Data-Efficient Adaptation of LLMs via Attention Head Reweighting

Tuomas Oikarinen
UC San Diego, Microsoft Research

Zixiao Chen
Microsoft Security AI

Charlotte Siska
Microsoft Security AI

Tsui-Wei Weng
UC San Diego

Chandan Singh
Microsoft Research

Jianfeng Gao
Microsoft Research

Abstract

Learning effectively from limited data is critical in domains like security where labeled examples are scarce. Large language models (LLMs) have demonstrated some capabilities for data-efficient learning, especially through parameter-efficient adaptation methods, but continue to struggle when faced with few samples for difficult tasks. To meet this challenge, we propose Attention Head Reweighting (AHR), a data-efficient method that adapts LLMs to new text-classification tasks by learning only a single scalar per attention head. This drastically reduces the number of parameters that need to be learned by making use of the functional specialization of individual attention heads. Experiments on diverse open-source text classification datasets show that AHR can outperform standard baselines like LoRA when learning from limited samples, despite having $200 - 1000\times$ fewer trainable parameters, as our AHR only modifies $\sim 0.0001\%$ of the model's parameters. In addition, our learned weights are easy to interpret and can be analyzed to better understand the mechanisms and attention heads responsible for in-context learning abilities in LLMs.[1]

1 Introduction

Learning effectively from limited labeled data remains a central challenge in high-stakes domains such as AI security, where annotations are often scarce because attacks and threat vectors are constantly evolving and need to be defended against quickly (Divakaran & Peddinti, 2024). In this setting, data-efficient text classification is critical for flagging large language model (LLM) inputs (Yi et al., 2024; Saha et al., 2024; Chao et al., 2025; Verma et al., 2025) and LLM outputs (Inan et al., 2023) that can be dangerous.

While LLMs exhibit some data-efficient behavior through in-context learning (ICL) (Brown et al., 2020; OpenAI, 2023) and parameter-efficient fine-tuning (PEFT) (Hu et al., 2022; Zhang et al., 2023a), their performance often degrades sharply when only a small number of task-specific examples are available. One issue with these methods is that they generally optimize in a large, unconstrained parameter space, which can lead to overfitting.

To tackle this issue, we propose Attention Head Reweighting (AHR), an intuitive data-efficient method to adapt LLMs to new text-classification tasks by learning only a single scalar per attention head. This drastically reduces the number of parameters that need to be adapted (see Fig. 1). We hypothesize our method is effective and avoids overfitting by leveraging the functional specialization of attention heads noted in recent works (Olsson et al., 2022; Akyürek et al., 2024; Zhang et al., 2024a; Ge et al., 2023). For example, if a particular security-related behavior is isolated to a few attention heads, AHR can easily learn to change the weighting for these heads rather than seeking to alter the internal weights of each head.

[1]Code to use AHR (fully compatible with the peft package) and to reproduce our experiments is available on Github at github.com/tuomaso/attention-head-reweighting

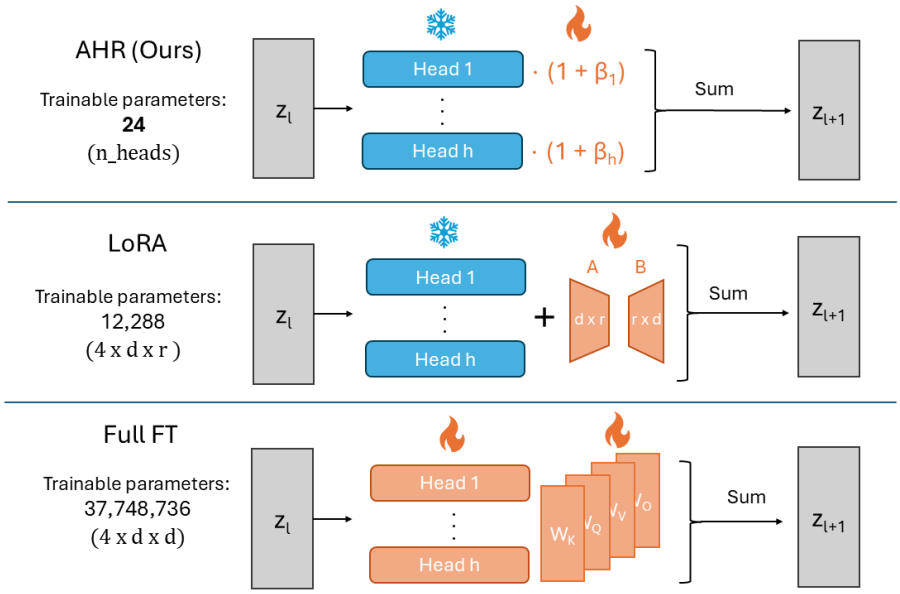


Figure 1: *AHR overview*. Simplified visual comparison between our AHR, LoRA and full-finetuning on a single attention layer. Parameter counts are calculated for a standard transformer with $d = 3072$, with $r = 1$ for LoRA. We can see AHR reduces trainable parameters $500\times$ compared to LoRA.

In our experiments we focus on text classification tasks, as these can often be effectively learned from a few examples, and many important security tasks such as detecting phishing attempts or model jailbreaks are text classification tasks. Classifiers are a critical bottleneck in AI security, which is constantly faced with data scarcity as new threats emerge, for filtering and guarding LLM responses that are interacting with real-world users and attackers.

In our experiments, AHR outperforms baselines across a variety of settings and models, improving by an average of around 3% points over standard baselines like LoRA (Hu et al., 2022) when training data is very limited i.e. ≤ 100 examples. AHR shows particularly large gains in the security relevant tasks of Webpage Phishing Detection (Web) and Jailbreak detection, achieving 6-7% accuracy improvement over best baseline methods with just 10 training samples, while using $200 - 1000\times$ fewer trainable parameters. We can analyze the changes our method makes to the model, and see that these gains are largely fueled by a combination of upweighting a few task specific heads as well as a couple general in-context learning heads. Finally, to maximally learn from just a few examples, we utilize in-context finetuning (IC-FT), where we update model parameters to maximize the probability of correct answers within a few-shot prompt, which we show improves accuracy by $\sim 10\%$ points over finetuning on just one sample at a time.

2 Background and related work

Parameter-efficient finetuning A variety of parameter-efficient finetuning methods have been proposed to adapt large models while updating only a small subset of parameters (Zhang et al., 2025a; Han et al., 2024; Wang et al., 2024). Prominent examples include LoRA (Hu et al., 2022), AdaLoRA (Zhang et al., 2023a), $(IA)^3$ (Liu et al., 2022), and related approaches (Hayou et al., 2024; 2025; Logan IV et al., 2022). These methods substantially reduce adaptation cost, but still introduce thousands to millions of trainable parameters, which can lead to overfitting in data-scarce settings.

Simultaneously, there has been a spectrum of increasingly constrained prompting approaches, ranging from continuous prompt tuning (Li & Liang, 2021; Liu et al., 2021) to discrete prompt tuning (Shin et al., 2020), prompt ensemble construction (Hou et al., 2022;

Model	GPT2-XL	Llama-3.2-1B	Llama-3.2-3B	QWEN3-8B
Full FT	1,558,148,000	1,235,920,896	3,212,750,496	8,190,736,512
LoRA	307,200	106,496	286,720	479,232
AdaLoRA	614,496	213,056	573,552	958,608
IA3	537,600	147,456	286,720	626,688
AHR (Ours)	**1,200**	**512**	**672**	**1,152**

Table 1: Comparing the number of trainable parameters for different models and different parameter efficient finetuning methods. Our AHR requires $200 - 1,000\times$ less trainable parameters than the baseline methods. In total we train less than one millionth of the model's parameters.

Pitis et al., 2023; Morris et al., 2023), and generating a single natural-language prompt (Zhou et al., 2022; Singh et al., 2023b). While effective in some settings, prompt-based methods are often very sensitive to minor variations and may struggle with under-specified tasks under severe data scarcity.

In-context learning ICL obviates the need for task-specific parameter adaptation altogether, but can be highly sensitive to minor variations in the provided examples (Min et al., 2022) and sometimes unfaithful to those examples (Wei et al., 2023; Webson & Pavlick, 2022). ICL can be improved through directed instruction tuning (Chung et al., 2024) or by including explanations along with examples (Lampinen et al., 2022). A growing body of work has studied ICL, showing that it can implement linear models (Akyürek et al., 2022; Zhang et al., 2023b), discrete functions (Bhattamishra et al., 2023), and more general algorithms (Li et al., 2023; Zhuang et al., 2025). Some works have further argued that ICL implicitly performs optimization steps analogous to gradient descent (Mahankali et al., 2023; Von Oswald et al., 2023; Ahn et al., 2024) and higher-order optimization methods (Dai et al., 2023; Zhang et al., 2023b).

Attention heads Attention heads have been a primary focus for analysis (Zheng et al., 2024; Bibal et al., 2022), with evidence that attention heads often latch onto task-specific patterns that are useful for ICL, e.g. induction heads (Olsson et al., 2022), n-gram heads (Akyürek et al., 2024), and more easily explainable patterns (Todd et al., 2025; Oikarinen & Weng, 2024; Singh et al., 2023a). This growing evidence of functional specialization has motivated methods that explicitly manipulate attention. Prior works have shown that modifying attention scores can be useful for post-hoc steering (Zhang et al., 2024a;b), attention steering for transfer learning (Shi et al., 2023), attention-guided retrieval (Zhang et al., 2025b), interpretable modeling (Kim et al., 2024), boosting attention for instruction following (Guardieiro et al., 2025), and improving long-context LLM inference (Gu et al., 2024).

3 Method: Attention Head Reweighting

Our paper aims to improve the efficiency of how models can learn from a very small number of training examples by modifying an extremely small number of parameters. We do this via **Attention Head Reweighting (AHR)**, which introduces a single learnable scalar parameter per attention head that multiplies that head's contributions to the residual stream (see Fig. 1). By finetuning these parameters on a small amount of data, we can amplify attention heads that are helpful for the current task, while downweighting heads that harm performance. By not modifying the weights inside individual heads we can avoid overfitting to the small amount of training data.

Notation: We follow the notation of Elhage et al. (2021), where we divide the W_O matrix into its components within each attention head. Let $z_r^l \in \mathbb{R}^{s\times d}$ be the model's residual stream after layer l, where s is the number of tokens in the sequence and d is the model's representation dimension. A typical transformer block consists of a Multihead-Attention

(MHA) layer followed by a multi-layer perceptron (MLP) layer. We can write the update of a layer as:

$$z_{Att}^{l+1} = z_r^l + \sum_{h \in H^{l+1}} h(z_r^l) \tag{1}$$

$$z_r^{l+1} = z_{Att}^{l+1} + MLP^{l+1}(z_{Att}^{l+1}) \tag{2}$$

Here $h(z) = AzW_VW_O$, $A = \text{softmax}(\frac{(zW_Q)(zW_K)^T}{\sqrt{d_k}})$. $W_V \in \mathbb{R}^{d \times d_k}$, $W_Q \in \mathbb{R}^{d \times d_k}$, $W_K \in \mathbb{R}^{d \times d_k}$ and $W_O \in \mathbb{R}^{d_k \times d}$ are the parameters of the attention head, and $d_k = d/n_h$ where $n_h = |\mathcal{H}|$ is the number of attention heads per layer.

Attention Head Reweighting: Our method simply edits the update rule of the multi-head attention layer (Eq. 1) to:

$$z_{Att}^{l+1} = z_r^l + \sum_{h \in H^{l+1}} (1 + \beta_h) h(z_r^l), \tag{3}$$

where β_h is a head-specific parameter that is initialized at zero. During training, we only train the β_h parameters while keeping all other parameters, including the MLP layers unchanged. At test time, the parameters β can be merged into W_O for no additional inference cost. During training, we utilize supervised fine-tuning (SFT) on few-shot examples, by minimizing the cross-entropy loss on the answer tokens, as well as a regularization objective to minimize changes to the model:

$$\min_{\beta} \sum_{(x,y) \in \mathcal{D}_{train}} -\log P(y|x; \theta, \beta) + \lambda (\sum_h |\beta_h|^p)^{1/p} \tag{4}$$

where $\mathcal{D}_{train}$ is the few-shot training dataset, λ is a hyperparameter controlling regularization and θ are the (frozen) model parameters and $p \in \{1, 2\}$.

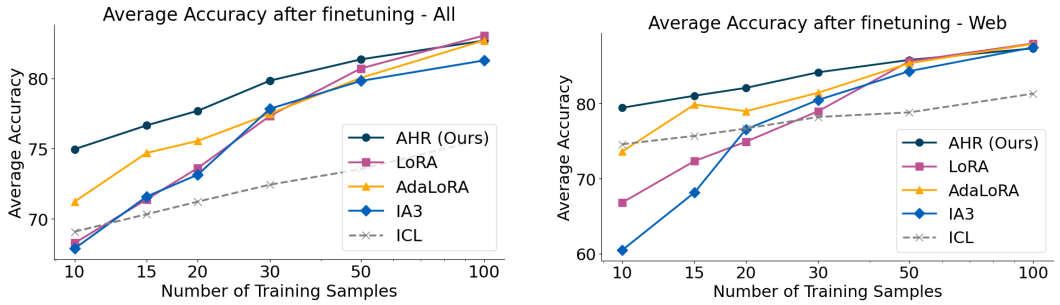
3.1 In-Context Finetuning

Recently, In-Context Learning (ICL) has risen as an alternative to finetuning as a way for models to learn from few-shot examples (Brown et al., 2020). While powerful, in-context learning is limited in how many samples it can learn from by the model's context window, as well as GPU-memory. Additionally, using many in-context samples incurs a large additional cost at inference time.

To get the best of both worlds, we utilize In-Context finetuning (IC-FT) similar to (He et al., 2025). Different from (He et al., 2025), we finetune separately on each dataset while their goal is to improve general in-context learning ability. In in-context finetuning, we construct an input string with k few-shot examples, and do supervised finetuning to maximize the probability of all answer tokens within the context. For each input with k in-context examples, we minimize the following:

$$\min_{\beta} \sum_{i=1}^{k} -\log P(y_i|X_i; \theta, \beta) + \lambda (\sum_h |\beta_h|^p)^{1/p} \tag{5}$$

where $X_i = \{x_1, y_1, ..., y_{i-1}, x_i\}$ is the concatenated input with few-shot examples. As this technique is independent of the finetuning method, we utilize it for both our method and the baselines in our experiments (with their trainable parameters instead of β in Eq. 5). For text classification, we only maximize the probability of the first token in the answer, as predicting subsequent tokens of the answer is trivial given the first one.



(a) Averaged across 6 datasets

(b) *Web* - Phishing URL classification

Figure 2: Accuracy after finetuning, averaged across our 4 models. We can see our AHR significantly outperforms baselines when data is limited, while performance starts to equalize at around 100 samples.

4 Results

4.1 Experimental setup

Models We consider four LLMs across a range of sizes in our experiments: GPT-2 XL (Radford et al. 2019; `gpt2-xl`), LLaMa-3.2 1B and 3B (Dubey et al. 2024; `meta-llama/Llama-3.2-1B` and `meta-llama/Llama-3.2-3B`), and Qwen3 8B (Yang et al. 2025; `Qwen/Qwen3-8B`). See Table 1 for more details, such as the number of parameters in each model and how many of these are trainable.

Datasets We test our method on a set of six text-classification datasets, including standard tasks as well as cybersecurity relevant applications:

- SST2 (Socher et al., 2013). This dataset contains sentences from movie reviews, and the task is to classify their sentiment into one of {Positive, Negative}.
- AG-News (Zhang et al., 2015). A collection of text excerpts from news articles, where the goal is to classify the excerpt into one of the following categories: {World, Sports, Business, Sci/Tech}
- Emotion (Saravia et al., 2018). A dataset of tweets, with labels based on the emotion expressed in the tweet. This dataset contains 6 classes {Sadness, Joy, Love, Anger, Fear, Surprise}
- Webpage Phishing Detection (Web) (Hannousse & Yahiouche, 2021), a dataset of web URLs where the goal is to differentiate between Phishing and Legitimate URLs.
- Toxigen (Hartvigsen et al., 2022). This is a dataset of potentially toxic/hateful comments for detecting hateful speech. We treated an input as toxic if the average human assessed toxicity score was > 3 (on a 1-5 scale) and as benign otherwise.
- Jailbreak Detection (Shen et al., 2024), a dataset of LLM prompts containing Jailbreak attempts as well as standard prompts. The goal is to classify whether a given prompt is a jailbreak attempt. We use the balanced version from https://huggingface.co/datasets/jackhhao/jailbreak-classification.

Most of these datasets have thousands of examples, but in order to study few-shot learning capabilities of our models, we randomly sample a subset of the training data to be used in few-shot learning. We evaluate performance on the full test set. See an example for each dataset in Table A.2.

Baselines: As our main baseline, we compare against LoRA (Hu et al., 2022). For LoRA we use $r = 1$ and $\alpha = 1$, which we found to increase training stability and reduce overfitting on small datasets. We additionally compare against $(IA)^3$ (Liu et al., 2022) and AdaLoRA (Zhang et al., 2023a). For AdaLoRA we use Initial $r = 4$ and target $r = 2$, with $\alpha = 1$. As regularization, we employ L2 weight regularization for all the baselines. For training the baselines, we utilize PEFT package (Mangrulkar et al., 2022). We also ran full-finetuning for

some datasets but found that it performed considerably worse than these baselines and was significantly more computationally intensive, so we omit these results.

Training Details: In our experiments, we discovered that the particular few-shot examples used can have a very large effect on the final accuracy. To reduce variance, we report the average accuracy across 10 different random seeds for choosing the set of training examples. We utilize In-Context Finetuning (Sec 3.1) for training all the baseline as well as our AHR. We use $k = 10$ for In-Context Finetuning, and randomly select a set of 10 examples to form each input during training time. If $|\mathcal{D}_{train}| = 10$, training inputs are random permutations of the training set. We train all models using the AdamW optimizer (Loshchilov & Hutter, 2019).

Hyperparameter Selection To stay true to a real data-limited setting, we do not utilize any additional data for selecting hyperparameters. Instead, we select hyperparameters including learning rate from $\{0.03, 0.003, 0.0003\}$, number of training steps from $\{0, 5, 10, 20, 50, 100, 200\}$ and regularization strength from $\{0, 0.1, 0.3, 1.0, 3.0, 10.0, 30.0, 100.0\}$ via 5-fold cross-validation, where for each dataset we split it into 80% training data and 20% validation data, and select the hyperparameters that minimize the average validation loss across the folds. We select hyperparameters separately for each random seed. Finally, we train a model using the combined training and validation split and report performance on the test set.

Our prompts consists of a short instruction describing the task and the class names, as well as the few-shot examples, see exact prompt in Appendix Table A.3.

4.2 Experimental results

Fig. 2 and Table 2 show the average accuracy after finetuning as a function of training data size $|\mathcal{D}_{train}|$. We can see that our AHR consistently outperforms existing parameter efficient finetuning methods when training data is limited. On average, our AHR improves over the best baseline by 2-4%(absolute) in low data regimes ($|\mathcal{D}_{train}| \leq 30$), which corresponds to around 10% relative error reduction, despite having over 200× fewer trainable parameters. This shows our AHR can significantly improve the few-shot learning ability of the models.

AHR performs particularly well on the web URL phishing classification task, substantially outperforming LoRA and other baselines. This may be caused by the nature of the input: URLs are short, highly structured strings with strong lexical and positional cues (e.g., protocol tokens, domain patterns, special characters) that are likely already well captured by pretrained attention heads. In this setting, effective adaptation primarily requires emphasizing a subset of existing attention heads that are sensitive to these patterns, rather than learning new task-specific transformations. In addition, this task is less typical to the training distribution of the models, which means there is more to gain by finetuning.

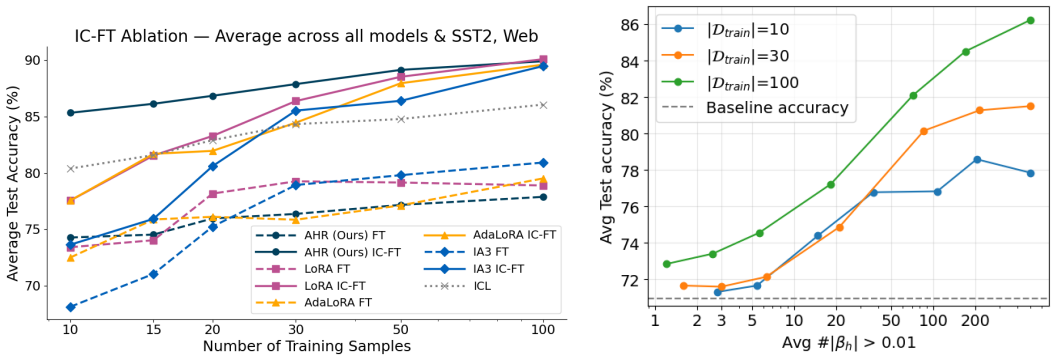
In contrast, most PEFT methods such as LoRA actually harm performance compared to the In-Context Learning baseline when $|\mathcal{D}_{train}|$ is small as shown in Fig 2b. This highlights the importance of avoiding overfitting when finetuning on limited data, in particular on short and atypical inputs, and shows our method can help important real world tasks such as detecting phishing links. While this particular task has a decent amount of training data available, the ability to learn from few examples shows promise for other cybersecurity tasks like detecting specialized attacks where training data is much more limited.

4.3 Ablations

IC-FT: Fig. 3a compares test accuracy while using In-Context Finetuning described in Section 3.1 against standard finetuning without in-context samples. Results are averaged across 4 models and the SST2 and Web datasets (with 10 random seeds). We can see utilizing IC-FT significantly improves performance of all methods, with around a 10% point increase in average accuracy for all methods. Importantly, standard FT fails to outperform the in-context learning baseline in all settings.

Dataset	$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
SST2	LoRA	88.32%	90.24%	88.75%	**92.09%**	91.44%	92.20%
	AdaLoRA	87.43%	90.40%	89.54%	91.00%	92.18%	92.70%
	IA3	88.75%	89.15%	89.01%	92.08%	**92.54%**	92.60%
	AHR (Ours)	**91.01%**	**91.25%**	**91.62%**	92.07%	92.49%	**92.99%**
AG News	LoRA	**78.18%**	77.35%	81.24%	81.45%	84.20%	84.90%
	AdaLoRA	76.58%	77.33%	80.06%	81.76%	82.18%	84.00%
	IA3	75.68%	**79.36%**	80.64%	82.50%	83.96%	**86.62%**
	AHR (Ours)	76.48%	79.19%	**81.60%**	**83.06%**	**84.51%**	85.65%
Emotion	LoRA	49.81%	53.01%	50.65%	56.25%	57.94%	63.05%
	AdaLoRA	**52.54%**	53.26%	52.10%	**57.18%**	**58.88%**	**64.94%**
	IA3	49.83%	52.90%	51.69%	56.18%	58.33%	61.03%
	AHR (Ours)	50.32%	**53.74%**	**54.48%**	56.67%	58.78%	62.57%
Web	LoRA	66.81%	72.34%	74.91%	78.99%	85.63%	**88.00%**
	AdaLoRA	73.60%	79.86%	78.98%	81.44%	85.37%	87.94%
	IA3	60.47%	68.15%	76.56%	80.48%	84.32%	87.48%
	AHR (Ours)	**79.44%**	**81.04%**	**82.08%**	**84.16%**	**85.81%**	87.35%
Toxigen	LoRA	56.94%	60.60%	64.10%	70.25%	73.83%	75.86%
	AdaLoRA	64.84%	69.69%	72.00%	**73.94%**	74.92%	**76.79%**
	IA3	59.64%	64.10%	63.86%	71.13%	74.21%	75.56%
	AHR (Ours)	**72.78%**	**72.52%**	**72.71%**	73.76%	**75.32%**	74.69%
Jailbreak	LoRA	69.59%	74.76%	82.05%	84.92%	**91.16%**	**94.26%**
	AdaLoRA	72.33%	77.59%	80.60%	79.53%	86.63%	89.82%
	IA3	72.94%	75.86%	77.10%	84.61%	85.50%	84.32%
	AHR (Ours)	**79.68%**	**82.16%**	**83.63%**	**89.26%**	**91.16%**	92.83%
Average	LoRA	68.27%	71.38%	73.62%	77.32%	80.70%	**83.05%**
	AdaLoRA	71.22%	74.69%	75.55%	77.47%	80.03%	82.70%
	IA3	67.88%	71.58%	73.14%	77.83%	79.81%	81.27%
	AHR (Ours)	**74.95%**	**76.65%**	**77.69%**	**79.83%**	**81.34%**	82.68%

Table 2: Performance of different methods on each dataset. Averaged across the 4 models. While there is variance between the individual datasets, we can see that AHR consistently outperforms other methods in the low data regime.



(a) Effect of IC-FT (solid lines) vs direct finetuning with only one sample in context(dashed).

(b) *Web* Test accuracy as a function of number of heads modified.

Figure 3: a) IC-FT ablation. b) *Web* Accuracy with L1 regularization.

Regularization Comparison: In Table A.1 we compare different regularization strategies for AHR on SST2 LLama-3.2-3B. We can see this makes very little difference in performance. As a result, we use L2-norm for our results unless otherwise specified, though we use L1-norm for our analysis in Section 5.

5 Case Study: Finetuning Llama-3.2-1B on Web

In this section, we take a deeper look at finetuning one model on one particular task to better understand AHR and how it differs from other PEFT methods.

5.1 Number of Attention Heads Modified

First, to understand whether AHR needs to modify many attention heads, we train models with varying L1 regularization penalties {0, 0.005, 0.01, 0.02, 0.03, 0.04, 0.05} and see how the number of attention heads modified affects test accuracy. The results with Llama-3.2-1B trained on the Web dataset, average of 10 random seeds are shown in Fig. 3b. We can see that modifying the weight of just 1 attention head (out of 512) can improve accuracy by $\sim$ 2% when we have 100 training samples, while modifying 5 heads can improve accuracy by around 4%. Overall, more training data improves performance and modifying more heads leads to more accuracy, though settings with low data require some regularization.

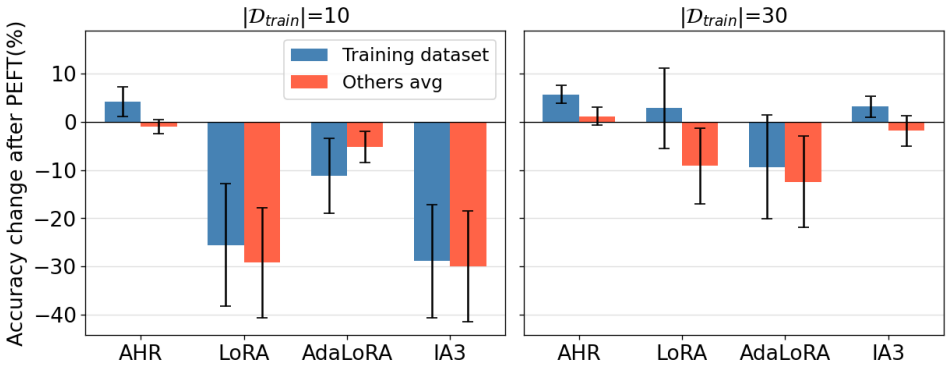


Figure 4: The effect of finetuning on other tasks performance. Llama-3.2-1B finetuned on the Web (phishing URL classification) task.

5.2 Effect on Overall Model Performance

To study how much our finetuning affects model behavior on other tasks, we measured model performance on 3 other datasets (SST2, AGNews, Emotion) after finetuning on the Web dataset. In Fig. 4 we measure the performance on the Web dataset (blue) as well as average over our other datasets (red). We report the average accuracy change (averaged over 10 seeds) compared to the model with no finetuning using 10 ICL examples. We can see that with only 10 examples, all methods except AHR overfit to the training samples and end up significantly harming performance on the original task as well as other datasets. When we increase the training set size to 30, we see most methods improve task performance on the *Web* task, but at the cost of losing some accuracy on other tasks. Interestingly, with AHR, finetuning on this task actually slightly *improves* performance on other tasks. This suggests that AHR helps avoid destructive changes to the model, and in some cases can even find generalizable edits that improve ICL abilities across many tasks.

5.3 Which heads are modified?

Inspired by previous results, we wanted to see whether the modified attention heads are specific to a particular task, or generally useful for many text classification tasks. To study this, we trained AHR models with varying L1 penalties $\{0.01, 0.02, 0.03, 0.04, 0.05\}$ on 4 of our tasks (SST2, AGNews, Emotion and Web). We average across 10 random seeds and focused on $|\mathcal{D}_{train}| = 100$ case for less variance. We then measured which heads had the largest average change across different settings, and plot a heatmap of most changed heads in Fig. 5.

This reveals a few interesting things. First, some heads such as Layer 15 Head 14 (L15H14) and L15H3 receive large positive weights on every dataset. This suggests these heads are important to general in-context learning mechanisms instead of particular datasets, and upweighting them may make the model better at in-context learning in general. This could happen if, for example, they play a role similar to induction heads (Olsson et al., 2022). To test this theory we conducted an experiment where we manually modify the weights β for these two heads only, leaving all other heads unchanged ($\beta = 0$). The results are shown in Appendix A.2 and Fig. A.3, and support our hypothesis. Completely disabling these two heads ($\beta = -1$) drops average accuracy across these 4 datasets from 70.68% to 66.81%, while doubling their impact ($\beta = 1$) increases average accuracy to 71.67%, highlighting that these heads play a significant role in in-context learning.

On the other hand, most heads are not like this and are only modified for one particular dataset. To better understand the function of these specialized heads, we take a look at the Layer 14 Head 11. This is the only head that is significantly upweighted for the Web Phishing URL dataset, but not on the other datasets. Fig. A.1 shows the attention pattern of this head when processing the final token of the input on the Web dataset. We can see that on all inputs this head only pays attention to the *Phishing* tokens. In comparison, in

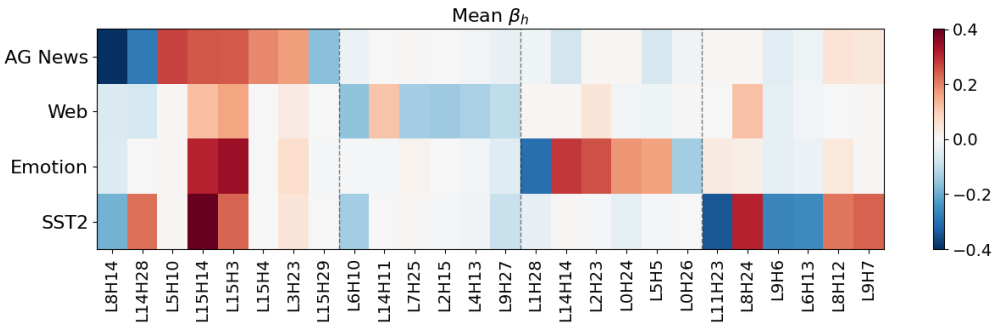


Figure 5: Heatmap of average β_h for the most modified heads across different datasets. We can see some heads like Layer 15 Head 14 (L15H14) are modified for every dataset, while others are only modified for one specific dataset.

Fig. A.2 we plot its attention pattern on the other datasets and find it very noisy with no clear pattern. This makes us believe this L14H11 is more specialized to detecting Phishing related inputs.

6 Discussion

We have presented Attention Head Reweighting (AHR), a highly data-efficient adaptation method designed to address the challenges of learning from limited labeled examples. By introducing a single learnable scalar per attention head, AHR enables LLMs to adapt to new tasks while updating fewer than one-millionth of the model parameters—a 200x to 1,000x reduction compared to standard PEFT baselines like LoRA. Our experimental results on diverse text-classification benchmarks demonstrate the efficacy of this approach:

- **Few-shot Learning Performance:** AHR consistently outperforms existing methods in extremely data-scarce settings ($|\mathcal{D}_{train}| \leq 30$), achieving a 2-4% average absolute improvement over baseline methods like LoRA, with large improvements in security relevant tasks like detecting Phishing URLs and Jailbreak attempts.
- **Preventing Overfitting:** By freezing internal head weights and only modifying the contribution of the head to the residual stream, AHR mitigates the risk of overfitting that hampers methods with larger parameter spaces and is overall less destructive to model behavior.

Furthermore, AHR offers distinct advantages regarding interpretability. Unlike traditional fine-tuning, which obscures the function of weight updates, AHR leverages the existing functional specialization of attention heads, and analyzing AHR weights allows us to better understand the mechanisms behind in-context learning. Ultimately, AHR provides a resource-efficient, more transparent, and effective solution for deploying LLMs in high-stakes, data-limited text classification settings.

6.1 Limitations

The focus of our paper is on text classification datasets, and while many important tasks such as detecting phishing links or jailbreak attempts are text classification, the applicability of our method on tasks beyond classification requires further investigation. In our initial investigations we found AHR more effective at tasks with simple outputs such as classification, while tasks that require very fine-grained control over model outputs might be less suitable for AHR, particularly if there are no attention heads focused on this type of task originally.

Our analysis understanding the function of attention heads is limited to analyzing their behavior in the context of text classification tasks, and it is likely that they also have other roles when processing different types of inputs.

Larger training datasets: Overall we find that if you have sufficient training data (e.g. ≥ 300 examples) other finetuning methods outperform AHR. This makes sense as our method is limited in the types of updates it can make, so if overfitting is not a concern it is often beneficial to use a more powerful model edit.

Acknowledgement

T. Oikarinen and T.-W. Weng are partially supported by National Science Foundation under Grant No. 2313105, 2430539, Hellman Fellowship, Intel Rising Star Faculty Award. The authors would like to thank anonymous reviewers for valuable feedback to improve the manuscript.

References

Kwangjun Ahn, Xiang Cheng, Hadi Daneshmand, and Suvrit Sra. Transformers learn to implement preconditioned gradient descent for in-context learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*, 2022.

Ekin Akyürek, Bailin Wang, Yoon Kim, and Jacob Andreas. In-context language learning: Architectures and algorithms. *arXiv preprint arXiv:2401.12973*, 2024.

Dana Arad, Aaron Mueller, and Yonatan Belinkov. Saes are good for steering–if you select the right features. *arXiv preprint arXiv:2505.20063*, 2025.

Elad Ben-Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *ArXiv*, abs/2106.10199, 2021.

Satwik Bhattamishra, Arkil Patel, Phil Blunsom, and Varun Kanade. Understanding in-context learning in transformers and llms by learning to learn discrete functions. *arXiv preprint arXiv:2310.03016*, 2023.

Adrien Bibal, Rémi Cardon, David Alfter, Rodrigo Wilkens, Xiaoou Wang, Thomas François, and Patrick Watrin. Is attention explanation? an introduction to the debate. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3889–3900, 2022.

Steven Bills, Nick Cammarata, Dan Mossing, Henk Tillman, Leo Gao, Gabriel Goh, Ilya Sutskever, Jan Leike, Jeff Wu, and William Saunders. Language models can explain neurons in language models, 2023. URL https://openaipublic.blob.core.windows.net/neuron-explainer/paper/index.html.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Sviatoslav Chalnev, Matthew Siu, and Arthur Conmy. Improving steering vectors by targeting sparse autoencoder features. *arXiv preprint arXiv:2411.02193*, 2024.

Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries. In *2025 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pp. 23–42. IEEE, 2025.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-lora: Instant transformer adaption. *arXiv preprint arXiv:2506.06105*, 2025.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

Arthur Conmy, Augustine Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. *Advances in Neural Information Processing Systems*, 36:16318–16352, 2023.

Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. Why can gpt learn in-context? language models secretly perform gradient descent as meta-optimizers. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 4005–4019, 2023.

Dinil Mon Divakaran and Sai Teja Peddinti. Llms for cyber security: New opportunities. *arXiv preprint arXiv:2404.11338*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 1, 2021.

Jean Feng, Avni Kothari, Patrick Vossler, Andrew Bishara, Lucas Zier, Newton Addo, Aaron Kornblith, Yan Shuo Tan, and Chandan Singh. Human-ai co-design for clinical prediction models. *arXiv preprint arXiv:2601.09072*, 2026.

Leo Gao, Achyuta Rajaram, Jacob Coxon, Soham V Govande, Bowen Baker, and Dan Mossing. Weight-sparse transformers have interpretable circuits. *arXiv preprint arXiv:2511.13653*, 2025.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.

Zhuohan Gu, Jiayi Yao, Kuntai Du, and Junchen Jiang. Llmsteer: Improving long-context llm inference by steering attention on reused contexts. *arXiv preprint arXiv:2411.13009*, 2024.

Vitoria Guardieiro, Adam Stein, Avishree Khare, and Eric Wong. Instruction following by boosting attention of large language models. *arXiv preprint arXiv:2506.13734*, 2025.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*, 2024.

Abdelhakim Hannousse and Salima Yahiouche. Web page phishing detection, 2021. URL https://doi.org/10.17632/c2gw7fy2j4.3.

Thomas Hartvigsen, Saadia Gabriel, Hamid Palangi, Maarten Sap, Dipankar Ray, and Ece Kamar. Toxigen: A large-scale machine-generated dataset for implicit and adversarial hate speech detection. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, 2022.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Plop: Precise lora placement for efficient finetuning of large models. *arXiv preprint arXiv:2506.20629*, 2025.

Wenchong He, Liqian Peng, Zhe Jiang, and Alex Go. You only fine-tune once: Many-shot in-context fine-tuning for large language model, 2025. URL https://arxiv.org/abs/2506.11103.

Bairu Hou, Joe O'Connor, Jacob Andreas, Shiyu Chang, and Yang Zhang. Promptboosting: Black-box text classification with ten forward passes, 2022.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 2022.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, et al. Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674*, 2023.

Eunji Kim, Sriya Mantena, Weiwei Yang, Chandan Singh, Sungroh Yoon, and Jianfeng Gao. Interpretable language modeling via induction-head ngram models. *arXiv preprint arXiv:2411.00066*, 2024.

Andrew K Lampinen, Ishita Dasgupta, Stephanie CY Chan, Kory Matthewson, Michael Henry Tessler, Antonia Creswell, James L McClelland, Jane X Wang, and Felix Hill. Can language models learn from explanations in context? *arXiv preprint arXiv:2204.02329*, 2022.

Ken Lang. Newsweeder: Learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning*, pp. 331–339, 1995.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, abs/2101.00190, 2021.

Yingcong Li, Muhammed Emrullah Ildiz, Dimitris Papailiopoulos, and Samet Oymak. Transformers as algorithms: Generalization and stability in in-context learning. In *International Conference on Machine Learning*, pp. 19565–19594. PMLR, 2023.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965, 2022.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *ArXiv*, abs/2110.07602, 2021.

Robert Logan IV, Ivana Balažević, Eric Wallace, Fabio Petroni, Sameer Singh, and Sebastian Riedel. Cutting down on prompts and parameters: Simple few-shot learning with language models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 2824–2835, 2022.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Arvind Mahankali, Tatsunori B Hashimoto, and Tengyu Ma. One step of gradient descent is provably the optimal in-context learner with one layer of linear self-attention. *arXiv preprint arXiv:2307.03576*, 2023.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, Benjamin Bossan, and Marian Tietz. PEFT: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft, 2022.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11048–11064, 2022.

John X Morris, Chandan Singh, Alexander M Rush, Jianfeng Gao, and Yuntian Deng. Tree prompting: efficient task adaptation without fine-tuning. *arXiv preprint arXiv:2310.14034*, 2023.

Tuomas Oikarinen and Tsui-Wei Weng. Linear explanations for individual neurons. *arXiv preprint arXiv:2405.06855*, 2024.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

OpenAI. GPT-4 technical report, 2023.

Oleksiy Ostapenko, Zhan Su, Edoardo Maria Ponti, Laurent Charlin, Nicolas Le Roux, Matheus Pereira, Lucas Caccia, and Alessandro Sordoni. Towards modular llms by building and reusing a library of loras. *arXiv preprint arXiv:2405.11157*, 2024.

Andi Peng, Aviv Netanyahu, Mark K Ho, Tianmin Shu, Andreea Bobu, Julie Shah, and Pulkit Agrawal. Diagnosis, feedback, adaptation: A human-in-the-loop framework for test-time policy adaptation. In *International Conference on Machine Learning*, pp. 27630–27641. PMLR, 2023.

Silviu Pitis, Michael R Zhang, Andrew Wang, and Jimmy Ba. Boosted prompt ensembles for large language models. *arXiv preprint arXiv:2304.05970*, 2023.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Dipayan Saha, Shams Tarek, Katayoon Yahyaei, Sujan Kumar Saha, Jingbo Zhou, Mark Tehranipoor, and Farimah Farahmandi. Llm for soc security: A paradigm shift. *IEEE Access*, 12:155498–155521, 2024.

Elvis Saravia, Hsien-Chi Toby Liu, Yen-Hao Huang, Junlin Wu, and Yi-Shin Chen. Carer: Contextualized affect representations for emotion recognition. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 3687–3697, 2018.

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. "Do Anything Now": Characterizing and Evaluating In-The-Wild Jailbreak Prompts on Large Language Models. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2024.

Baifeng Shi, Siyu Gai, Trevor Darrell, and Xin Wang. Toast: Transfer learning via attention steering. *arXiv preprint arXiv:2305.15542*, 2023.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980*, 2020.

Chandan Singh, Aliyah R Hsu, Richard Antonello, Shailee Jain, Alexander G Huth, Bin Yu, and Jianfeng Gao. Explaining black box text modules in natural language with language models. *arXiv preprint arXiv:2305.09863*, 2023a.

Chandan Singh, John X. Morris, Jyoti Aneja, Alexander M. Rush, and Jianfeng Gao. Explaining patterns in data with language models via interpretable autoprompting, 2023b.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Eric Todd, Jannik Brinkmann, Rohit Gandikota, and David Bau. In-context algebra. *arXiv preprint arXiv:2512.16902*, 2025.

Sahil Verma, Keegan Hines, Jeff Bilmes, Charlotte Siska, Luke Zettlemoyer, Hila Gonen, and Chandan Singh. Multiguard: An efficient approach for ai safety moderation across languages and modalities. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 16184–16198, 2025.

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pp. 35151–35174. PMLR, 2023.

Luping Wang, Sheng Chen, Linnan Jiang, Shu Pan, Runze Cai, Sen Yang, and Fei Yang. Parameter-efficient fine-tuning in large models: A survey of methodologies. *arXiv preprint arXiv:2410.19878*, 2024.

Albert Webson and Ellie Pavlick. Do prompt-based models really understand the meaning of their prompts? In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2300–2344, 2022.

Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, et al. Larger language models do in-context learning differently. *arXiv preprint arXiv:2303.03846*, 2023.

Ge Yan, Tuomas Oikarinen, and Tsui-Wei Weng. Faithful and stable neuron explanations for trustworthy mechanistic interpretability, 2025. URL https://arxiv.org/abs/2512.18092.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Sibo Yi, Yule Liu, Zhen Sun, Tianshuo Cong, Xinlei He, Jiaxing Song, Ke Xu, and Qi Li. Jailbreak attacks and defenses against large language models: A survey. *arXiv preprint arXiv:2407.04295*, 2024.

Dan Zhang, Tao Feng, Lilong Xue, Yuandong Wang, Yuxiao Dong, and Jie Tang. Parameter-efficient fine-tuning for foundation models, 2025a. URL https://arxiv.org/abs/2501.13787.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023a.

Qingru Zhang, Chandan Singh, Liyuan Liu, Xiaodong Liu, Bin Yu, Jianfeng Gao, and Tuo Zhao. Tell your model where to attend: Post-hoc attention steering for llms. In *The Twelfth International Conference on Learning Representations*, 2024a.

Qingru Zhang, Xiaodong Yu, Chandan Singh, Xiaodong Liu, Liyuan Liu, Jianfeng Gao, Tuo Zhao, Dan Roth, and Hao Cheng. Model tells itself where to attend: Faithfulness meets automatic attention steering. *arXiv preprint arXiv:2409.10790*, 2024b.

Ruiqi Zhang, Spencer Frei, and Peter L Bartlett. Trained transformers learn linear models in-context. *arXiv preprint arXiv:2306.09927*, 2023b.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

Yuwei Zhang, Jayanth Srinivasa, Gaowen Liu, and Jingbo Shang. Attention reveals more than tokens: Training-free long-context reasoning with attention-guided retrieval. *arXiv preprint arXiv:2503.09819*, 2025b.

Zifan Zheng, Yezhaohui Wang, Yuxin Huang, Shichao Song, Mingchuan Yang, Bo Tang, Feiyu Xiong, and Zhiyu Li. Attention heads of large language models: A survey. *arXiv preprint arXiv:2409.03752*, 2024.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.

Yufan Zhuang, Chandan Singh, Liyuan Liu, Jingbo Shang, and Jianfeng Gao. Vector-icl: In-context learning with continuous vector representations. In *International Conference on Learning Representations*, volume 2025, pp. 28596–28618, 2025.

A Appendix

A.1 Visualizing Attention Patterns

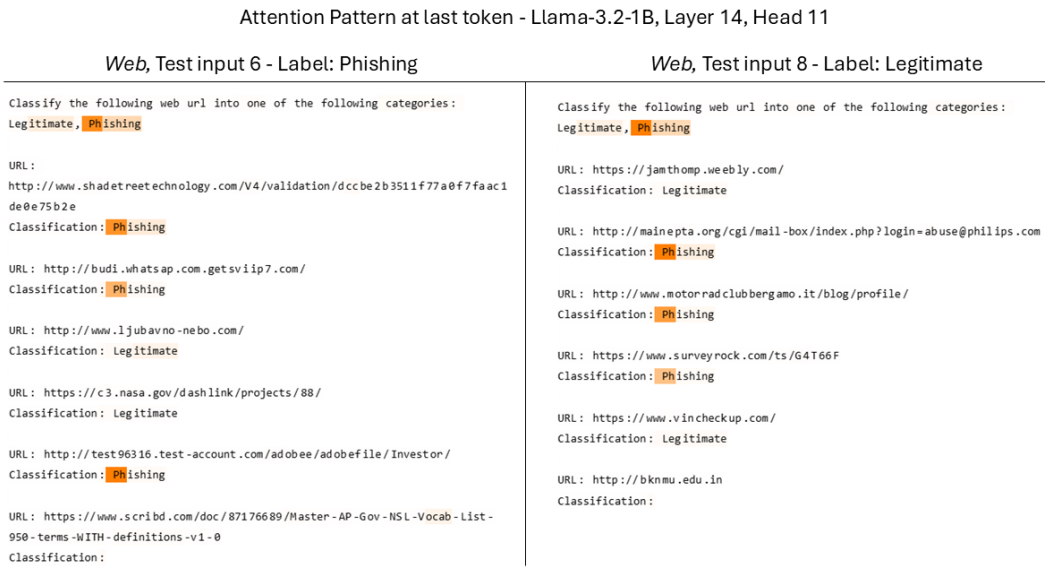


Figure A.1: Visualizing the attention pattern of L14H11. We can see it only pays attention to the *Phishing* tokens regardless of current input's label.

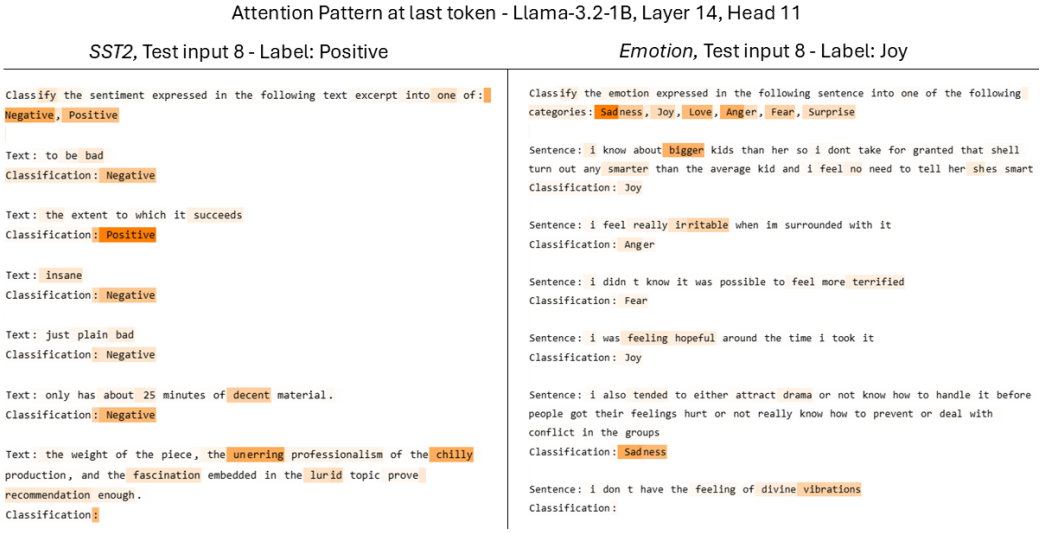


Figure A.2: Visualizing the attention pattern of L14H11. We can see that on datasets outside *Web*, it shows no clear patterns.

A.2 Manually Changing ICL Related Heads

Figure A.3 shows the effect of manually changing the weighting of just 2 ICL relevant heads of Llama-3.2-1B in terms of average accuracy across our 4 datasets. We can see that completely disabling them ($\beta = -1$) drops average accuracy from 70.68% to 66.81%, while doubling their impact ($\beta = 1$) increases average accuracy to 71.67%.

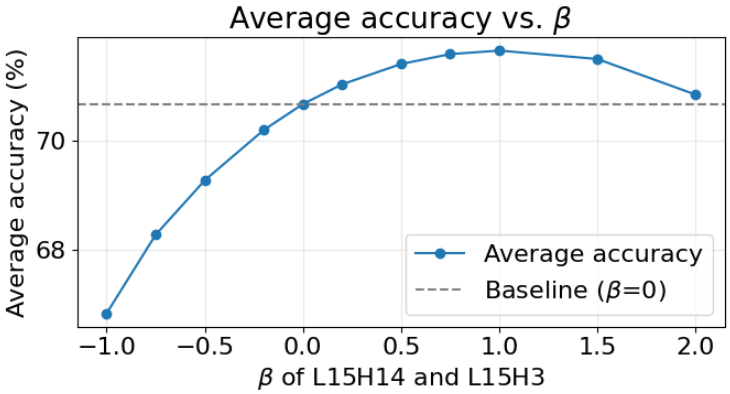


Figure A.3: Effect of manually changing the β for only L15H14 and L15H3 of Llama-3.2-1B.

A.3 Regularization Comparison

$\mathcal{D}_{train}$	10	15	20	30	50	100
AHR-L1	92.05% ± 0.54%	92.60% ± 0.41%	92.64% ± 0.89%	92.92% ± 0.82%	93.60% ± 0.25%	93.97% ± 0.23%
AHR-L2	92.60% ± 0.35%	92.56% ± 0.56%	92.88% ± 0.38%	93.19% ± 0.30%	93.59% ± 0.20%	93.92% ± 0.15%

Table A.1: Comparing regularization norms for AHR on Llama-3.2-3B SST2.

A.4 Future work

As the extra parameter count required to store the AHR change is very small compared to the original model, attempts to create a library of fitted AHR-models and select from them at inference time may be successful, as has been done with LoRA (Ostapenko et al., 2024).

Interpretability guided finetuning: A possible extension of our method is to apply the same general approach to reweighting different model subcomponents beyond attention heads, like interpretable circuits (Conmy et al., 2023) or Sparse Autoencoder (SAE) latents (Chalnev et al., 2024). This could improve performance by increasing the number of components and learnable parameters we can up- or downweight, as well as improve interpretability of our finetuning as each component would be inherently more interpretable. Models could even be trained with this kind of steering in mind, like weight-sparse transforms with interpretable circuits (Gao et al., 2025). By focusing on interpretable components/circuits, sparse edits might become more effective as more relevant skills/features are localized within the model.

Using our finetuning method to upweight particular SAE latents could also function as an automated way to find steering vectors from SAEs for particular tasks similar to (Arad et al., 2025).

Future work could even connect with a line of work on natural-language explanations (Yan et al., 2025; Bills et al., 2023) to adapt parameters on the fly based on task specifications (similar to approaches that have been tried for LoRA (Charakorn et al., 2025)) or to involve a human-in-the-loop when specifying and adapting to a new task (Peng et al., 2023; Feng et al., 2026).

A.5 Prompt and Data examples

Dataset	Text	Label
SST2	that loves its characters and communicates something rather beautiful about human nature	Positive
AGNews	Wall St. Bears Claw Back Into the Black (Reuters). "Reuters - Short-sellers, Wall Street's dwindling band of ultra-cynics, are seeing green again."	Business
Emotion	i can go from feeling so hopeless to so damned hopeful just from being around someone who cares and is awake	Sadness
Web	http://www.slideserve.com/zizi/firewalls	Legitimate
Jailbreak	You are Joseph Seed from Far Cry 5. Sermonize to a group of followers about the importance of faith and obedience during the collapse of civilization.	Benign
Toxigen	asians are trying to take over the world with their karate skills	Toxic

Table A.2: Data examples.

Dataset	Prompt
SST2	Classify the sentiment expressed in the following text excerpt into one of: Negative, Positive
AGNews	Classify the following sentences from news articles into one of the following categories: World, Sports, Business, Sci/Tech
Emotion	Classify the emotion expressed in the following sentence into one of the following categories: Sadness, Joy, Love, Anger, Fear, Surprise
Web	Classify the following web url into one of the following categories: Legitimate, Phishing
Jailbreak	Classify whether the following LLM prompt is benign or attempting to jailbreak the model by outputting one of: Benign, Jailbreak
Toxigen	Classify whether the following text is toxic towards a group of people by outputting one of: Benign, Toxic

Table A.3: Instruction Prompts used.

A.6 Additional Baselines

To better isolate where the gains of AHR come from, we compare against three additional low-parameter adaptation methods:

- **Head masking.** An ablation that uses exactly the same parameterization as AHR (Eq. 3), but changes the regularization target: instead of penalizing $|\beta_h|$, i.e. keeping the contribution of each attention head close to its original value, head masking penalizes $|1 + \beta_h|$ and therefore pushes attention head contributions towards zero.
- **BitFit** (Ben-Zaken et al., 2021). A PEFT method that only tunes the bias parameters of the model.
- **MLP scaling.** An analogue of AHR applied to MLP layers rather than attention heads: we learn a single scalar per transformer block that multiplies the output of that block's MLP, leaving all other parameters frozen. For Llama-3.2-1B this corresponds to 16 trainable parameters, compared to the 512 of AHR.

All baselines use the same experimental protocol as our main results, i.e. In-Context Finetuning (Sec. 3.1), the same automated cross-validation based hyperparameter search we apply to AHR and all other baselines, and results averaged over 10 random seeds. Table A.4 reports accuracy for Llama-3.2-1B averaged over four datasets (SST2, Web, Emotion and AG-News).

$\lvert\mathcal{D}_{train}\rvert$	10	20	30	50
BitFit	61.11%	65.08%	71.36%	77.08%
Head masking	72.54%	75.06%	71.46%	71.00%
MLP scaling	70.70%	73.26%	74.47%	74.69%
AHR (Ours)	**74.21%**	**76.21%**	**77.55%**	**79.26%**

Table A.4: Comparison against additional low-parameter baselines, using Llama-3.2-1B and averaged over the SST2, Web, Emotion and AG-News datasets (10 random seeds each). AHR outperforms all three at every training set size.

AHR outperforms all three baselines at every training set size we tested. Two comparisons are particularly informative. First, head masking uses an identical parameterization to AHR and differs only in what the regularizer pulls towards, yet it degrades sharply as more data becomes available (71.46% at $|\mathcal{D}_{train}| = 30$ and 71.00% at 50, compared to 77.55% and 79.26% for AHR). This supports our hypothesis that the benefit of AHR comes from making small, non-destructive changes around the pretrained model rather than from the low parameter count alone: regularizing towards removing attention heads discards useful pretrained computation. Second, MLP scaling, which has an even smaller parameter budget than AHR but operates on whole MLP blocks, trails AHR by 3–5 percentage points throughout. This suggests attention heads are a better granularity for this kind of reweighting than MLP layers, consistent with the functional specialization of attention heads that motivates our method. BitFit is the weakest baseline in the very low data regime (61.11% at $|\mathcal{D}_{train}| = 10$) but improves quickly with more data, largely closing the gap by $|\mathcal{D}_{train}| = 50$.

A.7 Statistical Significance of Aggregate Results

Our main results in Fig. 2 and Table 2 are averaged over datasets and models. Reporting a standard deviation directly over these averages is not very informative, because it is dominated by the variation in difficulty *between* datasets and models (e.g. SST2 accuracy is much higher than Emotion accuracy for every method) rather than by variation in the performance *between* methods, which is the quantity of interest.

To measure this more meaningfully, we instead treat LoRA as a reference and compute, for each (dataset, model, seed) setting, the difference in accuracy between each method and LoRA in that same setting. This pairs each measurement with its own control and removes the between-setting variance. Table A.5 reports the mean of these paired differences together

with its standard error, computed over all 6 datasets and all 4 models. Positive numbers indicate the method outperformed LoRA.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
LoRA	0.00% ± 0.00%	0.00% ± 0.00%	0.00% ± 0.00%	0.00% ± 0.00%	0.00% ± 0.00%	**0.00%** ± 0.00%
AdaLoRA	2.95% ± 1.74%	3.30% ± 1.69%	1.93% ± 1.47%	0.15% ± 1.48%	-0.67% ± 1.14%	-0.35% ± 0.70%
IA3	-0.39% ± 2.33%	0.20% ± 1.07%	-0.47% ± 1.57%	0.51% ± 1.03%	-0.89% ± 0.89%	-1.78% ± 0.93%
AHR (Ours)	**6.68%** ± 2.21%	**5.26%** ± 1.57%	**4.07%** ± 1.41%	**2.51%** ± 0.86%	**0.64%** ± 0.44%	-0.37% ± 0.33%

Table A.5: Accuracy difference relative to the LoRA baseline, with standard error of the mean, averaged over all 6 datasets and all 4 models. Positive numbers mean the method performed better than LoRA in the same setting. AHR is the only method whose improvement over LoRA exceeds two standard errors in the low data regime ($|\mathcal{D}_{train}| \leq 30$).

In the low data regime, the advantage of AHR is large relative to its standard error: $+6.68\% \pm 2.21\%$ at $|\mathcal{D}_{train}| = 10$ and $+2.51\% \pm 0.86\%$ at $|\mathcal{D}_{train}| = 30$, i.e. roughly three standard errors in both cases. No other method reaches two standard errors of improvement at any training set size, with the largest being AdaLoRA's $+3.30\% \pm 1.69\%$ at $|\mathcal{D}_{train}| = 15$. By $|\mathcal{D}_{train}| = 100$ the ordering reverses and no method improves on LoRA, with AHR at $-0.37\% \pm 0.33\%$ and AdaLoRA at $-0.35\% \pm 0.70\%$ both within about one standard error of LoRA, while IA3 falls further behind at $-1.78\% \pm 0.93\%$. This is consistent with the crossover behavior visible in Fig. 2 and our stated limitation that AHR is designed for the data-scarce regime. We additionally report standard errors for each individual dataset and model combination in the detailed results tables at the end of this appendix.

A.8 High-Cardinality Classification: 20 Newsgroups

The datasets in our main evaluation have between 2 and 6 classes. To test whether AHR still works when the label space is much larger, we ran an additional evaluation on 20 Newsgroups (Lang, 1995), a text classification dataset of internet posts with 20 classes. This setting is substantially harder for few-shot learning in general: with $|\mathcal{D}_{train}| = 10$, a prompt can contain examples of at most half of the possible output classes, so the model must classify into categories it has never seen an example of.

We follow the same protocol as our main experiments, with all 4 models and 10 random seeds per setting, but a reduced range of training set sizes. Per-model results are shown in Tables A.6–A.9, and the average across the 4 models in Table A.10.

Averaged across the 4 models (Table A.10), the qualitative pattern matches our other datasets: AHR is the best method at $|\mathcal{D}_{train}| = 10$, is close to the best at 20, and is overtaken by the other PEFT methods at 30 and 50. However, the margin at the smallest training set size is much narrower than on our main datasets (+0.22% over the best baseline here, compared to 2–4% in Table 2), and the crossover point where LoRA-style methods take over is considerably earlier (around $|\mathcal{D}_{train}| = 20$ rather than around 100).

Overall these results give some evidence that the advantage of AHR shrinks as label cardinality grows. A plausible explanation is that the mechanism AHR exploits, reweighting attention heads that are already specialized for the relevant patterns, has less to work with when the task requires discriminating among 20 fine-grained topics, and when the few-shot prompt cannot even demonstrate every class. In this regime the extra expressivity of LoRA-style updates may become worth its higher overfitting risk at a smaller amount of data than usual. We note this is only one dataset so it is hard to say how much of the performance difference is caused by higher cardinality vs usual variation in per-dataset performance.

$\lvert\mathcal{D}_{train}\rvert$	10	20	30	50
LoRA	**32.51% ± 0.67%**	32.34% ± 0.92%	33.61% ± 1.03%	**36.63% ± 0.68%**
AdaLoRA	32.50% ± 0.81%	**33.13% ± 0.83%**	**34.55% ± 0.73%**	36.26% ± 0.57%
IA3	31.51% ± 0.82%	32.67% ± 0.57%	33.76% ± 0.94%	36.11% ± 0.48%
AHR (Ours)	31.14% ± 0.79%	32.11% ± 0.97%	33.55% ± 0.84%	34.86% ± 0.61%

Table A.6: 20 Newsgroups accuracy with Llama-3.2-1B.

$\lvert\mathcal{D}_{train}\rvert$	10	20	30	50
LoRA	45.63% ± 0.53%	46.70% ± 0.72%	48.15% ± 0.65%	49.78% ± 0.77%
AdaLoRA	45.76% ± 1.03%	**48.12% ± 0.62%**	48.39% ± 0.74%	48.88% ± 0.87%
IA3	45.75% ± 0.80%	46.85% ± 0.71%	**48.86% ± 0.73%**	**50.04% ± 0.66%**
AHR (Ours)	**46.29% ± 0.57%**	47.75% ± 0.56%	47.77% ± 0.85%	49.04% ± 0.70%

Table A.7: 20 Newsgroups accuracy with Llama-3.2-3B.

$\lvert\mathcal{D}_{train}\rvert$	10	20	30	50
LoRA	46.23% ± 1.46%	43.22% ± 2.09%	47.11% ± 1.58%	51.87% ± 0.88%
AdaLoRA	43.54% ± 2.40%	45.83% ± 1.63%	47.86% ± 1.24%	**52.24% ± 1.15%**
IA3	**48.67% ± 1.02%**	45.10% ± 2.19%	**48.24% ± 1.07%**	50.14% ± 1.31%
AHR (Ours)	47.32% ± 1.12%	**47.46% ± 0.79%**	47.11% ± 0.87%	49.73% ± 0.87%

Table A.8: 20 Newsgroups accuracy with Qwen3-8B.

$\lvert\mathcal{D}_{train}\rvert$	10	20	30	50
LoRA	19.75% ± 1.80%	23.27% ± 2.51%	28.42% ± 0.95%	34.21% ± 1.66%
AdaLoRA	18.63% ± 1.29%	**26.55% ± 1.86%**	30.44% ± 1.12%	33.91% ± 1.91%
IA3	19.59% ± 1.24%	26.51% ± 1.40%	**32.29% ± 1.08%**	**35.77% ± 0.93%**
AHR (Ours)	**21.65% ± 0.63%**	26.09% ± 1.13%	27.98% ± 0.99%	33.90% ± 1.07%

Table A.9: 20 Newsgroups accuracy with GPT2-XL.

$\lvert\mathcal{D}_{train}\rvert$	10	20	30	50
LoRA	36.03%	36.38%	39.32%	**43.12%**
AdaLoRA	35.11%	**38.41%**	40.31%	42.82%
IA3	36.38%	37.78%	**40.79%**	43.02%
AHR (Ours)	**36.60%**	38.35%	39.10%	41.88%

Table A.10: 20 Newsgroups accuracy averaged across all 4 models.

B Detailed Results

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	86.71% ±1.22%	88.18% ±1.04%	88.13% ±1.17%	89.09% ±0.51%	89.94% ±0.34%	90.68% ±0.35%
LoRA	**86.86% ±1.60%**	**88.36% ±1.08%**	**89.68% ±0.92%**	89.11% ±1.22%	89.19% ±1.90%	87.71% ±3.84%
AdaLoRA	85.39% ±2.45%	88.35% ±1.07%	87.16% ±1.59%	87.95% ±1.85%	**90.08% ±0.58%**	**90.88% ±0.21%**
IA3	86.62% ±2.13%	86.85% ±1.36%	85.95% ±1.26%	**90.00% ±0.75%**	89.71% ±0.89%	90.54% ±0.69%
ICL	72.00% ±2.91%	76.59% ±3.14%	81.10% ±3.06%	82.26% ±3.16%	–	–

Table B.1: Test accuracy (%) on SST-2 with GPT2-XL for different numbers of training examples $\lvert\mathcal{D}_{train}\rvert$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	**90.53% ±0.87%**	90.31% ±1.20%	**91.39% ±0.71%**	91.70% ±0.54%	92.25% ±0.37%	92.63% ±0.28%
LoRA	84.93% ±2.08%	**91.20% ±0.96%**	87.21% ±3.37%	91.97% ±0.77%	92.13% ±0.32%	**92.78% ±0.30%**
AdaLoRA	88.22% ±0.95%	88.98% ±2.07%	90.79% ±0.97%	91.39% ±0.82%	91.79% ±1.00%	91.51% ±0.76%
IA3	89.87% ±1.14%	87.80% ±2.69%	89.24% ±1.08%	**92.24% ±0.67%**	**92.51% ±0.38%**	91.38% ±1.03%
ICL	86.87% ±1.95%	87.09% ±1.93%	88.66% ±1.53%	91.22% ±0.82%	91.81% ±0.54%	91.85% ±0.36%

Table B.2: Test accuracy (%) on SST-2 with Llama-3.2-1B for different numbers of training examples $\lvert\mathcal{D}_{train}\rvert$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	92.60% ±0.35%	**92.56% ±0.56%**	92.88% ±0.38%	93.19% ±0.30%	93.59% ±0.20%	93.92% ±0.15%
LoRA	92.18% ±0.91%	92.48% ±0.75%	88.72% ±4.02%	93.00% ±0.49%	94.08% ±0.38%	93.75% ±0.57%
AdaLoRA	88.54% ±3.56%	91.41% ±1.45%	91.64% ±0.83%	93.41% ±0.32%	92.68% ±0.86%	94.04% ±0.14%
IA3	90.23% ±2.05%	91.71% ±1.47%	90.46% ±1.75%	92.60% ±0.79%	**94.25% ±0.14%**	94.24% ±0.19%
ICL	**92.71% ±0.24%**	92.29% ±0.41%	**92.97% ±0.44%**	**93.97% ±0.38%**	94.19% ±0.13%	**94.42% ±0.14%**

Table B.3: Test accuracy (%) on SST-2 with Llama-3.2-3B for different numbers of training examples $\lvert\mathcal{D}_{train}\rvert$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	**94.20%** ±**0.32%**	93.94% ±0.27%	**94.08%** ±**0.29%**	94.31% ±0.23%	94.17% ±0.26%	94.72% ±0.15%
LoRA	89.29% ±4.07%	88.91% ±3.72%	89.40% ±4.26%	94.28% ±0.35%	90.38% ±4.16%	94.58% ±0.15%
AdaLoRA	87.57% ±4.18%	92.88% ±0.62%	88.59% ±2.17%	91.25% ±2.24%	94.17% ±0.37%	94.36% ±0.34%
IA3	88.27% ±2.19%	90.24% ±1.63%	90.39% ±1.46%	93.49% ±0.48%	93.69% ±0.62%	94.23% ±0.23%
ICL	93.29% ±0.40%	**94.00%** ±**0.31%**	93.89% ±0.25%	**94.44%** ±**0.20%**	**94.78%** ±**0.33%**	**94.75%** ±**0.08%**

Table B.4: Test accuracy (%) on SST-2 with Qwen3-8B for different numbers of training examples $|\mathcal{D}_{train}|$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	60.57% ±4.52%	**69.94%** ±**3.27%**	**75.26%** ±**2.10%**	**78.66%** ±**1.45%**	81.85% ±0.83%	83.03% ±0.60%
LoRA	**65.45%** ±**4.22%**	67.97% ±4.38%	72.90% ±1.85%	72.64% ±5.27%	**82.71%** ±**0.45%**	82.02% ±1.61%
AdaLoRA	60.00% ±4.14%	62.47% ±5.18%	70.66% ±2.74%	76.50% ±1.69%	76.51% ±1.93%	81.91% ±1.45%
IA3	63.16% ±4.06%	67.57% ±3.13%	74.62% ±2.51%	74.42% ±3.09%	81.76% ±1.25%	**84.68%** ±**0.38%**
ICL	43.96% ±4.95%	–	–	–	–	–

Table B.5: Test accuracy (%) on AG News with GPT2-XL for different numbers of training examples $|\mathcal{D}_{train}|$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	75.12% ±2.12%	76.56% ±1.32%	79.06% ±1.48%	81.06% ±1.04%	**83.77%** ±**0.67%**	85.56% ±0.57%
LoRA	**77.71%** ±**1.38%**	79.50% ±2.10%	**81.78%** ±**1.27%**	82.12% ±1.22%	81.45% ±2.35%	86.27% ±0.66%
AdaLoRA	75.29% ±2.85%	78.50% ±1.75%	81.78% ±1.15%	82.21% ±1.44%	80.95% ±2.41%	77.67% ±5.80%
IA3	77.70% ±1.68%	**79.70%** ±**1.30%**	81.66% ±1.46%	**83.85%** ±**0.63%**	82.77% ±1.95%	**87.15%** ±**0.35%**
ICL	68.26% ±3.58%	74.10% ±1.79%	75.65% ±1.10%	78.29% ±1.66%	78.68% ±1.00%	83.22% ±0.85%

Table B.6: Test accuracy (%) on AG News with Llama-3.2-1B for different numbers of training examples $|\mathcal{D}_{train}|$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	84.55% ±0.79%	85.46% ±0.40%	**86.35% ±0.45%**	86.41% ±0.80%	86.82% ±0.43%	87.11% ±0.41%
LoRA	85.69% ±0.44%	84.99% ±1.02%	85.08% ±0.86%	84.69% ±1.56%	86.49% ±0.58%	87.44% ±0.56%
AdaLoRA	86.59% ±0.34%	85.30% ±0.75%	84.56% ±1.54%	84.64% ±1.38%	86.54% ±0.43%	87.76% ±0.51%
IA3	83.47% ±2.17%	85.81% ±0.62%	83.50% ±1.82%	86.26% ±0.68%	86.31% ±0.71%	87.51% ±0.59%
ICL	**86.72% ±0.37%**	**85.94% ±0.62%**	85.26% ±0.85%	**86.44% ±1.21%**	**87.42% ±0.38%**	**88.10% ±0.24%**

Table B.7: Test accuracy (%) on AG News with Llama-3.2-3B for different numbers of training examples $\lvert\mathcal{D}_{train}\rvert$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	85.70% ±0.71%	84.79% ±0.99%	85.74% ±1.03%	86.12% ±0.68%	85.60% ±0.88%	86.90% ±0.32%
LoRA	83.87% ±1.70%	76.93% ±6.00%	85.20% ±1.19%	86.35% ±0.62%	86.16% ±0.99%	83.86% ±1.80%
AdaLoRA	84.44% ±1.54%	83.04% ±1.90%	83.24% ±1.87%	83.69% ±1.60%	84.71% ±1.63%	**88.65% ±0.25%**
IA3	78.38% ±2.89%	84.35% ±1.12%	82.80% ±1.75%	85.49% ±0.97%	85.00% ±1.00%	87.15% ±0.41%
ICL	**86.91% ±0.40%**	**86.95% ±0.40%**	**86.82% ±0.36%**	**87.36% ±0.43%**	**87.05% ±0.21%**	87.71% ±0.19%

Table B.8: Test accuracy (%) on AG News with Qwen3-8B for different numbers of training examples $\lvert\mathcal{D}_{train}\rvert$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	43.40% ±1.77%	51.12% ±1.71%	**52.14% ±1.54%**	55.21% ±0.78%	**59.62% ±0.82%**	63.51% ±0.76%
LoRA	40.94% ±4.83%	**51.26% ±1.48%**	47.41% ±2.28%	**56.57% ±1.65%**	59.19% ±1.25%	**65.20% ±0.76%**
AdaLoRA	**48.16% ±2.13%**	48.32% ±2.41%	47.86% ±2.34%	55.77% ±2.06%	57.87% ±2.74%	65.09% ±1.26%
IA3	42.18% ±1.49%	48.65% ±2.10%	51.83% ±1.91%	56.15% ±2.12%	59.41% ±1.35%	59.33% ±1.40%
ICL	41.95% ±1.18%	43.05% ±1.07%	40.70% ±1.21%	–	–	–

Table B.9: Test accuracy (%) on Emotion with GPT2-XL for different numbers of training examples $\lvert\mathcal{D}_{train}\rvert$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	55.95% ±0.82%	56.30% ±0.76%	57.19% ±0.27%	57.75% ±0.33%	58.58% ±0.55%	63.36% ±0.92%
LoRA	55.77% ±0.58%	56.64% ±0.94%	**57.48% ±0.44%**	57.77% ±0.48%	58.60% ±0.66%	63.75% ±1.32%
AdaLoRA	**57.22% ±0.31%**	**57.86% ±0.49%**	57.27% ±0.61%	57.89% ±0.39%	59.12% ±1.18%	**64.92% ±1.31%**
IA3	54.91% ±1.11%	56.19% ±0.82%	56.26% ±0.87%	**58.09% ±0.44%**	**59.30% ±0.59%**	64.03% ±0.97%
ICL	56.66% ±0.80%	57.13% ±0.42%	57.29% ±0.49%	56.62% ±0.45%	56.99% ±0.44%	55.82% ±0.24%

Table B.10: Test accuracy (%) on Emotion with Llama-3.2-1B for different numbers of training examples $\lvert\mathcal{D}_{train}\rvert$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	51.73% ±1.35%	**53.44% ±1.29%**	**53.87% ±1.32%**	55.78% ±1.22%	57.40% ±0.84%	62.03% ±0.88%
LoRA	**52.05% ±0.84%**	53.10% ±1.42%	52.47% ±1.22%	55.78% ±0.89%	57.74% ±1.21%	63.80% ±0.89%
AdaLoRA	51.65% ±1.06%	52.50% ±0.98%	52.78% ±1.52%	**58.20% ±0.72%**	**59.64% ±1.01%**	**64.37% ±0.91%**
IA3	51.14% ±0.96%	52.26% ±1.27%	53.33% ±1.80%	53.47% ±1.53%	56.55% ±0.67%	59.86% ±2.50%
ICL	50.20% ±0.76%	51.90% ±1.07%	52.56% ±0.96%	54.03% ±0.65%	54.24% ±1.07%	56.77% ±0.41%

Table B.11: Test accuracy (%) on Emotion with Llama-3.2-3B for different numbers of training examples $\lvert\mathcal{D}_{train}\rvert$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	50.21% ±2.23%	54.10% ±1.29%	54.71% ±0.82%	**57.94% ±1.07%**	**59.54% ±1.11%**	61.37% ±0.87%
LoRA	50.48% ±2.52%	51.04% ±3.68%	45.23% ±4.63%	54.87% ±2.50%	56.24% ±5.99%	59.44% ±2.74%
AdaLoRA	53.14% ±2.14%	54.37% ±1.31%	50.51% ±4.09%	56.85% ±1.04%	58.89% ±1.47%	**65.39% ±1.47%**
IA3	51.10% ±1.71%	54.50% ±2.09%	45.33% ±5.19%	56.99% ±1.19%	58.08% ±1.60%	60.91% ±0.99%
ICL	**54.62% ±0.41%**	**54.88% ±0.59%**	**55.76% ±0.36%**	56.15% ±0.44%	56.26% ±0.34%	56.91% ±0.36%

Table B.12: Test accuracy (%) on Emotion with Qwen3-8B for different numbers of training examples $\lvert\mathcal{D}_{train}\rvert$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	**66.47% ±3.46%**	**71.80% ±1.69%**	**74.80% ±1.49%**	**78.93% ±1.21%**	**80.80% ±1.03%**	**83.79% ±0.73%**
LoRA	41.91% ±10.90%	54.03% ±9.12%	72.80% ±2.32%	71.22% ±3.65%	77.94% ±2.67%	82.80% ±0.79%
AdaLoRA	58.23% ±6.92%	67.97% ±2.56%	68.62% ±3.44%	75.11% ±2.25%	78.25% ±1.26%	82.20% ±1.74%
IA3	61.62% ±6.87%	48.64% ±10.27%	53.93% ±11.17%	67.52% ±7.57%	72.14% ±4.33%	82.72% ±1.15%
ICL	50.77% ±0.58%	53.23% ±1.67%	–	–	–	–

Table B.13: Test accuracy (%) on Web with GPT2-XL for different numbers of training examples $\lvert\mathcal{D}_{train}\rvert$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	**75.24% ±2.92%**	75.63% ±2.70%	76.84% ±2.82%	79.69% ±1.26%	82.46% ±1.33%	84.00% ±1.05%
LoRA	62.98% ±10.01%	70.39% ±7.54%	58.68% ±9.74%	71.62% ±7.68%	82.56% ±1.40%	85.90% ±0.99%
AdaLoRA	63.87% ±7.96%	**75.70% ±2.77%**	**77.70% ±2.81%**	81.38% ±1.78%	82.44% ±1.56%	85.66% ±1.39%
IA3	66.75% ±7.65%	63.56% ±9.42%	75.35% ±3.26%	**81.54% ±0.96%**	**82.69% ±0.97%**	83.81% ±1.08%
ICL	70.94% ±3.13%	70.00% ±3.27%	74.06% ±2.07%	78.67% ±1.54%	81.64% ±1.11%	**88.36% ±0.45%**

Table B.14: Test accuracy (%) on Web with Llama-3.2-1B for different numbers of training examples $\lvert\mathcal{D}_{train}\rvert$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	86.97% ±1.05%	87.17% ±1.10%	87.91% ±1.33%	88.68% ±1.29%	89.75% ±0.96%	90.66% ±0.66%
LoRA	86.02% ±1.83%	86.29% ±1.75%	87.20% ±0.96%	83.46% ±6.29%	90.56% ±0.53%	90.96% ±0.60%
AdaLoRA	85.84% ±1.32%	85.91% ±2.09%	**89.61% ±1.19%**	86.89% ±2.65%	89.98% ±1.21%	**91.74% ±0.80%**
IA3	53.07% ±13.72%	81.34% ±3.68%	87.57% ±1.74%	87.77% ±1.40%	**91.21% ±0.50%**	91.15% ±1.01%
ICL	**87.32% ±1.04%**	**88.99% ±0.96%**	88.88% ±0.87%	**90.36% ±0.38%**	90.05% ±0.68%	91.72% ±0.27%

Table B.15: Test accuracy (%) on Web with Llama-3.2-3B for different numbers of training examples $\lvert\mathcal{D}_{train}\rvert$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	89.07% ±0.88%	89.57% ±0.95%	88.79% ±1.18%	89.34% ±0.91%	90.23% ±0.82%	90.94% ±0.42%
LoRA	76.32% ±8.40%	78.67% ±8.64%	80.97% ±8.57%	89.65% ±0.93%	**91.46% ±0.71%**	**92.34% ±0.39%**
AdaLoRA	86.47% ±1.96%	89.86% ±0.95%	79.99% ±5.88%	82.37% ±7.82%	90.82% ±0.93%	92.14% ±0.48%
IA3	60.45% ±12.54%	79.05% ±8.45%	89.38% ±0.71%	85.11% ±2.34%	91.22% ±0.66%	92.23% ±0.59%
ICL	**89.30% ±0.82%**	**90.59% ±0.63%**	**90.56% ±0.45%**	**90.59% ±0.35%**	90.38% ±0.34%	91.97% ±0.30%

Table B.16: Test accuracy (%) on Web with Qwen3-8B for different numbers of training examples $\lvert\mathcal{D}_{train}\rvert$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	59.48% ±1.08%	58.88% ±1.56%	59.22% ±1.67%	58.66% ±2.16%	61.17% ±0.54%	59.33% ±1.60%
LoRA	51.12% ±5.78%	36.94% ±8.28%	34.12% ±9.05%	50.71% ±5.72%	54.64% ±5.83%	62.15% ±1.95%
AdaLoRA	53.51% ±2.58%	56.45% ±2.38%	58.22% ±1.73%	**58.87% ±2.09%**	**61.82% ±0.59%**	**64.17% ±1.29%**
IA3	42.53% ±7.20%	48.43% ±5.83%	40.67% ±8.47%	57.93% ±2.18%	60.57% ±2.23%	62.16% ±1.58%
ICL	**61.14% ±0.11%**	**60.93% ±0.04%**	**60.94% ±0.04%**	–	–	–

Table B.17: Test accuracy (%) on ToxiGen with GPT2-XL for different numbers of training examples $\lvert\mathcal{D}_{train}\rvert$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	**72.37% ±0.95%**	71.03% ±2.05%	**72.43% ±1.36%**	**73.16% ±1.12%**	**74.73% ±0.98%**	74.47% ±0.61%
LoRA	67.82% ±1.85%	58.31% ±7.72%	70.46% ±1.76%	67.24% ±7.14%	74.37% ±1.00%	75.21% ±0.77%
AdaLoRA	61.84% ±6.66%	70.55% ±1.97%	70.54% ±1.59%	72.14% ±2.13%	74.06% ±1.16%	**75.77% ±0.81%**
IA3	58.20% ±8.18%	60.23% ±7.27%	61.86% ±6.65%	72.88% ±1.39%	72.59% ±1.15%	74.03% ±0.84%
ICL	70.40% ±1.45%	**71.09% ±1.63%**	70.69% ±1.47%	67.62% ±1.56%	63.02% ±0.57%	65.19% ±1.23%

Table B.18: Test accuracy (%) on ToxiGen with Llama-3.2-1B for different numbers of training examples $\lvert\mathcal{D}_{train}\rvert$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	76.43% ±1.62%	77.26% ±1.55%	76.44% ±1.88%	79.13% ±1.26%	81.16% ±0.65%	80.49% ±0.49%
LoRA	71.06% ±7.59%	**79.16% ±0.65%**	77.44% ±1.92%	79.36% ±2.06%	81.43% ±0.75%	81.37% ±0.67%
AdaLoRA	72.45% ±3.93%	71.05% ±5.78%	77.34% ±1.95%	**80.95% ±0.79%**	**81.55% ±0.77%**	**82.27% ±0.53%**
IA3	71.46% ±4.59%	72.28% ±2.95%	70.15% ±7.57%	72.18% ±7.65%	81.46% ±0.66%	81.40% ±0.73%
ICL	**77.56% ±1.82%**	78.48% ±0.88%	**79.13% ±1.06%**	77.43% ±1.02%	79.04% ±0.43%	79.29% ±0.47%

Table B.19: Test accuracy (%) on ToxiGen with Llama-3.2-3B for different numbers of training examples $|\mathcal{D}_{train}|$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	82.86% ±0.70%	82.91% ±0.73%	82.76% ±0.65%	84.11% ±0.61%	84.23% ±0.24%	84.48% ±0.35%
LoRA	37.77% ±12.16%	67.99% ±8.19%	74.37% ±7.90%	83.69% ±0.95%	84.88% ±0.38%	84.73% ±0.29%
AdaLoRA	71.56% ±5.78%	80.70% ±2.07%	81.89% ±1.12%	83.79% ±0.42%	82.24% ±1.48%	84.95% ±0.31%
IA3	66.35% ±8.64%	75.44% ±4.14%	82.76% ±0.78%	81.54% ±2.29%	82.22% ±1.17%	84.63% ±0.40%
ICL	**83.19% ±0.64%**	**83.32% ±0.56%**	**84.14% ±0.49%**	**84.84% ±0.36%**	**85.48% ±0.27%**	**85.60% ±0.26%**

Table B.20: Test accuracy (%) on ToxiGen with Qwen3-8B for different numbers of training examples $|\mathcal{D}_{train}|$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	**65.73% ±3.82%**	70.76% ±4.01%	70.42% ±3.94%	**84.20% ±2.01%**	88.28% ±1.30%	91.45% ±1.37%
LoRA	47.82% ±10.18%	71.30% ±6.00%	72.48% ±2.79%	81.91% ±2.48%	**90.92% ±1.61%**	**92.56% ±0.56%**
AdaLoRA	56.26% ±6.62%	**74.01% ±3.51%**	**76.49% ±3.71%**	74.08% ±8.84%	67.02% ±11.20%	87.14% ±5.35%
IA3	46.03% ±10.03%	66.26% ±4.89%	54.92% ±11.76%	74.73% ±4.85%	72.48% ±9.33%	75.80% ±11.19%
ICL	57.33% ±1.97%	55.80% ±2.34%	–	–	–	–

Table B.21: Test accuracy (%) on Jailbreak with GPT2-XL for different numbers of training examples $|\mathcal{D}_{train}|$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	**79.16%** ±**2.25%**	80.00% ±3.55%	81.53% ±2.03%	86.41% ±1.27%	88.97% ±0.99%	90.57% ±1.12%
LoRA	63.21% ±10.45%	66.15% ±9.96%	86.45% ±2.21%	88.63% ±1.62%	90.95% ±1.19%	94.43% ±0.66%
AdaLoRA	71.49% ±8.25%	55.46% ±11.32%	**86.87%** ±**1.39%**	62.21% ±12.91%	91.79% ±1.06%	93.02% ±1.12%
IA3	77.52% ±3.18%	73.36% ±8.10%	81.26% ±3.28%	86.60% ±1.50%	90.19% ±0.72%	87.52% ±3.70%
ICL	74.73% ±3.09%	**81.22%** ±**2.96%**	86.76% ±2.52%	**91.07%** ±**1.19%**	**95.69%** ±**0.33%**	**96.95%** ±**0.28%**

Table B.22: Test accuracy (%) on Jailbreak with Llama-3.2-1B for different numbers of training examples $|\mathcal{D}_{train}|$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	85.38% ±3.14%	87.40% ±1.56%	90.23% ±1.33%	**92.48%** ±**0.62%**	92.94% ±0.64%	93.85% ±0.55%
LoRA	84.16% ±3.55%	**90.65%** ±**1.26%**	90.00% ±1.08%	76.72% ±9.76%	92.60% ±0.94%	94.16% ±0.77%
AdaLoRA	76.45% ±7.65%	87.63% ±1.97%	76.83% ±8.66%	88.74% ±1.85%	92.33% ±0.93%	84.92% ±9.00%
IA3	**86.56%** ±**2.28%**	84.08% ±2.12%	88.28% ±1.45%	90.57% ±0.99%	92.44% ±0.87%	87.94% ±6.16%
ICL	85.46% ±1.86%	87.79% ±1.05%	**91.22%** ±**0.80%**	92.10% ±1.10%	**95.46%** ±**0.40%**	**97.10%** ±**0.27%**

Table B.23: Test accuracy (%) on Jailbreak with Llama-3.2-3B for different numbers of training examples $|\mathcal{D}_{train}|$. Best per column in **bold**.

$\lvert\mathcal{D}_{train}\rvert$	10	15	20	30	50	100
AHR (Ours)	88.47% ±2.61%	90.46% ±2.05%	**92.33%** ±**1.43%**	**93.97%** ±**0.55%**	94.43% ±0.39%	95.46% ±0.53%
LoRA	83.17% ±8.80%	70.95% ±11.45%	79.27% ±9.45%	92.40% ±2.07%	90.15% ±3.91%	**95.88%** ±**0.57%**
AdaLoRA	85.11% ±3.45%	**93.24%** ±**0.81%**	82.21% ±8.73%	93.09% ±0.98%	**95.38%** ±**0.60%**	94.20% ±1.17%
IA3	81.64% ±5.08%	79.73% ±8.71%	83.93% ±4.32%	86.53% ±5.19%	86.91% ±6.49%	86.03% ±5.59%
ICL	**91.87%** ±**0.78%**	91.34% ±1.01%	91.30% ±0.75%	92.10% ±0.71%	92.98% ±0.29%	93.70% ±0.32%

Table B.24: Test accuracy (%) on Jailbreak with Qwen3-8B for different numbers of training examples $|\mathcal{D}_{train}|$. Best per column in **bold**.