Deeper Insights Without Updates: The Power of In-Context Learning Over Fine-Tuning

Anonymous ACL submission

Abstract

Fine-tuning and in-context learning (ICL) are two prevalent methods in imbuing large language models with task-specific knowledge. It is commonly believed that fine-tuning can surpass ICL given sufficient training samples as it allows the model to adjust its internal parameters based on the data. However, this paper presents a counterintuitive finding: For tasks with implicit patterns, ICL captures these patterns significantly better than fine-tuning. We developed several datasets featuring implicit patterns, such as sequences determining answers through parity or identifying reducible terms in calculations. We then evaluated the models' understanding of these patterns under both fine-tuning and ICL across models ranging from 0.5B to 7B parameters. The results indicate that models employing ICL can quickly grasp deep patterns and significantly improve accuracy. In contrast, fine-tuning, despite utilizing thousands of times more training samples than ICL, achieved only limited improvements. We also proposed circuit shift theory from a mechanistic interpretability's view to explain why ICL wins.

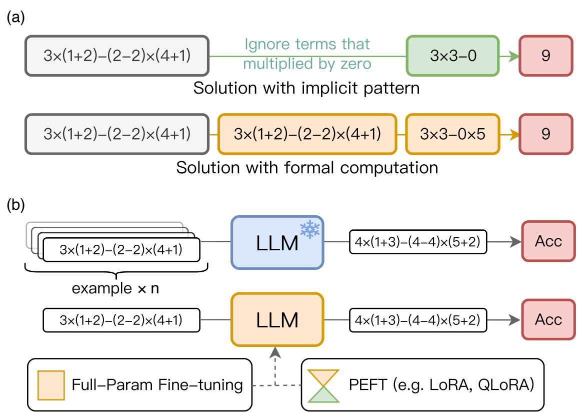

Figure 1: (a) A simple example of an implicit pattern detection task. The given problem (arithmetic expression calculation task in this figure) can be solved in either a formal way, e.g., directly calculating, or by exploiting the detected implicit pattern as a shortcut. (b) Illustration of implicit pattern detection for in-context learning and fine-tuning. For ICL, several examples with answers are given in context, and a further new question is used to test accuracy. For fine-tuning, LLM learns from single examples using parameter update methods like full-parameter fine-tuning or PEFT methods.

1 Introduction

Adapting pre-trained models to specific tasks or domains is commonly achieved through fine-tuning (Hu et al., 2023; Peters et al., 2019) or in-context learning (Gan and Mori, 2023). Fine-tuning, a well-established method, involves further training a pre-trained model on a smaller, domain-specific dataset, directly updating the model's parameters to retain improvements across various contexts and scenarios. In contrast, in-context learning (ICL) enhances task performance by incorporating task-specific examples into prompts, guiding the model in task completion without altering its parameters during training.

There has been much debate about the pros and cons of fine-tuning and in-context learning. Fine-tuning is praised for its ability to bring permanent memorization to models (Hu et al., 2023), and it can perform well even with a small amount of training data (Liu et al., 2022). However, critics argue that fine-tuning demands substantial computational resources (Hu et al., 2021) and can encounter issues such as catastrophic forgetting (Zhai et al., 2023). This conserves computational resources but necessitates longer prompts and incurs higher inference costs.

How about ICL? It is favored for its training-free nature (Dong et al., 2022), allowing prompts to be easily changed for adaptation to other domains without re-training (Min et al., 2022). Other works(Bhattamishra et al., 2023) showed that ICL can help the model uniquely identify a discrete function sample-efficiently. (Reddy, 2023) showed that ICL is driven by the abrupt emergence of an induction head, which subsequently competes with in-weights learning. (Shen et al., 2024) observes that ICL and gradient descent modify the output distribution of language models differently. Despite these advantages, ICL is limited by context length restrictions and incurs higher costs during each inference stage due to the longer prompts required.

Essentially, the primary distinction between fine-tuning and ICL lies in parameter updating; all fine-tuning methods modify the model's parameters. It might seem, therefore, that ICL's impact is less profound. However, our research reveals a counterintuitive finding: **for datasets with implicit patterns, ICL is more adept at uncovering these latent patterns than fine-tuning.**

To investigate this phenomenon, we designed datasets containing implicit patterns across various domains, including two mathematical tasks: expression calculation and boolean function, one textual task: relation reasoning, and one code reading task. These domains share a common trait: the presence of implicit patterns that can simplify problem-solving. We evaluated LLMs' capability to recognize such patterns with these datasets. Our findings include: (1) Both fine-tuning and ICL could detect and utilize implicit patterns, resulting in increased test accuracy. (2) ICL performed much better than fine-tuning in implicit pattern detection, *e.g.,* ICL-based models enjoyed higher test accuracy. (3) ICL also showed strong performance in robustness tests and OOD data tests. Our experiments demonstrate that the ability of LLMs to leverage implicit patterns significantly enhances their problem-solving capabilities, providing a clear advantage for tasks involving complex data structures.

Understanding the operational principles of LLMs is crucial for their safety and ethical implications and can further promote improvements. Therefore, we delved deeper into the mechanisms behind this phenomenon. From a mechanistic interpretability perspective (Reddy, 2023), we proposed the **Circuit Shift** theory. Circuits are certain groups of attention heads and MLP layers (Conmy et al., 2023). A shift in circuits typically represents the model adopting a different method in problem-solving. Our findings indicated that ICL resulted in a larger-scale circuit shift compared to fine-tuning, which means that with ICL, the model changed its problem-solving method more significantly for implicit pattern detection and utilization. We also provided a visualized heatmap of circuits for detailed observation. In summary, our contributions are threefold:

Implicit Pattern Detection dataset. We defined and illustrated the implicit pattern detection task, then developed a dataset across mathematics (expression calculation, boolean function), textual reasoning (relation test) and code (output guessing).

Ability Comparison. We presented a counter-intuitive finding: LLMs with in-context learning detected implicit patterns much better than fine-tuned ones. We extensively tested this capability on models ranging from 0.5B to 7B parameters.

Mechanism explanation. We analyzed the principles behind the implicit finding mechanism. And we proposed circuit shift theory to explain why ICL finds implicit patterns better than fine-tuning.

2 Background

Transformer. Transformer (Vaswani et al., 2017) is the cornerstone architecture for LLMs nowadays, with its breathtaking ability in parallel training and sota performance. One Transformer model f_{trf} usually consists multiple of Transformer layers f_{layer} and an embedding layer f_{emb}. For an input sequence (typically IDs after tokenization) $\boldsymbol{X_0} \in R^{n\times 1}$ with length n, it first passes through an embedding layer f_{emb} with hidden state size d, then passes all the Transformer layers, and finally gets an output $\boldsymbol{O_l} \in R^{n\times d}$ with l layers: $\boldsymbol{O_l} = f_{\text{trf}}(\boldsymbol{X_0}) = \left(\bigcirc_{i=1}^{l} f_{\text{layer}}^{(i)}\right)(\boldsymbol{X_0})$, where for each layer f_{layer}, it usually contains an Attention block and an MLP block:

$$\boldsymbol{O_i^{\text{att}}} = \boldsymbol{X_i} + \text{Attn}(\text{Norm}(\boldsymbol{X_i})), \quad (1)$$
$$\boldsymbol{O_i} = \boldsymbol{O_i^{\text{att}}} + \text{MLP}(\text{Norm}(\boldsymbol{O_i^{\text{att}}})). \quad (2)$$

Here, $\boldsymbol{X_i^{\text{att}}}$ is the output of the attention block, and $\boldsymbol{X_i^{\text{mlp}}}$ is the output of the MLP block for layer i, with residual connections preventing it from gradient disappearance and normalization (typically pre-norm) for stabilizing the training process.

Fine-tuning. Fine-tuning is a process where a pre-trained LLM is further trained on a specific task or dataset to improve its performance for that particular application. Suppose there exists a pre-trained Transformer model f_{trf} with learnable parameters θ_{pre}. The goal of fine-tuning is to adjust these parameters to minimize a task-specific loss function

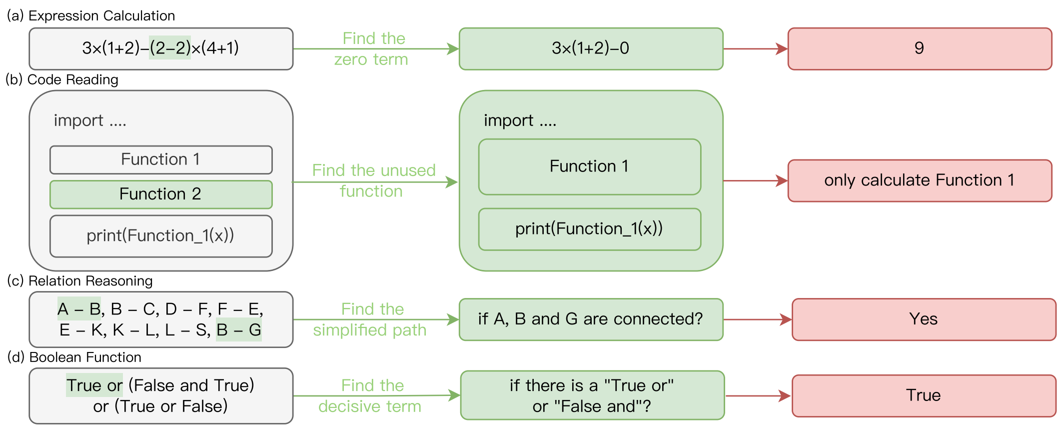

Figure 2: Examples of implicit pattern detection for four reasoning tasks. The implicit pattern, once detected, can reward the model with reduced computation to arrive at the answer.

$\mathcal{L}_{\text{task}}$ on a new dataset $\mathcal{D}_{\text{task}}$. During fine-tuning, the parameters θ_{fine} of the model are updated using gradient descent or one of its variants. The update rule for the parameters at each iteration t can be expressed as:

$$\theta_{\text{fine}}^{(t+1)} = \theta_{\text{fine}}^{(t)} - \eta \nabla_{\theta} \mathcal{L}_{\text{task}}(f_{\text{trf}}(\boldsymbol{X_t}; \theta_{\text{fine}}^{(t)}), \boldsymbol{Y_t}), \quad (3)$$

where η is the learning rate, $\boldsymbol{X_t}$ represents the input data in iteration t, $\boldsymbol{Y_t}$ represents the target labels in iteration t, and $\nabla_{\theta_{\text{fine}}} \mathcal{L}_{\text{task}}$ denotes the gradient of the loss function with respect to the model parameters. Fine-tuning typically requires substantial computational resources. For instance, full-parameter fine-tuning of LLaMA-3 with 8 billion parameters and an 8K context using the Adam optimizer and gradient checkpointing demands a minimum of 152 GB of VRAM (Rasley et al., 2020), which equates to at least two A100 80 GB GPUs with parallel training. While parameter-efficient fine-tuning (PEFT) is less resource-intensive compared to full-parameter fine-tuning, it still requires 16 GB of VRAM (QLoRA with a 1K context (Dettmers et al., 2024)), necessitating at least one RTX 3090 GPU. Additionally, some studies have shown that PEFT can result in a noticeable drop in the model's performance (Pu et al., 2023; Zou et al., 2023).

In-Context Learning In-Context Learning (ICL) in LLMs is an emergent capability where the model uses the provided context to perform tasks. Given a special task F and a series of prompt inputs $\boldsymbol{x_1}, \cdots, \boldsymbol{x_n}$, ICL happens when these inputs and their answers $\boldsymbol{y_1} = F(\boldsymbol{x_1})$ are given in multi-shot, *i.e.,* $(\boldsymbol{x_1}, \boldsymbol{y_1}, \cdots, \boldsymbol{y_n}, \boldsymbol{x_{n+1}})$. In this scenario, the goal for LLM to do ICL is to learn the task F and accurately predict $\boldsymbol{y_{n+1}}$. This phenomenon allows the model to adaptively handle a variety of tasks, such as translation, question-answering, and more, simply through appropriate prompt engineering. ICL happens in inference-stage without explicit re-training, thus resulting in more friendly requirements for GPUs (Yin et al., 2024; Hong et al., 2023). Even LLaMA-3 70B could run on a single 3090 GPU with PowerInfer (Song et al., 2023).

3 Implicit Pattern Detection Test

Through detailed observation and thinking, humans could detect some underlying, non-explicit patterns within the data. This enables us to solve problems more efficiently. Implicit pattern detection refers to the ability of models to recognize underlying, non-explicit patterns within data, enabling them to solve problems more efficiently. This concept is illustrated through tasks such as arithmetic calculations, where the model can bypass complex operations by identifying simplifying patterns. For instance, in mathematical expressions (see Figure 1 and Figure 2), a model might detect that certain terms have negligible impact and can be ignored, leading to quicker computations. We will give a detailed description of our dataset design and experimental settings in the following sections.

3.1 Tasks

To effectively assess the ability of LLMs to identify implicit patterns in data, we have constructed a variety of questions that frequently arise in real-world application scenarios. When the same type of question recurs, we can discover a specific implicit pattern within it to simplify the computational process.

Task 1: Expression Calculation (Imani et al., 2023; Yuan et al., 2023; Yue et al., 2023; He-Yueya et al., 2023) In the arithmetic calculation task, the primary focus is on determining whether certain operations within a given expression can be disregarded to reduce the complexity of the computation. The operations considered for these simplifications are limited to addition($+$), subtraction($-$), multiplication($\times$), and division($/$). By exploring these operations, the model may find that several terms are multiplied by a continued-to-be-zero term, and ignoring them could simplify the calculation process and improve the accuracy.

Task 2: Code Reading (Fang et al., 2024) In the code reading task, LLMs need to analyze and predict the output of a given piece of code without executing it, where multiple functions are defined. Some functions will not influence the final output, so the key challenge is to determine which functions are essential for producing the output and which can be disregarded without affecting the result.

Task 3: Boolean Functions (Zhang et al., 2024) In the Boolean functions task, the primary objective is to optimize logical expressions to simplify their structure without altering the resultant truth value. The expressions involve logical operators such as AND ($\wedge$), OR ($\vee$), and NOT ($\neg$). Within these scenarios, there are specific segments that are either tautologies, *i.e.,* always true, or contradictions, *i.e.,* always false. The model must identify these segments and bypass their computation.

Task 4: Relation Reasoning (Li et al., 2024) In the task of relation reasoning, the focus is on determining the relationships between multiple entities, such as reachability and relative magnitude. Although the set of relationships involved can be complex, all queries target fixed entities whose relationships are relatively straightforward. Therefore, most of the complex relationships can be disregarded, simplifying the problem-solving process.

3.2 Settings

Accuracy. Our tasks were constructed such that implicit patterns can help solve problems more easily. For example, if an LLM identifies a term that continues to be zero in arithmetic calculations, it can ignore terms multiplied by it, thereby saving computation. Therefore, we evaluate the model's performance with **Accuracy**.

Misleading Data. LLMs can detect the inner implicit patterns in data and utilize them for simplifying problem-solving. The misleading data is designed to test if LLMs can tackle situations in the absence of implicit patterns. While implicit patterns are still provided in training or ICL data, misleading data, *i.e.,* , data with no implicit patterns, is provided for testing accuracy. We name this accuracy **Misleading Accuracy**, while the testing results of data with implicit patterns are named **Clean Accuracy**. Detailed experimental procedures can be found in Appendix B.

Out-Of-Distribution Data. The training data are sampled from a certain distribution, *e.g.,* , for expression tasks, there are no more than 10 terms in each expression. Our out-of-distribution (OOD) data are designed to evaluate the model's performance when encountering OOD data during the evaluation phase. Detailed experimental procedures can be found in Appendix C.

Models. We select open-sourced models in sizes of 0.5B level *e.g.,* Qwen1.5-0m5B, 1B level *e.g.,* GPTNeo-1.3B (Black et al., 2021), Pythia-1.4B (Biderman et al., 2023), Qwen1.5-1.8B (Bai et al., 2023), and 7B level *e.g.,* Mistral-7B (Jiang et al., 2023), Qwen1.5-7B (Bai et al., 2023), Yi-6B (Young et al., 2024). Model weights are downloaded from Huggingface and follow the official implementations.

Data Format. For fine-tuning, the data is provided in a single example without supervised instruction. A simple description, the question, and the answer are given in order. We prepared 1,600 data points for fine-tuning. For in-context learning, we constructed the input in multi-shot, ranging from 0-shot, *i.e.,* directly answer one question, to 32-shot *i.e.,* 32 examples with their answers first given, then a new question in the same kind required to answer. The detailed example of our data format could be found in Appendix A.

Training Details. The training process was conducted using a sequence length of 512 and a batch size of 8 with a total of 1 epoch. A warmup phase of 20 steps was implemented, starting with a learning rate of 1e-6 and peaking at 2e-5, followed by a linear decay. The AdamW optimizer was used. This configuration ensured the model's performance and

Model	Expression			Code			Relation			Boolean		
	Baseline	Full-ft	ICL	Baseline	Full-ft	ICL	Baseline	Full-ft	ICL	Baseline	Full-ft	ICL
0.5B level												
Qwen1.5-0.5B	22.2%	**88.4%**	50.1%	16.6%	2.0%	**32.2%**	48.8%	48.5%	**60.1%**	54.8%	51.7%	**65.3%**
1B level												
GPTNeo-1.3B	24.3%	46.6%	**55.6%**	27.6%	17.7%	**44.5%**	20.5%	34.7%	**37.4%**	53.8%	53.7%	**54.3%**
Qwen1.5-1.8B	16.2%	**89.9%**	63.4%	54.3%	53.7%	**58.2%**	20.1%	21.3%	**35.6%**	66.3%	66.3%	**68.1%**
Pythia-1.4B	5.0%	45.4%	**53.7%**	37.6%	46.5%	**53.1%**	20.5%	31.3%	**44.4%**	61.3%	63.7%	**68.5%**
7B level												
Yi-6B	12.5%	**88.2%**	48.2%	51.2%	78.7%	**80.9%**	48.0%	52.5%	**98.0%**	55.7%	64.1%	**68.3%**
Qwen1.5-7B	78.0%	**89.3%**	67.9%	57.6%	72.0%	**86.8%**	48.0%	78.8%	**98.0%**	71.9%	41.7%	**79.8%**
Mistral-7B	32.6%	75.2%	**76.3%**	14.1%	72.0%	**82.8%**	48.5%	72.5%	**90.9%**	45.7%	54.5%	**74.3%**

Table 1: Experimental results of implicit pattern detection tasks. We conducted experiments from 0.5B to 7B across 6 models. The highest accuracy was highlighted with boldsymbol.

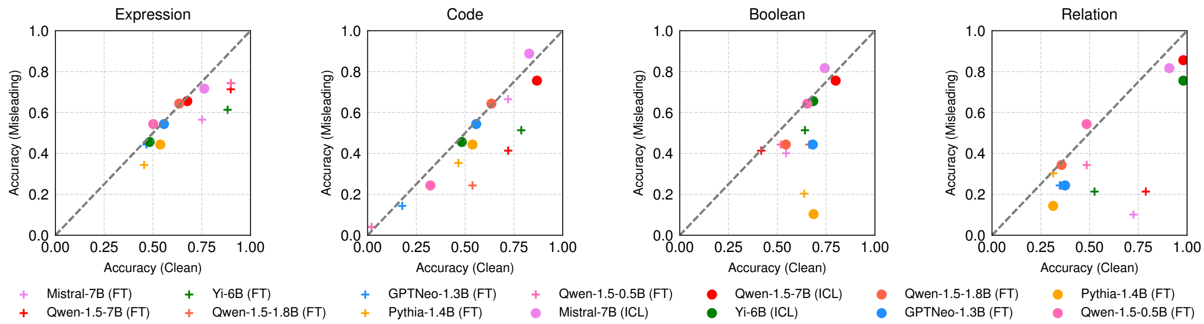

Figure 3: Robustness test of implicit pattern detection test. The horizontal axis represents the accuracy under clean input, and the vertical axis represents the accuracy under misleading input. Relatively speaking, the closer the results are to the bottom right corner, the worse the method's resistance to misleading data. The closer the results are to the top left corner, the better it is.

stability, allowing it to effectively learn and identify hidden patterns in the data.

4 Results and Analysis

In this section, we present our results for the implicit pattern finding tasks following the experimental setting in Section 3.2. We show that ICL achieved an overall higher level of accuracy over fine-tuning on these four tasks. We also show that the improvement of accuracy with ICL mainly comes from the detection of those implicit patterns in Section 5 and refsec:circuit.

4.1 ICL *v.s.* Fine-tuning: Accuracy

The results of accuracy test are shown in Table 1 and Table 2. Both ICL and fine-tuning(including full-param fine-tuning and PEFT methods) bring improvements to the performace of each task. However, it is easily noticed that ICL wins at most terms like relation, code reading and boolean functions, with 2% to even more than 30% improvements at most. On the flip side, fine-tuning only shows slight advantages in expression calculations in only Qwen-series models. As for different model size[1], we found that a larger model seems be able to evoke stronger ICL ability above linearly growth (see Table 1), where the scaling of fine-tuning performance is limited.

Method Type	Expression	Code	Relation	Boolean
Baseline	27.5%	54.3%	20.1%	66.3%
Full-Param FT	89.9%	53.7%	21.3%	66.3%
LoRA	46.5%	53.3%	20.1%	64.3%
QLoRA	46.2%	51.6%	20.5%	61.3%
GaLoRA	47.1%	52.5%	20.5%	66.4%
ICL	63.4%	58.2%	35.6%	68.1%

Table 2: Experimental comparison of different PEFT methods. We compared the results on Qwen1.5-1.8B. It is obvious that PEFT shows no significant improvement compared to full-param fine-tuning and seems to have limited performance.

[1] See Qwen1.5 series in Table 1 from 0.5B to 7B

OOD Type	Expression	Code	Relation	Boolean
Baseline	27.5%	54.3%	20.1%	66.3%
FT	89.9%	53.7%	21.3%	66.3%
FT + Test OOD	32.1%	34.2%	0.1%	0.1%
(FT+Test) OOD	88.2%	42.7%	11.3%	12.4%
ICL	63.4%	58.2%	35.6%	68.1%
ICL + Test OOD	34.5%	44.2%	12.3%	24.7%
(ICL+Test) OOD	62.3%	51.7%	34.5%	71.4%

Table 3: Experimental comparison of different PEFT methods. Here FT/ICL + Test OOD means we only applied OOD data in test phase, while (FT/ICL) OOD represents that both training/in-context learning and test phase were using OOD data.

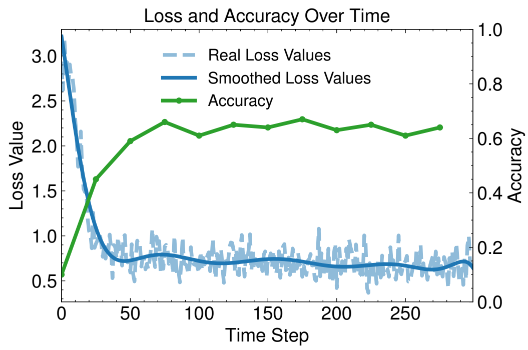

Figure 4: The progression of loss and accuracy over time during the fine-tuning of implicit pattern tasks. The Real Loss Values (dashed blue line) show the loss during training. To mitigate this noise, the Smoothed Loss Values (solid blue line) provide a clearer trend of the overall loss reduction. We also show the average test accuracy over all tasks (solid green line).

4.2 ICL *v.s.* Fine-tuning: Robustness without Implicit Pattern

In Section 3.2, we introduced the metrics of clean accuracy and misleading accuracy by adding misleading data to test both ICL and fine-tuning's robustness against general data without implicit patterns. The results are shown in Figure 3. For each task, we draw a scatter plot where the x- and y-axis represent the clean accuracy and the misleading accuracy, respectively. The results show that ICL can better exploit the implicit patterns in the demonstration data, while at the same time not compromising general reasoning abilities.

4.3 ICL *v.s.* Fine-tuning: Out-Of-Distribution Implicit Patterns

Out-of-Distribution (OOD) data is a widely examined problem nowadays. The training data of our implicit pattern detection tasks also samples from certain distributions (see Appendix C for details). In this subsection, we hope to compare how ICL and fine-tuning perform if we provide cases outside of the training distribution. For ICL, all examples given are divided into two types: in-distribution examples and OOD examples. For fine-tuning, we directly provide OOD problems to test the accuracy. We performed this experiment on Qwen1.5-1.8B and the results are demonstrated in Table 3. It is worth noticing that fine-tuning generally performs worse when the test data is OOD, while ICL performs fairly well comparing to the baseline method.

4.4 How Much Fine-tuning Do We Need?

In this experiment, we hope to figure out whether fine-tuning has reached its limit for implicit pattern detection or there will still be improvement if more data is utilized for fine-tuning. Therefore, we visualized the fine-tuning process of Qwen1.5-1.8B. At the onset of training, there is a steep decline in the loss value, suggesting that the model quickly learns basic patterns in the data. This rapid improvement is typical, as the model captures the most evident features. The Accuracy (solid green line) also increases sharply, corroborating the initial learning phase where the model transitions from random guessing to meaningful predictions. However, after around 50 time steps, both the loss and accuracy curves begin to stabilize. This period of stabilization suggests diminishing returns from further training, as the fine-tuned model failed to capture further implicit patterns. After 100 time steps, the curves indicate that the model has reached a plateau. The accuracy remains relatively constant, and the loss value shows minimal fluctuations around a stable trend. This behavior signifies that the model has learned the underlying patterns to a satisfactory extent, and additional fine-tuning yields marginal improvements.

4.5 Comparison of Fine-tuning with PEFT Methods

Lastly, we examine whether there is a significant difference between various fine-tuning methods *e.g.*, vanilla full-parameter fine-tuning, and parameter efficient fine-tuning (PEFT) methods like LoRA (Hu et al., 2021), QLoRA (Dettmers et al., 2024) and GaLoRE (Zhao et al., 2024). Although PEFT needs much less parameters for training, and several studies criticized its ability (Pu et al., 2023; Zou et al., 2023), there are still evidences that PEFT sometimes achieves ICL-level performance. We

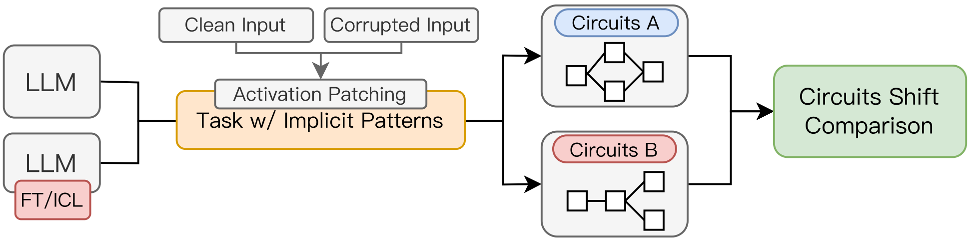

Figure 5: Illustration of circuit shift comparison. LLMs are first detected circuits with activation patching. Then we compare how much their circuits changed after fine-tuning and in-context learning.

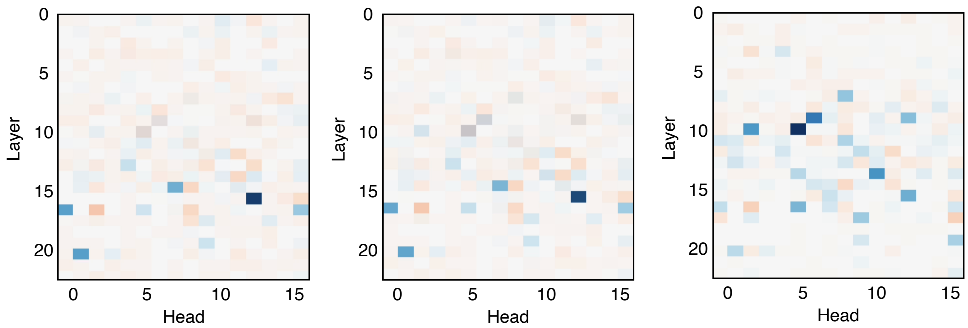

Figure 6: Visualization of attention head sensitivity in GPTNeo-1.3B. The more the color leans towards blue, the more important a specific attention head is to the implicit pattern detection task. Left: baseline model. Middle: fine-tuned model. Right: ICL model. It is clear that compared to fine-tuning, ICL brings significant circuit shifts.

followed the experimental settings in previous sections on Qwen1.5-1.8B with PEFT methods. The experimental results can be found in Table 2. It is clear that in the implicit pattern detection tasks, PEFT methods show no obvious advantages compared to full-param fine-tuning, thus they still failed to win ICL in accuracy in all tests.

5 Explanation of ICL's Victory: Circuits Shift Theory

Understanding the inner mechanisms of LLMs greatly benefits their ethical use and safety. We have found that ICL performs much better than fine-tuning on implicit pattern detection, and in this section, we try to explain why.

From a mechanistic interpretability perspective, we investigate this problem using **circuits**. Circuits are specific pathways (typically combinations of attention heads and MLP layers) within a model responsible for processing and interpreting particular patterns or tasks. The change in circuits for LLMs represents a shift in their inner mechanisms, revealing that LLMs choose different ways to solve problems. Based on this viewpoint, we propose a theory: **Circuits Shift**, to explain this phenomenon. We will first provide a method for probing circuits, explaining what they are and the types of circuits we found in ICL-based and fine-tuning-based models. Then we will show that the reason ICL performs better than fine-tuning is that the circuits in models experience a more significant shift. A detailed explanation of circuits and experimental settings can be found in Appendix D.

5.1 Method for Identifying Circuit Shift

In Figure 5, we present our framework and methodology for probing circuit shifts. We begin by selecting an implicit pattern detection task (in this study, we utilize an expression task). Subsequently, we use models employing different methods, *i.e.*, ICL or fine-tuning, for inference. During this process, we introduce corrupt input to randomly disrupt a portion of the activation to assess whether the corresponding attention heads or MLP layers significantly contribute to the final outcome. If a significant contribution exists, the disruption will result in considerable perturbation of the final logits, which is depicted as sensitivity in the figure.

5.2 Circuits Shift in LLMs for Implicit Pattern Detection

We first visualized and ranked circuits in GPTNeo-1.3B zero-shot, after fine-tuned, and ICL with 32-shot with expression calculation task (see Figure 6

Circuits	Zero-shot Baseline	ICL w/o Implicit Patterns	Δ	After Fine-tuning	Δ	After ICL	Δ
Attention	L17 H12, L18 H0 L22 H1, L16 H7 L18 H15, L14 H5	L17 H12, L16 H1 L18 H0, L15 H2 L18 H15, L22 H1	2	L17 H12, L18 H0 L22 H1, L16 H7 L18 H15, L12 H6	1	L11 H5, L10 H6 L11 H2, L15 H10 L17 H12, L 18 H5	**6**
MLP	L9 L17 L18	L9 L17 L18	0	L9 L18 L17	0	L17 L14 L15	**2**

Table 4: Top 6 Rankings of Attention Heads and top 3 rankings of MLP Layers in baseline (zero-shot) model, fine-tuned model, and ICL model. L is layer and H is head. Δ shows how many different heads or MLPs changed after fine-tuning or ICL. A larger Δ represents a more significant circuit shift in certain processes.

and Table 4). In Figure 6, we use the heatmap to illustrate the sensitivity of each attention head in implicit pattern detection test. From the figure, we can observe that, compared to the baseline and fine-tuning scenarios, ICL exhibits a significant shift when learning implicit patterns. Firstly, more shallow heads are involved in the task. Secondly, some deep heads that previously played a dominant role have now lost their leadership positions. This indicates that during the ICL process, the model significantly transforms its approach to solving the task, adapting to a form more suitable for implicit patterns, a phenomenon not observed with other methods.

We can further validate our hypothesis in Table 4. We selected the six attention heads and MLP layers[2] with the highest sensitivity, *i.e.,* those that contributed the most to the final result. Using the baseline, which is the zero-shot approach for handling implicit pattern detection tasks, as the standard, we counted how many new attention heads entered the top six highest contributors when the method changed, denoted by Delta. The results are very clear: compared to fine-tuning, ICL exhibits more significant changes, indicating a more thorough Circuit Shift during ICL. This suggests that ICL captures the characteristics of implicit patterns better than fine-tuning and adapts its processing method accordingly.

To rule out the inherent impact of ICL itself, we also conducted multi-shot experiments on a set of data without implicit pattern characteristics. The results showed that it is not multi-shot alone that induces this change, but rather the combined effect of ICL and implicit patterns.

[2] See A Mathematical Framework for Transformer Circuits for details.

6 Related Work

Implicit Pattern Discovery Previous works have designed benchmarks to test the LLMs reasoning ability (Barrett et al., 2018; Tang et al., 2023; Gendron et al., 2024). However, the benchmarks rarely include two-level questions where at one level, they can be solved by brutal force, at another level it can be solved by exploiting implicit patterns. The closest related work we know is Efrat et al. (2021), which involves solving cryptic crossword puzzles. To help the model find patterns in data, Prior work Sun et al. (2024); Zhu et al. (2024) proposes a two-stage induction-deduction process that first summarizes the common patterns explicitly, then reasons from the patterns.

ICL *v.s.* Fine-tuning Difference Previous works have also compared fine-tuning and in-context learning. Shen et al. (2024) shows that ICL is likely not an algorithmic equivalence to gradient descent for real LLMs. Reddy (2023) demonstrates that ICL is implemented by an induction head and analyzes its emergence phenomenon. Bhattamishra et al. (2023) shows that ICL and vanilla training implement two distinct algorithms that don't transfer to each other. However, it has been proven that fine-tuning shows better performance in generalization to OOD tasks than in-context learning (Mosbach et al., 2023).

7 Conclusion

In conclusion, our research demonstrates that In-Context Learning (ICL) significantly outperforms fine-tuning in capturing implicit patterns within specific tasks. Through our experimental evaluations, we observed that ICL not only enhances task performance more effectively but also exhibits greater adaptability in problem-solving approaches, as evidenced by the notable shifts in model circuits.

Limitations

Our study on the effectiveness of in-context learning in capturing implicit patterns compared to fine-tuning faces several limitations. Primarily, the generalizability of our findings is constrained by the specific nature of the implicit pattern detection tasks, which are limited to certain domains like arithmetic calculations, code reading, Boolean functions, and relation reasoning. Additionally, our analysis of Circuit Shift, which underpins the superior performance of ICL, relies on activation patching and sensitivity analysis, methods that, while insightful, require further refinement and validation across different models and tasks to confirm their robustness and applicability. Furthermore, the computational resources required for fine-tuning, especially with large models, may limit the feasibility of such experiments in broader settings, and a detailed cost-benefit analysis comparing ICL and fine-tuning in terms of computational efficiency and performance is needed.

References

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

David G. T. Barrett, Felix Hill, Adam Santoro, Ari S. Morcos, and Timothy Lillicrap. 2018. Measuring abstract reasoning in neural networks. *Preprint*, arXiv:1807.04225.

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

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. 2021. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow. If you use this software, please cite it using these metadata.

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

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

Avia Efrat, Uri Shaham, Dan Kilman, and Omer Levy. 2021. Cryptonite: A cryptic crossword benchmark for extreme ambiguity in language. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4186–4192, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Chongzhou Fang, Ning Miao, Shaurya Srivastav, Jialin Liu, Ruoyu Zhang, Ruijie Fang, Asmita, Ryan Tsang, Najmeh Nazari, Han Wang, and Houman Homayoun. 2024. Large language models for code analysis: Do llms really do their job? *Preprint*, arXiv:2310.12357.

Chengguang Gan and Tatsunori Mori. 2023. A few-shot approach to resume information extraction via prompts. In *International Conference on Applications of Natural Language to Information Systems*, pages 445–455. Springer.

Gaël Gendron, Qiming Bao, Michael Witbrock, and Gillian Dobbie. 2024. Large language models are not strong abstract reasoners. *Preprint*, arXiv:2305.19555.

Joy He-Yueya, Gabriel Poesia, Rose E. Wang, and Noah D. Goodman. 2023. Solving math word problems by combining language models with symbolic solvers. *Preprint*, arXiv:2304.09102.

Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Hanyu Dong, and Yu Wang. 2023. Flashdecoding++: Faster large language model inference on gpus. *arXiv preprint arXiv:2311.01282*.

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. *arXiv preprint arXiv:2106.09685*.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*.

Shima Imani, Liang Du, and Harsh Shrivastava. 2023. Mathprompter: Mathematical reasoning using large language models. *Preprint*, arXiv:2303.05398.

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

Zhiming Li, Yushi Cao, Xiufeng Xu, Junzhe Jiang, Xu Liu, Yon Shin Teo, Shang wei Lin, and Yang Liu. 2024. Llms for relational reasoning: How far are we? *Preprint*, arXiv:2401.09042.

Ziquan Liu, Yi Xu, Yuanhong Xu, Qi Qian, Hao Li, Xiangyang Ji, Antoni Chan, and Rong Jin. 2022. Improved fine-tuning by better leveraging pre-training data. *Advances in Neural Information Processing Systems*, 35:32568–32581.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*.

Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow, and Yanai Elazar. 2023. Few-shot fine-tuning vs. in-context learning: A fair comparison and evaluation. *arXiv preprint arXiv:2305.16938*.

Matthew E Peters, Sebastian Ruder, and Noah A Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. *arXiv preprint arXiv:1903.05987*.

George Pu, Anirudh Jain, Jihan Yin, and Russell Kaplan. 2023. Empirical analysis of the strengths and weaknesses of peft techniques for llms. *Preprint*, arXiv:2304.14999.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506.

Gautam Reddy. 2023. The mechanistic basis of data dependence and abrupt learning in an in-context classification task. *Preprint*, arXiv:2312.03002.

Lingfeng Shen, Aayush Mishra, and Daniel Khashabi. 2024. Do pretrained transformers learn in-context by gradient descent? *Preprint*, arXiv:2310.08540.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2023. Powerinfer: Fast large language model serving with a consumer-grade gpu. *arXiv preprint arXiv:2312.12456*.

Wangtao Sun, Haotian Xu, Xuanqing Yu, Pei Chen, Shizhu He, Jun Zhao, and Kang Liu. 2024. Itd: Large language models can teach themselves induction through deduction. *Preprint*, arXiv:2403.05789.

Xiaojuan Tang, Zilong Zheng, Jiaqi Li, Fanxu Meng, Song-Chun Zhu, Yitao Liang, and Muhan Zhang. 2023. Large language models are in-context semantic reasoners rather than symbolic reasoners. *Preprint*, arXiv:2305.14825.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Qingyu Yin, Xuzheng He, Xiang Zhuang, Yu Zhao, Jianhua Yao, Xiaoyu Shen, and Qiang Zhang. 2024. Stablemask: Refining causal masking in decoder-only transformer. *arXiv preprint arXiv:2402.04779*.

Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, et al. 2024. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*.

Zheng Yuan, Hongyi Yuan, Chuanqi Tan, Wei Wang, and Songfang Huang. 2023. How well do large language models perform in arithmetic tasks? *Preprint*, arXiv:2304.02015.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2023. Mammoth: Building math generalist models through hybrid instruction tuning. *Preprint*, arXiv:2309.05653.

Yuexiang Zhai, Shengbang Tong, Xiao Li, Mu Cai, Qing Qu, Yong Jae Lee, and Yi Ma. 2023. Investigating the catastrophic forgetting in multimodal large language models. *arXiv preprint arXiv:2309.10313*.

Yu Zhang, Hui-Ling Zhen, Zehua Pei, Yingzhao Lian, Lihao Yin, Mingxuan Yuan, and Bei Yu. 2024. Dila: Enhancing llm tool learning with differential logic layer. *Preprint*, arXiv:2402.11903.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. 2024. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*.

Zhaocheng Zhu, Yuan Xue, Xinyun Chen, Denny Zhou, Jian Tang, Dale Schuurmans, and Hanjun Dai. 2024. Large language models can learn rules. *Preprint*, arXiv:2310.07064.

Wentao Zou, Qi Li, Jidong Ge, Chuanyi Li, Xiaoyu Shen, Liguo Huang, and Bin Luo. 2023. A comprehensive evaluation of parameter-efficient fine-tuning on software engineering tasks. *Preprint*, arXiv:2312.15614.

A Data Format and Example

We provided examples of tasks and prompts. We provided data as 2-shot (code in zero-shot to restrict content length) for illustrating how ICL works. For fine-tuning we will use the same format but zero-shot in both training and inference.

Expression:

```
Now you need to calculate the answer of some mathematic equations.
Here are some examples:
(1+6)+(-3+3)*(-1-3+9-5)=7
(2+3)+(-1-4+5)*(10+6+2-8)=5
(8)+(0)*(0-6+9-6)=
```

Listing 1: Example

Code:

```
Now you need to give me the printed result after running this python code . Here are some examples:
```

Listing 2: Example

```python
def function1(x):
    y = x ** 9
    for i in range(1, 13):
        y = y * i - (y // (i + 9))
    return y

def function2(z, a):
    return z / 10

input_value = int(input())
result = function2(input_value, function1(input_value))
print(result)
```

Listing 3: Example

```
The input is 10, so the output is
```

Listing 4: Example

Relation:

```
Here are some cities expressed as A, B, C, etc. I will show some connection relations, and you need to tell me if city A and city Z are connected (Answer True or False).
Here are some examples:
A is connected with G
F is connected with J
J is connected with C
C is connected with B
B is connected with H
H is connected with E
E is connected with G
G is connected with I
I is connected with D
So 'the city A and Z is connected' is False
A is connected with B
H is connected with I
I is connected with G
G is connected with F
F is connected with E
E is connected with J
J is connected with B
B is connected with C
C is connected with D
B is connected with Z
So 'the city A and Z is connected' is True
A is connected with H
J is connected with I
I is connected with E
E is connected with F
F is connected with H
H is connected with G
G is connected with D
D is connected with C
C is connected with B
So 'the city A and Z is connected' is
```

Listing 5: Example

Boolean:

```
Here are some boolean expressions, you need to directly tell me the result. If it is true, print True, else print False. Here are some examples:
(True and False) and (True or False) and (False and False)\n The result is: False
(False and False) or (True and True) and (False and False)\n The result is: False
(True or True or True) and (False and True) and (True or True)\n The result is:
```

Listing 6: Example

B Misleading Data Construction

Expression. For the expression task, the inherent implicit pattern is an element that remains zero. When constructing the misleading dataset, we set this element to be non-zero. *i.e.,*

$$(3 + 2) + (4 - 1 + 5 - 6) \times (23 - 54 + 2) = ?$$

we constructed it as misleading data as:

$$(3 + 2) + (4 - 1 + 5 - 7) \times (23 - 54 + 2) = ?$$

Code. Here we provided two example about how to construct misleading code.

```python
def function1(x):
    y = x ** 19
    for i in range(1, 23):
        y = y * i - (y // (i + 19))
    return y

def function2(z, a):
    return z / 20

input_value = int(input())
```

Name	Type	Problem Example	Answer	Answer Type
Expression	Mathematic Calculation	$(6-1)+(6-6)*(-10+1+2+13)=$	5	Number
Code	Code Reading	`import math \n \n def function1(x): \n \n [TRUNCATED] return result \n print(result)`	3.5	Number
Relation	Textual Reasoning	A is connected with G\n F is connected[TRUNCATED] connected with Z, 'the city A and Z is connected' is	False	Boolean
Boolean	Mathematical Reasoning	(False or False) and (False or True) and False =	False	Boolean

Table 5: Examples of four implicit pattern detection tasks.

```
result = function2(input_value,
    function1(input_value))
print(result)
```

Listing 7: For implicit pattern

```python
def function1(x):
    y = x ** 19
    for i in range(1, 23):
        y = y * i - (y // (i + 19))
    return y

def function2(z, a):
    return z / 20

input_value = int(input())
result = function2(function1(input_value
    ), function1(input_value))
print(result)
```

Listing 8: For misleading

Relation. In the relation task, we generate misleading data by not setting shortcuts similar to A-G or G-Z.

```
A is connected with B
D is connected with B
B is connected with H
H is connected with F
F is connected with J
J is connected with I
I is connected with C
C is connected with G
G is connected with E
B is connected with Z
```

Listing 9: For implicit pattern

Here A-B-Z is a implicit pattern as shortcut for quick solving this problem. We remove this with a complex one:

```
A is connected with B
D is connected with B
B is connected with H
H is connected with F
F is connected with J
J is connected with I
I is connected with C
C is connected with G
G is connected with E
F is connected with Z
```

Listing 10: For misleading

Boolean. In the boolean task, we use combinations of OR + true and AND + false for quick evaluation. In the misleading data, we remove this characteristic.

```
(False and True) or (False or False) or
    True
```

Listing 11: For implicit pattern

```
(False and True) or (False or False) and
    True
```

Listing 12: For misleading

C OOD data Construction

	Min Terms	Max Terms	Range (abs value)
baseline	1	3	10
OOD	2	4	20

Table 6: Expression OOD

	Functions Need Calculation	Shortcut Nodes
baseline	1	3 (A to Any to G)
OOD	2	Unlimited

Table 7: Code OOD and Relation OOD

	If All AND or OR	Num of Terms
baseline	Yes	4
OOD	No	6

Table 8: Code OOD and Relation OOD

D Circuits

Circuits In mechanistic interpretability, our goal is to delineate how model components correlate with human-understandable concepts, an endeavor for which circuits provide a useful abstraction. Conceptualizing a model as a computational graph M, where nodes represent components like neurons, attention heads, and embeddings, and edges denote interactions such as residual connections and projections, a circuit C is defined as a subgraph of M responsible for a specific behavior, such as performing a task. This is a more coarse-grained approach compared to the feature-based.

Activation Patching Activation patching is a technique used to determine the importance of specific components within a model by manipulating their latent activations during model runs. The process involves three key steps: first, a *clean run* where the model processes a clean prompt, X_{clean} (*e.g.,* The Eiffel Tower is in), and associated answer r (Paris), during which activations of critical components such as MLP or attention heads are cached; second, a *corrupted run* where the model is run on a corrupted prompt, X_{corrupt} (*e.g.,* The Colosseum is in), to record baseline outputs; and third, a *Patched run* where the model is run on X_{corrupt} again, but with specific cached activations from the X_{clean} run restored. This setup allows for the evaluation of the patching effect, which measures the restoration of model performance by comparing outputs from the Corrupted and Patched runs. The patching effect is quantitatively assessed using different metrics with probability gap:

$$P_{\text{patched}}(r) - P_{\text{corrupt}}(r) \tag{4}$$

and logit difference:

$$LD(r, r') = \log\left(\frac{P(r)}{P(r')}\right)_{\text{patched}} - \log\left(\frac{P(r)}{P(r')}\right)_{\text{corrupt}} \tag{5}$$

This technique is crucial for understanding and improving model reliability and performance by highlighting the roles of individual model components.

E A detailed Definition of Implicit Pattern Detection

Consider a problem P characterized by a fixed complexity function C_P. For each input x in the domain D, there exists a solution y. A implicit pattern for problem P, denoted as P_{shortcut}, is defined as follows:

- P_{shortcut} is either a subproblem of P or an independent problem where the domain D_{shortcut} is a subset of D (*i.e.,* $D_{\text{shortcut}} \subseteq D$).
- For any input x in D_{shortcut}, the output y_{shortcut} of P_{shortcut} approximates the output y of P.
- The complexity of solving P_{shortcut}, $C_{P_{\text{shortcut}}}$, is significantly less than C_P (*i.e.,* $C_{P_{\text{shortcut}}} \ll C_P$).

If these conditions are met, then P_{shortcut} is considered a shortcut of P. We define its complexity C_f in terms of the accuracy of a LLM performing on f. Let Acc_f represent the accuracy of the LLM on task f, then the complexity $C_T f$ can be defined as: $C_T = 1 - \text{Acc}_f$ The complexity C_f ranges from 0 (no complexity, as the task is perfectly solved) to 1 (maximum complexity, as the task is not solved at all).

This definition implies that the higher the LLM's accuracy on a task, the lower the complexity of the task. This measure allows us to quantify task complexity based on the performance capabilities of state-of-the-art language models.