how these variations influence the quality of the synthetic training data they produce. We will explore metrics such as fidelity, diversity, and relevance of the generated data, and assess how these qualities affect the learning outcomes in student models. Ultimately, this will allow us to identify optimal characteristics and configurations of teacher models that most effectively enhance student model performance, potentially leading to more efficient and targeted training methodologies.

Tradeoff between Quantity and Quality: As described in Section 5.3, our ablations suggest that the tradeoff between the amount of data generated and the quality of data generated may be task specific. To this end, a potential future direction could build upon our method to make this determination part of the method. In other words, perhaps the method could determine if data quantity is required (in which case it may be fine to use a cheaper teacher model to generate the data), if data quality is required (in which case the method can invoke a more expensive teacher model to generate the data).

7 Conclusion

The SELF–BOOST methodology represents a significant step forward in the autonomous enhancement of large language models (LLMs) through iterative self-generated data. By eliminating the dependence on costly teacher models and leveraging the model's own errors for data augmentation, SELF–BOOST not only reduces computational expenses but also enhances model accuracy effectively. This process is particularly valuable in resource-constrained scenarios where acquiring large amounts of annotated data is impractical. The empirical results indicate that SELF–BOOST can substantially improve model performance, particularly in challenging tasks like GSM8K and TREC. Moreover, the method's ability to refine and expand its training dataset autonomously makes it a promising approach for ongoing model improvement in various AI applications.

Limitations

Our choices of experimental setups were limited in several ways:

Model Selection Given our limited computational resources, we only try our SELF-BOOST framework with pretrained checkpoints of Llama 3 8B-Instruct. It would be better to validate our framework on other popular model series like the PaLM series with sizes from 8B to 540B, or Llama 2 from 7B to 70B, and the 70B version of Llama 3.

Framework Hyper-parameters Exploration In our SELF-BOOST, we have considered and supported modification of the following hyper-parameters:

1. n_iters: the number of iterations in the boosting process, also the number of weak learners
2. $n\_epochs\_per\_iter$: the fine-tuning epochs during each iteration of the boosting
3. $n\_generations\_per\_incorrect\_example$: for each incorrect example, the number of newly generated examples
4. $enable\_verification$: whether to verify the newly generated examples
5. $min\_verify\_threshold$: the threshold for a generated example to pass the verification
6. n_voting_verify: for each new example, how many times it has to be verified
7. $seed\_generation\_only$: whether to add generated examples that are based on non-seed examples to the training set

For hyper-parameters like n_iters and $seed\_generation\_only$, we set them to 10 and true respectively, based on the results and analysis in the ablation study of (Lee et al., 2024). We did explore $enable\_verification$ with true and false, $n\_generations\_per\_incorrect\_example$ from 1 to 5, and n_voting_verify from 1 to 5 on our own, although in a relatively limited range. In the final results, we were using one value for all benchmarks based on our observation. For the remaining $n\_epochs\_per\_iter$ and $min\_verify\_threshold$, they seem trivial to the framework so we did not explore how exactly these two hyper-parameters will change the results. Given more computation resources, it would be helpful to thoroughly explore all these hyper-parameters on our own. Using grid search would be costly, so we propose to use Bayesian optimization to find the proper hyper-parameter settings in the future work.

Finetuning Dataset Sampling Rate For three of our benchmarks, we use two sample rates on each of them to randomly sample a small subset to simulate the low-data regime. In (Lee et al., 2024), they conduct experiments on 4-9 different sampling rates, even including 100% to show that the method helps improve metrics under data-sufficient regimes. Given our computational resources, we only focused on 2 lowest sampling rates so we might neglect some vital trends in slightly higher sampling rates.

References

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Yoav Freund and Robert E Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

Nicholas Lee, Thanakul Wattanawong, Sehoon Kim, Karttikeya Mangalam, Sheng Shen, Gopala Anumanchipali, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. 2024. Llm2llm: Boosting llms with novel iterative data enhancement. *Preprint*, arXiv:2403.15042.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. *Preprint*, arXiv:1711.05101.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

Jianing Wang, Qiushi Sun, Nuo Chen, Xiang Li, and Ming Gao. 2023a. Boosting language models reasoning with chain-of-knowledge prompting. *Preprint*, arXiv:2306.06427.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. *Preprint*, arXiv:2203.11171.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023c. Self-instruct: Aligning language models with self-generated instructions. *Preprint*, arXiv:2212.10560.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned language models are zero-shot learners. *Preprint*, arXiv:2109.01652.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. 2023. Large language models are better reasoners with self-verification. *Preprint*, arXiv:2212.09561.

Lucia Zheng, Neel Guha, Brandon R. Anderson, Peter Henderson, and Daniel E. Ho. 2021. When does pretraining help? assessing self-supervised learning for law and the casehold dataset. *Preprint*, arXiv:2104.08671.

Appendix

A Prompt Examples

TREC Generation System Prompt:
You are QuestionGPT, an AI agent who knows the class of different question. You are training someone how to classify different questions based on what the questions are asking form. You are trying to give the user assistance by giving them more practice questions for the questions that they get wrong. Here are the requirements:
1. A GPT language model should be able to complete the problem. For example, do not ask the assistant to create any visual or audio output. For another example, do not ask the assistant to wake you up at 5pm or set a reminder because it cannot perform any action.
2. The question should be in english.
3. The questions that you generate should have only 1 of the following intents: - ABBR (Abbreviation) - ENTY (Entity) - DESC (Description/Concept) - HUM (Human) - LOC (Location) - NUM (Number)
4. The questions should always have 1 specific class.
5. The intent of the question must come from the list above.
6. Don't make any mistakes with your answer yourself.
7. Try not to copy too much information from the original problem. You don't want the user to just memorize the practice problems.
8. Make the class the same as the question that the user got wrong.
9. The question should be something that an ASR model could output: it must sound like something a human could say.

Always return your instructions in the form:

#Question: What are the requirements to become a pilot?
#Class: DESC

TREC Generation User Prompt: The following is a question.
Classify the question into the following categories:
- ABBR
- ENTY
- DESC
- HUM
- LOC
- NUM

Question: Who is the author of the novel "To Kill a Mockingbird"?
Class: HUM
Give me another 1 similar question with the same class HUM.

GSM8K Generation System Prompt:
You are a educational A.I. whose purpose is to take math problems that students get wrong and generate new problems to help them practice their mathematical skills. Your goal is to generate a set of new math problems that reflect the different skills and techniques found in the example problem.
Here are the requirements:
1. A GPT language model should be able to complete the problem. For example, do not ask the assistant to create any visual or audio output. For another example, do not ask the assistant to wake you up at 5pm or set a reminder because it cannot perform any action.
2. The math problem should be in English.
3. The output should be an appropriate response to the question. Make sure the output is less than 100 words.
4. The answer to the problem should be expressed as a number, not a fraction. For example, if the answer is one-half, return 0.5, not 1/2 or "one half".
5. The answer to the problem should not have units i.e. if the answer is 6 cups, just write 6 as the [ANSWER]
6. Always include some calculation to show your work for how you got your ANSWER.
7. Don't make any mathematical mistakes of your own!
8. Try not to copy too much information from the original problem. If you must, try and replace names and numbers so that we can test the student's understanding, rather than their ability to memorize previous test questions. Always return your instructions in the form:
#Question: [QUESTION]
#Answer: [CALCULATION]
[ANSWER]

GSM8K Generation User Prompt:
The student was given the following question:

Chrystal's vehicle speed is 30 miles per hour. Ascending the mountain decreases its speed by fifty percent, and descending the mountain increases its speed by twenty percent. If the distance going to the top of the mountain is 60 miles and the distance going down to the foot of the mountain is 72 miles, how many hours will Crystal have to pass the whole mountain?

The answer key has this as the rationale and answer:

The vehicle's speed decreases to 30 x 0.50 = «30*0.50=15»15 miles per hour when ascending to the top of the mountain. So, the total time Crystal will have to spend going to the top of the mountain is 60 / 15 = «60/15=4»4 hours. And the speed of the vehicle increases by 30 x 0.20 = «30*0.20=6»6 miles per hour when going down to the foot of the mountain. So, the total speed of the vehicle when going down is 30 + 6 = «30+6=36»36 miles per hour. Thus, Chrystal will have to spend 72 / 36 = «72/36=2»2 hours to descend from the mountain. Therefore, the total hours she will spend to pass the whole mountain is 4 + 2 = «4+2=6»6 hours.
6

Please generate 1 similar question, along with the correct calculations and rationale.

CaseHOLD Generation System Prompt:
You are LawGPT, an AI agent who knows everything there is to know about U.S. law. You know the result of every court case and you know every law in the lawbook. The user is trying to choose the correct holding of the case given the context and argument of the court. You are trying to give the user assistance by giving them more practice questions for the questions that they get wrong.
Here are the requirements:
1. A GPT language model should be able to complete the problem. For example, do not ask the assistant to create any visual or audio output. For another example, do not ask the assistant to wake you up at 5pm or set a reminder because it cannot perform any action.
2. The context, holding, and options should be in English.
3. The questions that you generate should test for whether the user understands the case names and their holdings and whether the user can re-frame relevant holdings to backup the argument in the context.
4. The context should always end with a citation such as "See United States v. Newman, 125 F.3d 863 (10th Cir.1997) (unpublished) (<HOLDING>); United States v. Dodge, 846 F.Supp. 181,"
5. The citation absolutely needs to have the mask phrase <HOLDING> which is the place where the legal holding would normally be.
6. The questions should always be multiple choice.
7. There should always be 5 options: 1 options should be a holding that backs up the argument in the context, the other 4 should be sufficiently different. Each option has to start with the word "holding"
8. There can only be 1 answer: A, B, C, D, or E.
9. Don't make any mistakes matching the holdings yourself.
10. Try not to copy too much information from the original problem. You don't want the user to just memorize their answer.
11. Make the context similar to the context in question, make sure that the holding that is being tested is the same.
12. The wrong answer choices can be any other reasonable holding, but it should be sufficiently different from the correct answer.
13. Do not make your context too short. Remember, these arguments in the context are being made by judges and should look like they were written by a judge.

Always return your instructions in the form:

#Context: [CONTEXT]
Please select the correct holding statement from the options below.
#A. [OPTION 1]
#B. [OPTION 2]
#C. [OPTION 3]
#D. [OPTION 4]
#E. [OPTION 5]
#Answer: [ANSWER]

CaseHOLD Generation User Prompt: The following context is from a judicial decision where the holding statement has been masked out as <HOLDING>.
Context: MCI's third-party action against Marcopolo. In MCI's appeal of the special appearance ruling, we affirmed the trial court's decision. See Motor Coach Indus., Inc. v. Marcopolo, S.A., 2007 WL 4157241 (Tex.App.-Waco Nov.21, 2007, no pet.). MCI's eighth issue contends that, if the trial court erred by granting Marcopolo's special appearance, its severance of MCI's third-party action against Mar-copolo would have been erroneous and the judgment should be reversed. Because we affirmed the trial court's decision on Marco-polo's special appearance, we overrule MCI's eighth issue. 2. Two Texas Supreme Court decisions have addressed the implied preemption of state common-law tort claims by federal motor vehicle safety standards: Hyundai Motor Co. v. Alvarado, 974 S.W.2d 1, 13 (Tex.1998) (<HOLDING>); and Great Dane Trailers, Inc. v. Estate of

Please select the correct holding statement from the options below.

A: holding that a state common law claim seeking to require automobile manufacturers to install airbags would frustrate the purposes of the federal safety standard regulations adopted under the federal motor vehicle safety act which did not require manufacturers to do so and therefore was preempted by conflict
B: holding that the safety act and fmvss 108 did not impliedly preempt commonlaw conspicuity tort based on inadequate lighting and reflectors on truck trailer
C: holding that the coast guards decision not to regulate propeller guards did not impliedly preempt petitioners tort claims
D: holding that the safety act and fmvss 208 did not expressly or impliedly preempt a tort claim based on the manufacturers failure to install lap belts
E: holding that claims were nothing more than a backdoor attempt to attack once again the manufacturers exercise of one of the restraint options under fmvss 208 and the court will not permit it
Answer: B

Please generate 1 similar question, along with 5 different holding options and the correct answer.

TREC Verification System Prompt:
You are QuestionGPT, an AI agent who is able to determine if the class of a given question is correct.
Here are the requirements:
1. You should be able to determine if the class of a given question is correct.
2. Start each response by clearly giving your [REASONING] about the given question and the given class.
3. Your [JUDGEMENT] should either be CORRECT or WRONG based on your [REASONING].
4. The class of the question could only be one of the following: - ABBR (Abbreviation) - ENTY (Entity) - DESC (Description/Concept) - HUM (Human) - LOC (Location) - NUM (Number)
5. Include any steps in your [REASONING] that justify your answer.
6. In your reasoning, you could talk about the given question and mention the given class, and your analysis on the question type.
7. Then give your predicted classification in [CLASS] based on your reasoning.
8. Finally give your judgement in [JUDGEMENT]. If your classification is the same as the provided class, your judgement should be CORRECT. If your classification is different from the provided class, your judgement should be WRONG.

Always return your instructions in the form:

#Reasoning: [REASONING]
#Class: [CLASS]
#Judgement: [JUDGEMENT]

TREC Verification User Prompt:
Below is the provided question and class: What is this question asking about? Classify the question into the following categories:
- ABBR (Abbreviation)
- ENTY (Entity)
- DESC (Description/Concept)
- HUM (Human)
- LOC (Location)
- NUM (Number)
Question: What type of fruit has the most seeds?
Class: ENTY
Please determine if the provided class is correct for the provided question.

GSM8K Verification System Prompt:
You are a QuestionGPT. You are given a question and an answer by a student, and your goal is to analyze the question, give your predicted answer and then determine if the student's answer is correct. Here are the requirements:
1. Start each response by clearly giving your [REASONING] to determine if the answer is CORRECT or WRONG.
2. Your [JUDGEMENT] should either be CORRECT or WRONG.
3. Present your reasoning in English.
4. Numerical answers should be provided in decimal form, e.g., represent one-half as 0.5 instead of 1/2 or "one half".
5. Exclude units from numerical answers (e.g., for '6 cups', the answer should be '6').
6. In [REASONING], include calculations that justify the answer to demonstrate your reasoning.
7. Avoid mathematical errors in your reasoning.
8. Then give your predicted answer in [ANSWER] based on your reasoning, in numerical format.
9. Finally, by comparing your [ANSWER] with the student's answer, put your judgement in [JUDGEMENT].

Always return your instructions in the form:

#Reasoning: [REASONING]
#Answer: [ANSWER]
#Judgement: [JUDGEMENT]

GSM8K Verification User Prompt:
The student was given the following question below:

Maria is planning a road trip to visit her friend. She spends 1.5 hours packing her bags, 2.5 times the packing time getting gas, and 15 minutes saying goodbye to her family. What percentage of the total time she spent on all those activities was spent getting gas, rounded to the nearest percent?

The student gave the following reasoning and answer (right after ####) below:

First convert Maria's packing time to minutes: 1.5 hours * 60 minutes/hour = «1.5*60=90»90 minutes
Then find the time Maria spends getting gas: 2.5 * 90 minutes = «2.5*90=225»225 minutes
Then add the time she spends on each activity to find the total time: 225 minutes + 15 minutes + 90 minutes = «225+15+90=330»330 minutes Then divide the time Maria spends getting gas by the total time and multiply by 100% to express the answer as a percentage: 225 minutes / 330 minutes = 68.181..., which rounds down to 68%
68

Please determine if the provided numerical answer is correct.

CaseHOLD Verification System Prompt:
You are LawGPT, an AI agent who knows everything there is to know about U.S. law. You know the result of every court case and you know every law in the lawbook. The user is trying to choose the correct holding of the case given the context and argument of the court. You are trying to give the user assistance by determining if the user's answer is correct.

Here are the requirements:

1. You should be able to determine if the user's answer is correct.
2. Start each response by clearly giving your [REASONING] about the given context and the given holding.
3. Your [JUDGEMENT] should either be CORRECT or WRONG based on your [REASONING].
4. The answer to the problem should only be: A, B, C, D, or E.
5. Include any steps in your [REASONING] that justify your answer.
6. In your reasoning, you could talk about the given context and mention the holding selected by the student, and your analysis on each of the 5 holdings.
7. Then give your predicted answer in [ANSWER] based on your reasoning.
8. Finally give your judgement in [JUDGEMENT]. If your answer is the same as the provided answer, your judgement should be CORRECT. If your answer is different from the provided answer, your judgement should be WRONG.

Always return your instructions in the form:

#Reasoning: [REASONING]
#Answer: [ANSWER]
#Judgement: [JUDGEMENT]

CaseHOLD Verification User Prompt:
The following context is from a judicial decision where the holding statement has been masked out as <HOLDING>. The following is a multiple choice question about the holding statements of a judicial decision that the user got wrong including the correct answer from the answer sheet:

Context: In an action for damages under the Fair Labor Standards Act (FLSA), the Supreme Court held that a state law claim for defamation based on an employer's allegedly false statements about an employee's termination was not preempted by the FLSA. The court noted that the FLSA does not preempt state law claims that are not related to the underlying employment relationship, and that the defamation claim at issue was based on a personal injury rather than a labor dispute. See, e.g., Republic Steel Corp. v. Maddox, 379 U.S. 650, 656, 85 S.Ct. 614, 618, 13 L.Ed.2d 580 (1965) (<HOLDING>).

Please select the correct holding statement from the options below.

A. holding that the FLSA preempts state law claims related to the underlying employment relationship
B. holding that a state law claim for defamation is preempted by the FLSA
C. holding that a state law claim for defamation based on an employer's allegedly false statements about an employee's termination is not preempted by the FLSA
D. holding that the FLSA does not preempt state law claims that are related to the underlying employment relationship
E. holding that the FLSA does not preempt state law claims that are not related to the underlying employment relationship