Task Vectors in In-Context Learning: Emergence, Formation, and Benefits

Liu Yang[w], **Ziqian Lin**[w], **Kangwook Lee**[w], **Dimitris Papailiopoulos**[w,m] **& Robert D. Nowak**[w]
[w]University of Wisconsin-Madison, [m]Microsoft Research
{liu.yang, zlin284, kangwook.lee}@wisc.edu,
dimitris@papail.io, rdnowak@wisc.edu

Abstract

In-context learning is a remarkable capability of transformers, referring to their ability to adapt to specific tasks based on a short history or context. Previous research has found that task-specific information is locally encoded within models, though their emergence and functionality remain unclear due to opaque pre-training processes. In this work, we investigate the formation of task vectors in a controlled setting, using models trained from scratch on synthetic datasets. Our findings confirm that task vectors naturally emerge under certain conditions, but the tasks may be relatively weakly and/or non-locally encoded within the model. To promote strong task vectors encoded at a prescribed location within the model, we propose an auxiliary training mechanism based on a *task vector prompting loss (TVP-loss)*. This method eliminates the need to search for task-correlated encodings within the trained model and demonstrably improves robustness and generalization.

1 Introduction

To understand the underlying mechanisms of in-context learning in transformers, researchers have probed pre-trained models from various perspectives, such as altering the labels in demonstrations [Min et al., 2022, Kim et al., 2022] and investigating circuit mechanisms [Elhage et al., 2021, Wang et al., 2023, Hanna et al., 2024, Singh et al., 2024]. Additionally, controlled, small-scale studies have been conducted by training transformers from scratch to observe their in-context learning behavior on linear regression tasks [Garg et al., 2022, von Oswald et al., 2022, Lin and Lee, 2024], discrete functions [Bhattamishra et al., 2023], hidden Markov chains [Xie et al., 2021], and DFAs [Akyürek et al., 2024]. Furthermore, theoretical approaches have also been applied to this problem [Xie et al., 2021, Lin and Lee, 2024, Giannou et al., 2023].

Among the various efforts to probe pre-trained models, one significant line of research employs the concept of a "task vector", which is a vector in the model's activation or weight space that encodes task-specific information. The concept of the task vector was first introduced by Ilharco et al. [2022], where it is defined as a direction in a model's weight space corresponding to a particular task. Subsequently, Hendel et al. [2023] demonstrated that, given a task demonstration as context, a pre-trained large language model forms a task vector in its activation space[1] at certain layers. This task vector encodes only the task information and is independent of the specific demonstration of the task. By inserting the task vector directly into the model, it is able to perform the task without context or demonstration (i.e., zero-shot). We will refer to this as *Task Vector Prompting* (TVP) in this paper. Concurrently, Liu et al. [2023], Todd et al. [2023], Merullo et al. [2023a], Li et al. [2024b], Saglam et al. have also identified a single vector that encodes the task information, albeit using different terminology. We omit the details here and refer the reader to the related work and the original papers for more information.

[1]The "activation space" refers to the space where the output of each transformer layer resides.

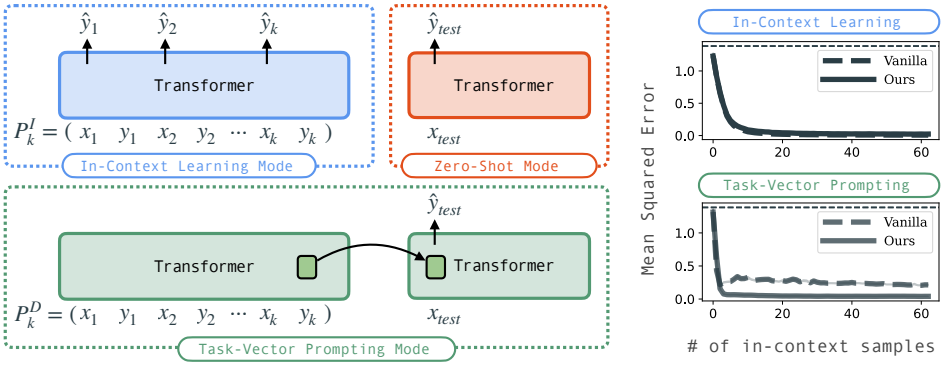

Figure 1: *Overview of the transformer operating in in-context learning (ICL) and task vector prompting (TVP) modes.* A transformer can be configured to operate in ICL mode, using input-output pairs as prompts, or in TVP mode, extracting task-specific embeddings for zero-shot predictions (Architecture and training details in Section 2). On the right, the ICL and TVP performances of the vanilla-trained model and our method are shown, with the dashed horizontal line indicating random prediction performance (i.e., no task information is inferred). Compared to vanilla training, our approach enhances task-specific representations in the TVP mode while preserving comparable ICL performance.

To better understand the emergence of task vectors, we examine in-context learning behavior in a controlled setting, training models from scratch on various synthetic datasets. As shown in Figure 1, when trained from scratch on the linear regression task defined as $y_i = \boldsymbol{w}^T \boldsymbol{x}_i$, the transformer (dashed line) demonstrates the ability to use in-context learning (ICL) to solve the task (top-right). We evaluate the trained model's performance in task vector prompting (TVP) mode (Figure 1, bottom-left), where the task vector is extracted from the in-context learning mode and injected back in a zero-shot manner, as defined by Hendel et al. [2023]. The TVP performance is much better than chance, but a bit worse than ICL performance, which we attribute to the fact that the encoding of the task may not be strong and localized by normal training methods. To encourage the formation of a strong and localized task vector, we propose an auxiliary training loss, called the *task vector prompting loss (TVP-loss)*. In our new approach, the model is trained using the TVP-loss in addition to normal training losses. As illustrated in Figure 1, our approach (solid line) achieves comparable ICL performance to the vanilla model in the ICL mode (top-right) and its TVP performance is significantly improved and comparable to ICL performance with multi-shot demonstration.

The remainder of the paper is organized as follows. Section 2 provides the formal definition of task-vector prompting. In Section 3, we analyze the characteristics of task vectors in trained-from-scratch transformers and identify that the encoded task vectors remain noisy. Section 4 introduces a training algorithm to encourage the formation of task vectors. Finally, we extend our analysis to synthetic formal language tasks in Section 5. Our main contributions and findings are outlined below:

Emergence of Task Vectors During Training. We investigate the effectiveness of task vector extraction methods, originally proposed for pre-trained large language models, when applied to small-scale models trained from scratch on synthetic datasets. Our study reveals that task vectors can naturally emerge during training, provided that specific input formats are used and the model has sufficient capacity. To understand the conditions that promote this emergence, we examine different hyper-parameters such as the model depth, the use of sparse attention, and input embedding sizes.

Strong Task Vectors with Proposed Auxiliary Loss. Although task vectors naturally emerge, they are often weak and entangled with information from input queries. This prevents task vectors from representing purely task-specific knowledge, as noted by Hendel et al. [2023]. To overcome this limitation, we propose a training algorithm that explicitly encourages the formation of task vectors independent of query information. This approach strengthens the task vector, ensuring that task-specific encoding is explicitly established within the model. Another benefit is that our approach specifies the embedding location of the task vector within the model, eliminating uncertainty about where it might emerge.

Task Vectors for In-Context Learning Robustness. Using our proposed training algorithm, we analyze models trained on synthetic tasks, both with and without the enhanced task vectors at certain layers. We then assess the advantages of having strong task vectors in the model, demonstrating their robustness in in-context learning performance and examining the effects of task vectors forming in different layers.

2 Task Vector Definition

In this section, we formally define the notion of task vector. Let $\mathcal{F}$ denote a class of functions or "tasks", and let $\mathcal{X}$ and $\mathcal{Y}$ be the input and output spaces, respectively. If $f \in \mathcal{F}$ is the task in a specific "context", then for any input $\boldsymbol{x} \in \mathcal{X}$, the corresponding output is $\boldsymbol{y} = f(\boldsymbol{x}) \in \mathcal{Y}$. Consider $\boldsymbol{x}_{\text{test}} \in \mathcal{X}$, and the transformer model M. We can measure the model's zero-shot performance by $\ell(f(\boldsymbol{x}_{\text{test}}), M(P_{\text{query}}))$, where $P_{\text{query}} = [\boldsymbol{x}_{\text{test}}]$ denotes the query input. The in-context learning performance with k-shot examples is measured by $\ell(f(\boldsymbol{x}_{\text{test}}), M(P_k))$, where P_k is the in-context prompt. The prompt contains the k-shot examples $[\boldsymbol{x}_1, f(\boldsymbol{x}_1), \cdots, \boldsymbol{x}_k, f(\boldsymbol{x}_k), \boldsymbol{x}_{\text{test}}]$.

Now consider a *demonstrated prompt* P_k^D, where the superscript D highlights its role in demonstrating task information. This prompt contains k in-context samples:

$$P_k^D = [\boldsymbol{x}_1, f(\boldsymbol{x}_1), \cdots, \boldsymbol{x}_k, f(\boldsymbol{x}_k)]$$

A task vector is an internal embedding $\boldsymbol{\tau}$ extracted when the model is presented with P_k^D, which encodes the task at hand (f). Inserting the task vector into the model during zero-shot prompting is denoted by $M(P_{\text{query}}; \boldsymbol{\tau})$. It is worth noting that the task vector $\boldsymbol{\tau}$ is extracted from the internal embeddings when inputting P_k^D, not P_k, therefore the task vector $\boldsymbol{\tau}$ does not have explicit knowledge of $\boldsymbol{x}_{\text{test}}$ when encoding the task.

A task vector extractor g tries to locate this single embedding $\boldsymbol{\tau}$ in the model given the demonstrated prompt P_k^D, i.e. $\boldsymbol{\tau} = g(M(P_k^D))$. Then the performance of this extractor can be measured by $\ell(f(\boldsymbol{x}_{\text{test}}), M(P_{\text{query}}; \boldsymbol{\tau}))$. A task vector is considered to be successfully formed in the model M for the task f if the performance of the extracted task vector, i.e. task vector prompting performance, closely aligns with the in-context learning performance and surpasses the zero-shot performance.

Hendel et al. [2023] confirmed that, in pre-trained large language models, $g(M(P_k^D))$ corresponds to the output embedding at approximately the middle layers of the model during the forward pass, which is responsible for predicting $f(\boldsymbol{x}_{\text{test}})$. Similarly, Sia et al. [2024] found that the task in context is recognized by the model during the middle stage of the forward pass.

3 Emergence of Task Vectors in Trained-from-Scratch Models

In this section, we investigate whether task vectors can be identified in a transformer model trained from scratch on synthetic tasks.

3.1 Experimental Setup

We investigate the localization of the task vector using the GPT-2 decoder model [Radford et al., 2019] trained on the generated random prompts in the following tasks:

Linear Regression. Following the setup in [Lin and Lee, 2024, Garg et al., 2022], the input vectors $\boldsymbol{x}_i \in \mathbb{R}^d$ are sampled from a normal distribution $\mathcal{N}(0, \boldsymbol{I})$. The linear function weights $\boldsymbol{w} \in \mathbb{R}^d$, which define the task, are sampled from a mixture of d Gaussians, each with means corresponding to the standard basis vectors: $\boldsymbol{\mu}_i \in \mathbb{R}^d$ with $\boldsymbol{\mu}_{ij} = 1$ if $i = j$, and 0 otherwise. All Gaussians share a covariance matrix $\Sigma = \frac{1}{4}\boldsymbol{I}$, and each component is selected with equal probability. The target outputs are computed as $f(\boldsymbol{x}_i) = \boldsymbol{w}^\top \boldsymbol{x}_i$. We set $d = 6$ in our experiments.

Sinusoidal Regression. Following the same sampling strategy as the linear regression task, the target outputs are defined as $f(\boldsymbol{x}_i) = \sin(\boldsymbol{w}^\top \boldsymbol{x}_i)$. In this task, $\boldsymbol{w}$ determines the underlying function.

Discrete Token Offset Prediction. In this task, we set the vocabulary size to be C, and each input token $\boldsymbol{x}_i$ is a discrete value drawn from the set $\{0, 1, \ldots, C-1\}$. The target output is defined as $f(\boldsymbol{x}_i) = (a \times \boldsymbol{x}_i + b) \bmod C$, where $a \in \{1, 2, 3\}$, $b \in \{0, 1, 2\}$ are uniformly sampled, and $\bmod$ denotes the modulo (remainder) operation. The task is uniquely determined by the choice of a and b, resulting in total 9 possible tasks. By default, we set $C = 1000$.

During training, prompts are generated on the fly. Specifically, for each prompt, a function $f \in \mathcal{F}$ is randomly sampled according to the function distribution described for each task above. Subsequently, input tokens $\{\boldsymbol{x}_i\}_{i=1}^k$ are independently sampled from the corresponding input distribution. The function f is then evaluated on these inputs to produce the target outputs, forming the in-context learning prompt $P_k = [\boldsymbol{x}_1, f(\boldsymbol{x}_1), \ldots, \boldsymbol{x}_k, f(\boldsymbol{x}_k), \boldsymbol{x}_{\text{test}}]$. Let the distribution of such prompts be denoted as $\mathcal{P}$. The transformer M parameterized by θ is then trained to minimize the following expected loss:

$$\min_{\theta} \mathbb{E}_{P_k \sim \mathcal{P}} \sum_{i=1}^{k} \ell(M_\theta(P_i), f(\boldsymbol{x}_{i+1})).$$

where $P_i = [\boldsymbol{x}_1, f(\boldsymbol{x}_1), \ldots, \boldsymbol{x}_i, f(\boldsymbol{x}_i), \boldsymbol{x}_{i+1}]$ represents the prompt prefix containing i in-context examples. In our experiments, we use GPT-2 model with an embedding size of 64, 4 attention heads, and 3 layers, trained with a maximum context length of 63 (i.e., $k = 63$) and without positional embeddings (NoPE). We use the Adam optimizer with a learning rate of 0.0001 and a batch size of 256, training for 300k iterations. With dynamically generated prompts, this corresponds to 76.8 million distinct prompts. All experiments are run on an NVIDIA GeForce RTX 3090.

We investigate various input formats and task vector extraction methods (details in Appendix B.1 and Appendix B.2) to identify conditions under which task encoding emerges in trained-from-scratch transformers. Based on this investigation, we focus on the following setup, where task vectors emerge most distinctly.:

- *input format*: Prompts with k in-context examples are generated in the format $P_k = [\boldsymbol{z}, \boldsymbol{x}_1, f(\boldsymbol{x}_1), \ldots, \boldsymbol{x}_k, f(\boldsymbol{x}_k), \boldsymbol{x}_{\text{test}}]$, where $\boldsymbol{z}$ is a special token placed at the beginning of the prompt to serve as a placeholder for injecting task encoding during zero-shot task vector prompting. During training, the embedding of $\boldsymbol{z}$ is treated as a learnable parameter that is shared across all prompts.
- *task vector extractor method*: inspired by Hendel et al. [2023], we locate the task vectors *at the activations of the token $\boldsymbol{z}$ and $\{y_i\}$* in the input format mentioned above. Specifically, during task vector prompting, the embedding is copied into the corresponding position in a zero-shot model, and the zero-shot loss is measured. A lower zero-shot loss indicates that the embedding effectively encodes task-specific information (details in Appendix B.1).

As noted in Hendel et al. [2023], the extraction process involves identifying the layer that optimally encodes the task vector. The optimal layer index $l^\star$ is determined by evaluating task vectors extracted from each layer and selecting the one that minimizes the average loss ℓ across all N demonstrated prompts. This procedure ensures that the selected layer provides the most effective task vector for predicting the test outputs.

Formally, for each task f, we generate N demonstrated prompts $\{P_k^{D,j}\}_{j=1}^N$, where the same function f is consistently used across all N prompts, and the input to each prompt $P_k^{D,j}$ is uniformly sampled. Let P_{query}^j, with test input $\boldsymbol{x}_{\text{test}}^j$, represents the corresponding test queries for the j-th demonstrated prompt. For a task vector extracted from a k-shot prompt, the optimal layer index $l^\star$ is computed as:

$$l^\star = \arg\min_{l} \sum_{j=1}^{N} \ell\left(f(\boldsymbol{x}_{\text{test}}^j), M\left(P_{\text{query}}^j; g(M(P_k^{D,j}), l)\right)\right)$$

where $g(M(P_k^{D,j}), l)$ denotes the task vector extracted from the l-th layer when processing the demonstrated prompt $P_k^{D,j}$, and $M(P_{\text{query}}^j; \boldsymbol{\tau})$ represents the model's prediction for the query prompt P_{query}^j, where the task vector $\boldsymbol{\tau}$ extracted from the l-th layer is copied to the corresponding position in the model's representation when processing P_{query}^j. A lower loss ℓ indicates that the corresponding layer provides a more effective encoding of task-specific information.

Remark (Task Vector Location). *The work of Hendel et al. [2023] employs an in-context learning prompt format structured as* $P_k = [\boldsymbol{x}_1, \boldsymbol{z}, f(\boldsymbol{x}_1), \cdots, \boldsymbol{x}_k, \boldsymbol{z}, f(\boldsymbol{x}_k), \boldsymbol{x}_{k+1}, \boldsymbol{z}]$*, where* $\boldsymbol{z}$ *indicates the "maps-to" token. This setup differs from the prompt format used in our experiments. As detailed in Appendix B.1, we investigate various input formats to determine their effect on task vector emergence. We find that the trained-from-scratch model exhibits task vector emergence only when the input format for training is* $P_k = [\boldsymbol{z}, \boldsymbol{x}_1, f(\boldsymbol{x}_1), \cdots, \boldsymbol{x}_k, f(\boldsymbol{x}_k), \boldsymbol{x}_{k+1}]$*, where no additional tokens are placed between* $\boldsymbol{x}$ *and* $f(\boldsymbol{x})$*.*

We hypothesize that the discrepancy arises because pre-trained LLMs leverage the semantic meaning of the "maps-to" token as an anchor for task summarization, whereas in trained-from-scratch transformers, this extra "maps-to" token serves only as a computational placeholder, with fixed token positions negating the need for a delimiter.

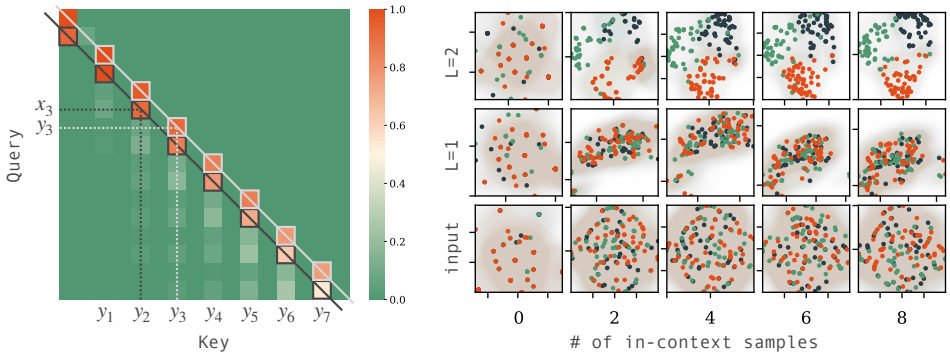

Figure 2: *Attention map and PCA visualization of activations across three different linear functions.* (**Left**) For the linear regression task described in Section 3.1, the attention map illustrates how each query (rows) attends to all available keys (columns), with each row summing to 1. The heatmap reveals that the activations at the $\boldsymbol{x}_i$ positions predominantly attend to the activations of the preceding y_{i-1}, where task information is stored (highlighted by the black boxes). Additionally, the activations at the y_i positions attend to both themselves and the preceding y_{i-1}, enabling the online updating of task information (highlighted by the white boxes). (**Right**) PCA visualizations of token y_i activations ($i \in \{0, 2, 4, 6, 8\}$) across layers L reveal that task-specific clusters (three colors correspond to three different tasks) begin to emerge at the output of the 2nd layer, indicating that the model progressively encodes task information as depth increases.

3.2 Task Vector Emergence

Finding 1: *Task vectors naturally emerge in small models trained from scratch.*

In Figure 1, we demonstrate the presence of task encoding in the activations of token y_i under the linear regression task, using the experimental setup described in Section 3.1 with number of demonstrated prompts $N = 50$. In the task vector prompting mode, the transformer achieves a mean squared error (MSE) of approximately 0.25, significantly lower than the random prediction baseline of around 1.2, indicating that the model successfully infers task information in its representation. Throughout the paper, we use $N = 50$ by default unless stated otherwise. Additionally, we examine the impact of different N values on task vector prompting performance, as detailed in Appendix B.6.

To understand the non-random performance observed in the task vector prompting mode (Figure 1), we analyze how the model encodes task information using attention maps (*left*) and a PCA of activations at each layer while varying the number of in-context examples (*right*), as shown in Figure 2. In transformer models, attention mechanisms enable each input token to dynamically focus on specific parts of the input data by assigning importance to them. This process can be visualized using attention maps, which quantify the relationships between input tokens. These maps are computed using query and key vectors: the query represents the current focus of interest, while the key interact with the query to calculate importance scores (via their dot product), determining how relevant each input element is to the query.

Recall that we follow the experimental setup in Section 3.1 and conduct experiments on the linear regression task using a 3-layer transformer with a 63-shot context. On the left, the attention map of a

specific head in 2nd layer shows that the activations at $\boldsymbol{x}_i$ position primarily attend to the activations of the preceding y_{i-1}, where task information is stored. Furthermore, the activations at the y_i location attend to both themselves and the preceding y_{i-1}, enabling the model to update task information online. On the right, we present a PCA of the activations for token y_i ($i \in \{0, 2, 4, 6, 8\}$) across layers L, evaluated on three different linear functions: $\boldsymbol{w}_1 = [1, 0, 0, 0, 0, 0]^T$, $\boldsymbol{w}_2 = [0, 0, 1, 0, 0, 0]^T$, and $\boldsymbol{w}_3 = [0, 0, 0, 0, 1, 0]^T$. At the input to the third layer, the activations for different tasks begin to form distinct clusters, though there is still considerable overlap and dispersion among the clusters. We also perform this analysis for the input format used in Hendel et al. [2023] and present the results in Figure 10 in Appendix B.1. In this format, we found no evidence of task encoding.

In the following sections, we analyze the impact of model depth and context length on task vector emergence across different problem dimensions (Section 3.2.1) and investigate the locations of task vectors (Section 3.2.2). Additionally, we explore the effects of embedding size and sparse attention in Appendices B.3 and B.4, respectively.

3.2.1 Effects of Model Depth and Context Length on Task Vector Emergence

In the linear regression task, we examine how model depth and context length affect the emergence of task vectors by varying the problem dimension. Instead of fixing $d = 6$, we explore $d \in \{4, 5, 6, 7, 8, 9\}$, keeping the transformer's embedding size at 64 and adjusting the model depth from $L = 3$ to $L = 8$. The transformer's performance in ICL mode (solid line) and TVP mode (line with triangular markers) is shown in Figure 3, with $N = 50$ for clarity.

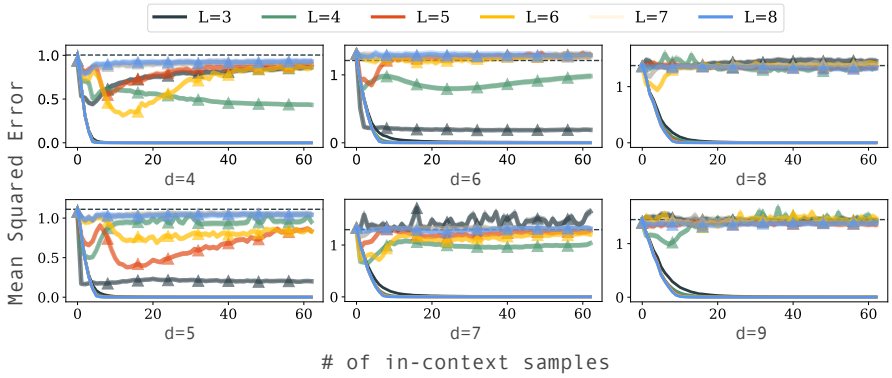

Figure 3: *Performance of the transformer in in-context learning (ICL) mode (solid line) and task vector prompting (TVP) mode (line with triangular markers) across problem dimensions $d \in \{4, 5, 6, 7, 8, 9\}$ and varying model depths L.* The dashed line indicates the random-guess baseline. The results indicate that smaller problem dimensions ($d = 4$ to $d = 7$) and shallower model depths ($L = 3$ to $L = 5$) yield stronger and more stable task encoding in the TVP mode. However, task encoding remains noisy in most cases. Task vectors emerge more clearly when the ICL loss plateaus but tend to disperse with increasing in-context length. In deeper models, task information appears to distribute across layers, reducing the distinctiveness of task vectors.

Finding 2: *In the linear regression task, task vectors are most distinct with moderate model depth and just before the in-context learning loss plateaus.*

We observe the following characteristics of task vectors:

Along the Model Depth: Tasks with smaller problem dimensions ($d = 4$ to $d = 7$) exhibit a clear pattern of specific model depths producing strong and stable task encoding, as illustrated in the plot. However, in most cases, task encoding remains noisy. Notably, as model depth increases, the clarity of task encoding diminishes. This could be due to the model's increased capacity to approximate the least square solution directly in a single forward pass, reducing its reliance on previously calculated task encoding.

Across the In-Context Length: Task vectors become more distinct as the in-context learning loss plateaus, but they tend to disperse afterward. For example, when $d = 5$ with $L = 4$ or $d = 7$ with

$L = 4, 5, 6$, task encoding is most evident around the 4-th in-context example but then disperses into random task encodings. This suggests that the model initially learns the task in a compact and focused manner; however, over time, the task information becomes distributed more broadly across the context.

3.2.2 Task Vector Layer Localization

Finding 3: *Depending on the tasks, task vectors may emerge in intermediate or later layers.*

When extracting the task vector, the extractor must determine which layer the task vector resides in, based on the N demonstrated prompts. For linear regression task with the experimental setup described in Section 3.1, we record the proportion of each layer being chosen as the location of the task vector and present the results in Figure 4 (upper left). As suggested by the PCA visualization of activation clusters in Figure 2, among the 3 layers, the 2nd layer is predominantly selected as the location where the task vector resides.

To further clarify, we also measure model's performance in the TVP mode when the extracted task vector is placed at each layer and average the loss across varying context lengths. These results, shown in Figure 4 (lower left), demonstrate that the selected task vector layer achieves significantly lower TVP performance compared to other layers, confirming that the emergence of the task vector is meaningful and not due to marginal differences in loss across layers.

This observation contrasts with the phenomenon reported in pre-trained LLMs by Hendel et al. [2023], where task vectors for in-context learning tasks primarily emerge in the early layers out of a total of around 20 layers. In our experiments, however, we find that task vectors consistently emerge in the penultimate layer for shallow models (e.g., 3 layers). As the model depth increases (e.g., 4 or more layers), task vectors no longer emerge distinctly at any single layer, as shown in Figure 3. Instead, task information becomes distributed across multiple layers, reducing the distinctiveness of any specific layer as the task vector's location. For additional details, Figure 13 in Appendix B.5 extends this analysis to the linear regression task with $d = 6$ and model depths $L = 4, 5, 6$.

One factor to consider is the nature of the task at hand. For the linear function class, the "task" appears to be identifying the parameter $\boldsymbol{w}$, after which the "task execution" phase simply requires the model to compute $\boldsymbol{w}^T \boldsymbol{x}$ to generate predictions. To explore this further, we examine the sinusoidal regression task using the same GPT-2 model configured with an embedding size of 64, 4 attention heads, and 6 layers. Additionally, we analyze the discrete token offset task with a larger embedding size of 256.

As shown in Figure 4 (middle), for this 6-layer transformer model, the 3rd layer is primarily selected as the task vector location. This finding aligns with observations in pre-trained LLMs and provides the insight that the task vector's emergence layer in trained-from-scratch models depends on the characteristics of the task being performed. Additionally, for the discrete token offset task, the 5th layer is most frequently selected as $l^\star$, with other layers also being selected occasionally at lower probabilities.

4 A New Training Algorithm to Encourage the Formation of Task Vector

As evidenced in the previous section, it is difficult to locate a single vector in the trained-from-scratch small-scale transformer model that cleanly encodes the task at hand. In this section, we propose a training algorithm that explicitly encourages the formation of task vectors (as defined in Hendel et al. [2023]) in the in-context learning process.

Through this algorithm, which is a straightforward extension of the task vector definition, we obtain a model with the task vector explicitly formed. As demonstrated in Section 4.2, comparing this model to the vanilla model (where task vectors are not explicitly formed) reveals that task vector formation enhances the model's robustness in in-context learning tasks, particularly in scenarios with out-of-distribution prompts. This highlights the value of task vector formation in improving generalization and interpretability in in-context learning.

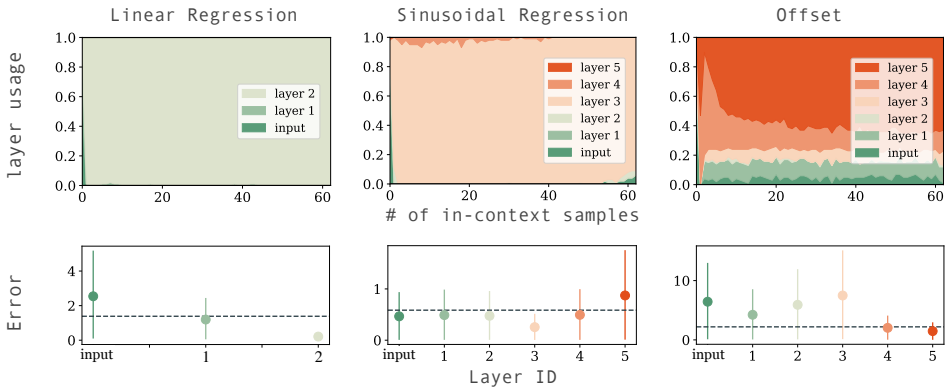

Figure 4: *Layer selection distribution (upper row) and averaged task vector prompting performance (lower row) for task vector emergence in: Linear Regression (left), Sinusoidal Regression (middle), and Discrete Token Offset (right).* **Upper row:** we show the frequency of each layer being selected as the task vector location across varying numbers of in-context examples. For the linear regression task on a 3-layer transformer, the task vector predominantly emerges in the 2nd layer. For the more complex sinusoidal regression task on a 6-layer transformer, the task vector shifts to the 3rd layer, while for the discrete token offset task, it is primarily found in the last layer. These results suggest that task complexity influences the depth at which the task vector emerges. **Lower row:** we present the averaged task vector prompting (TVP) performance for each layer, measured across various context lengths. The dashed line represents random-guess performance, indicating no task information is inferred. Notably, the selected task vector layer demonstrates significantly lower loss in TVP mode compared to other layers, confirming that the emergence of the task vector is meaningful rather than due to marginal differences in loss across layers.

4.1 A New Training Algorithm to Encourage Task Vector Formation

To encourage the formation of task vectors, we include the performance metric of the task vector in the training loss. Specifically, following the notation in Section 3, let each random prompt be $P_k = [\boldsymbol{z}, \boldsymbol{x}_1, f(\boldsymbol{x}_1), \cdots, \boldsymbol{x}_k, f(\boldsymbol{x}_k), \boldsymbol{x}_{k+1}]$ to be sampled following the prompt distribution $\mathcal{P}$, and a random test prompt $[\boldsymbol{z}, \boldsymbol{x}_{test}]$. We train the transformer M_θ by optimizing the following loss:

$$\min_\theta \mathbb{E}_{P\sim\mathcal{P}} \sum_{i=1}^{k} \left[\underbrace{\ell(M_\theta(P_i), f(\boldsymbol{x}_{i+1}))}_{\text{ICL-loss}} + \underbrace{\ell(f(\boldsymbol{x}_{\text{test}}), M_\theta([\boldsymbol{z}, \boldsymbol{x}_{\text{test}}]; \boldsymbol{\tau}))}_{\text{TVP-loss}} \right], \tag{1}$$

where $\boldsymbol{\tau} = h_i^l$ represents the hidden state at the l-th layer of the transformer, extracted from the i-th in-context example. Instead of adaptively identifying the layer where the task vector resides, we simplify the process by directly designating a specific layer l as the location for forming the task vector. This ensures consistent representation of task-specific information and facilitates training. In Section 4.2, we further explore the impact of this hyper-parameter choice on the model's performance.

The loss comprises two complementary components: (1) **In-Context Learning Loss (ICL-loss)**: The term $\ell(M_\theta(P_i), f(\boldsymbol{x}_{i+1}))$ trains the model to predict the output $f(\boldsymbol{x}_{i+1})$ for the $(i+1)$-th example, based on the preceding context P_i. Summing over k examples ensures the model learns from the entire context effectively. (2) **Task Vector Prompting Loss (TVP-loss)**: The term $\ell(f(\boldsymbol{x}_{\text{test}}), M_\theta([\boldsymbol{z}, \boldsymbol{x}_{\text{test}}]; \boldsymbol{\tau}))$ with $\boldsymbol{\tau} = h_i^l$ evaluates the model's ability to use the injected task vector h_i^l, derived from the hidden state at the i-th in-context example, to predict the test output $f(\boldsymbol{x}_{\text{test}})$. This term encourages the model to encode task-specific information in the task vector $\boldsymbol{\tau}$. As evidence of this, in Figure 1, although the model trained with ICL-loss already exhibits emergence of task vector, the model with TVP-loss further improves the encoded task vector, achieving a lower TVP error. We illustrate the training algorithm in Figure 5, where the red arrow indicates the flow of the gradient.

4.2 Experimental Results on Synthetic Tasks

We apply this training algorithm to the aforementioned three synthetic tasks: (1) the linear regression task, (2) the sinusoidal regression task, and (3) the discrete token offset task described in Section 3.1.

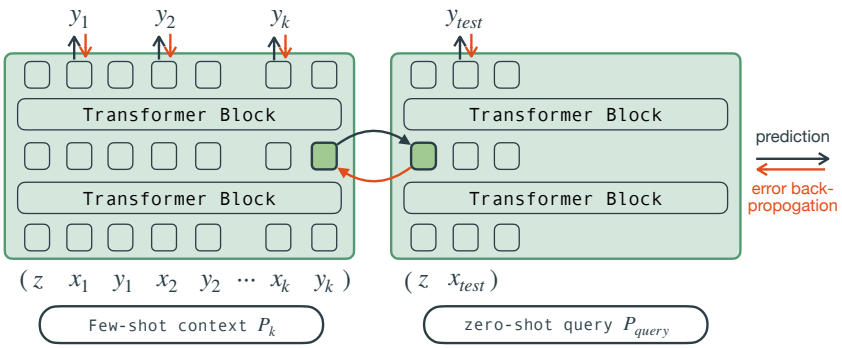

Figure 5: *Demonstration of our training algorithm.* In vanilla Meta-ICL training, the model is updated using the ICL-loss signal from the *few-shot context*. To encourage the formation of task vectors, we also explicitly include the TVP-loss from the *zero-shot query*. This means the model is asked to predict y_{test} when only $\boldsymbol{x}_{\text{test}}$ and the injected hidden states are given. In the given illustrated example, there are in total 2 layer in the transformer model, and we set $l = 1$ to encourage the formation of the task vector at the first transformer block's output.

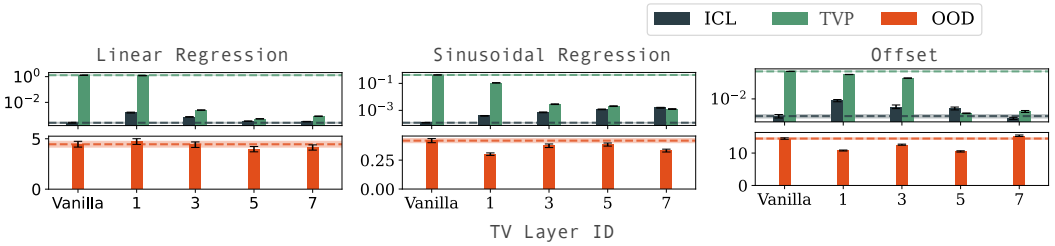

Figure 6: *Model performance with vanilla and TVP-loss training.* This figure compares the performance of models trained with vanilla training and those trained with TVP-loss across different layers where task vectors are formed. The upper row shows in-context learning (ICL) and task vector prompting (TVP) performance, while the bottom row illustrates out-of-distribution (OOD) prompt performance. Dashed lines represent the performance of vanilla-trained models for reference. Each column corresponds to a specific task: linear regression (left), sinusoidal regression (middle), and discrete token offset (right). The horizontal axis indicates the layer in which the task vector is formed, which impacts TVP performance but not ICL performance. **ICL performance:** Models trained with TVP-loss achieve comparable ICL performance to vanilla-trained models. **TVP performance:** Models trained with TVP-loss exhibit improved TVP performance, particularly when task vectors are formed in intermediate or later layers of the model. **OOD robustness:** Models with TVP-loss demonstrate stronger generalization to OOD prompts compared to vanilla-trained models, particularly in more complex tasks like sinusoidal regression and discrete token offset, and when task vectors are formed at specific layers.

For all experiments, the transformer model is configured with a total of 8 layers. For consistency to the previous results, we set $N = 50$ here as well.

Finding 4: *The proposed TVP-loss enhances task vector prompting performance, aligning it closely with in-context learning performance.*

For clarity, we evaluate the in-context learning and task vector prompting performance at the 63rd context (i.e., the final query) for models trained with and without the proposed TVP-loss. These results remain consistent across various context lengths, with full details provided in Appendix C.1. In this experiment, we designate task vectors to form at the 1st, 3rd, 5th, and 7th layers. As illustrated in Figure 6, for the 8-layer transformer, the task vector prompting performance of the vanilla-trained model is nearly random. In contrast, models trained with the TVP-loss exhibit task vector prompting performance that closely matches their in-context learning performance, as long as the task vector layer is set to an intermediate or later layer rather than the initial layers of the model.

Performance on OOD Prompts. Forcing the task vector to form at a specific layer can be interpreted as introducing an implicit bottleneck architecture within the model's forward pass. In this section, we examine the model's generalization ability when the task vector is constrained to

form at a particular layer. For the three synthetic tasks described earlier, we evaluate the following out-of-distribution (OOD) tasks:

1. *Linear Regression*: We introduce an OOD prompt using the quadratic regression task, defined as $f(\boldsymbol{x}_i) = \boldsymbol{w}^\top(\boldsymbol{x}_i \cdot \boldsymbol{x}_i)$, where $\cdot$ denotes element-wise multiplication.
2. *Sinusoidal Regression*: we consider the OOD prompt where the task is $f(\boldsymbol{x}_i) = \sqrt{\boldsymbol{w}^T\boldsymbol{x}_i}$. Considering sinusoidal regression as a composite function, with $f_1(\boldsymbol{x}_i) = \boldsymbol{w}^T\boldsymbol{x}_i$, and $f_2(y) = \sin(y)$, we replace the $f_2(y) = \sin(y)$ with $f_2(y) = \sqrt{y}$ in this OOD task.
3. *Discrete Token Offset*: the OOD prompt switches the context at the 6th position.

Finding 5: *Strong task vectors improve in-context learning performance on out-of-distribution prompts.*

Additional OOD tasks and their results are presented in Appendix C.2. Note that during training, none of these OOD settings were encountered during training. As shown in Figure 6, models with task vectors formed at specific layers exhibit equal or improved performance on OOD prompts compared to vanilla-trained models. For the linear regression task, forming the task vector at the 5-th and 7-th layer results in slightly better OOD performance.

In the sinusoidal regression task, which involves compositional functions, models with task vectors show enhanced generalization to the modified compositional task, indicating that capturing task representations at intermediate layers improves OOD generalization. This aligns with findings from Kobayashi et al. [2024], Schug et al. [2023], where the use of an explicit bottleneck architecture in compositional tasks demonstrated improved compositional generalization ability.

In the discrete token offset task, models with task vectors handle context changes significantly better than those trained without the auxiliary loss. This suggests that the auxiliary loss enables the model to adapt more effectively by filtering out unrelated context at earlier positions, compared to vanilla-trained models.

5 In-Context Learning for Formal Language

In this section, we extend our study to synthetic formal language settings and investigate the impact of forming task vectors within the model. We conduct experiments on the following two benchmarks: the Generative IN-Context learning dataset (GINC) introduced in Xie et al. [2021] and the RegBench dataset introduced in Akyürek et al. [2024].

5.1 The GINC Dataset: the Factorial Hidden Markov Chain.

Experimental Setup. The GINC dataset is introduced in Xie et al. [2021] to study the behavior of language pretraining and the emergence of in-context learning. We follow the setup as described in Xie et al. [2021]: we define a uniform mixture of HMMs over a family of five concepts. Each entity and property is assigned unique tokens, resulting in a total vocabulary size of 100. To generate samples, we first select an HMM from the mixture and use it to produce documents containing 576 tokens. Next, we randomly insert 192 dummy tokens, denoted as "-," into these documents. These dummy tokens are labeled as `-100` to be ignored during processing, resulting in documents with a total length of 768 tokens.

We use the GPT-2 model with 8 layers, 8 heads, and an embedding dimension of 256. The context length of this transformer is set to 768. To train the model to form a task vector at the l-th layer, we sample 10 extra tokens from the same HMM and randomly insert a dummy token "-". This forms the "zero-shot" document. The task vector loss is then calculated by injecting the l-th layer's hidden states at the token "-" into the "-" token in the zero-shot document. This training pipeline (shown in Figure 17) differs from the Meta-ICL pipeline described earlier (Figure 5) because GINC specifies a pre-trained distribution that is different from the prompt distribution. In GINC, there are no explicit input-output pairs; instead, the entire trajectory performs one task. Therefore, we encode the task vector into the inserted token "-" in the trajectory.

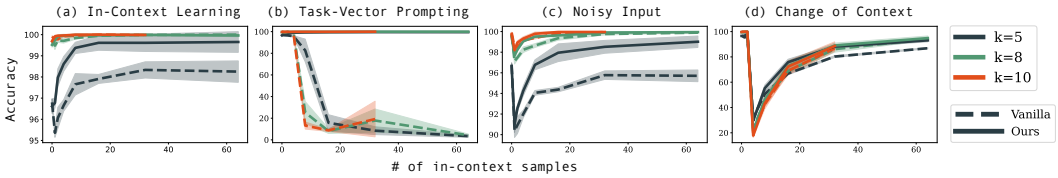

Figure 7: *GINC dataset in-context learning performance.* Comparison of the in-context learning performance for the vanilla-trained model (dashed line) and the model trained with TVP-loss to encourage the formation of task vectors at the 5th layer (solid line). From left to right, we measure: (a) in-context learning, (b) task vector prompting, (c) in-context learning with noisy labels, and (d) in-context learning with changes in the context. In all cases, the model with task vector formation outperforms the vanilla-trained model. We follow the definition from Xie et al. [2021] in letting k to be the number of example presented in each context sample. Please refer the detailed setup in Section 5.1 and in Xie et al. [2021].

To evaluate in-context learning performance, we follow the setup described in Xie et al. [2021] and sample prompts from the five concepts used during training. An example of an in-context prompt with a context length of $n = 3$ and example length $k = 4$ is: `a b c - d / e f g - h / i j k - l`. Similarly, an example zero-shot prompt with example length $k = 4$ would be: `m n o - p`. Here, the input $\boldsymbol{x}$ represents a short document, such as `a b c`, and the task f corresponds to an underlying hidden Markov model. The output $f(\boldsymbol{x})$ is the next sampled token in the document. Unlike the synthetic tasks described earlier, this function f is stochastic.

To evaluate task vector prompting performance, we extract the hidden states at the i-th "`-`" token and inject them into the l-th layer activations of the zero-shot prompt at its "`-`" token. This setup aligns with the input format described in Hendel et al. [2023]. Notably, because the first "`-`" token in the in-context prompt appears after an initial document of the HMM (e.g., the `a b c` document in the earlier example) and is then injected into the zero-shot prompt, additional context information from this HMM document may be included, which could lead to better task vector prompting performance compared to in-context learning performance. Through out the experiment, we set $N = 1$, i.e. the task vector is extracted from a single demonstrated prompt.

In Context Learning Results. Though the model is not Meta-ICL trained, when training with the TVP-loss, during in-context learning, the task vector can still be formed. We present the result in Figure 7. From left to right, we measure both model's performance on (a) in-context learning, (b) task vector prompting, (c) noisy in-context learning, where the in-context token is replaced with a random token with probability 0.1, and (d) when the underlying HMM is changed at the second context.

Among all the in-distribution and out-of-distribution setting, model trained with TVP-loss consistently out-perform the vanilla-trained model. Specifically, the results indicate: (a) the effectiveness of picking up in-context learning skills when the model is only trained on next-token prediction on documents; (b) the successful formation of the task vector even when this in-context learning task is not directly optimized during pre-training; (c) and (d) indicating the robustness to noise in the input prompt when trained with TVP-loss.

5.2 The RegBench Dataset: the Probabilistic Finite Automata.

Experimental Setup. The RegBench dataset is introduced in Akyürek et al. [2024] to study the behavior of language pretraining and the emergence of n-gram head. We follow and revise the setup of Akyürek et al. [2024]: we sample 100 deterministic finite automata (DFAs) for training. At each training iteration, we randomly generate 5,000 training sequences from these DFAs. Each DFA has a fixed initial state, and each state transits to another state given an input token. For each DFA, the total vocabulary size (i.e., the number of unique tokens) is set to 40, the maximum number of states is set to 20, and each state is allowed a maximum of 10 outgoing edges. To generate samples (sequences) from a DFA, we convert it to a probabilistic finite automaton (PFA) by assigning uniform transition probabilities across all outgoing edges, following Akyürek et al. [2024].

Each prompt contains 32 samples (sequences) generated by the sampled DFA and those example are with length k separated by the special token "`|`". To help the formation of task vectors, we insert a dummy token "`>`" before the last token of the example. After inserting `>`, a prompt with context length $n = 3$ and example length $k = 4$ would be like "`abc>d|efg>h|ijk>l`".

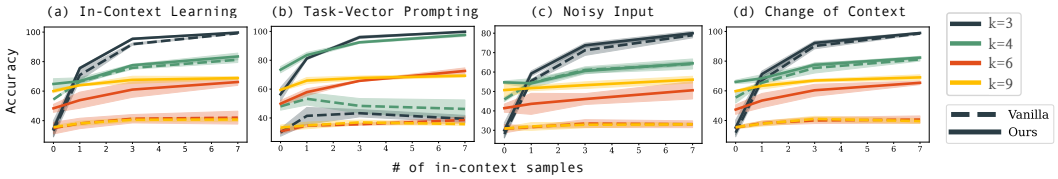

Figure 8: *RegBench dataset in-context learning performance.* Comparison of the in-context learning performance for the vanilla-trained model (solid line) and the model trained with TVP-loss at the 4th layer (dashed line). From left to right, we measure: (a) in-context learning, (b) task vector prompting, (c) in-context learning with noisy labels, and (d) in-context learning with changes in the context. In all cases, the model with TVP-loss outperforms the vanilla-trained model.

We use the GPT-2 model with 8 layers, 2 heads, and an embedding dimension of 128 following Akyürek et al. [2024]. For each training sequence, we sample an additional example from the same DFA and insert a dummy token > before the last token of the example. This serves as a zero-shot prompt. Similar to the GINC dataset, the input $\boldsymbol{x}$ corresponds to the generated sequence of tokens from the PFA. The task f is determined by the PFA's transition function, and the output $f(\boldsymbol{x})$ is the next token sampled by the DFA. Importantly, this transition function is stochastic as well. During training, we use the next token prediction loss on all tokens except the special tokens ">" and "|". During inference, we evaluate task vector prompting performance with $N = 1$.

In Context Learning Results. We test with various $k = 3, 4, 6, 9$, and present the results in Figure 8, which include: (a) in-context learning (ICL) performance, (b) task vector prompting (TVP) performance, (c) noisy in-context learning, where 4 out of 32 in-context labels are replaced with random tokens, and (d) performance when the underlying DFA of 4 out of 32 samples is changed to another DFA.

For evaluating the next predicted token in the in-context learning prompt, we follow the same setup as in Akyürek et al. [2024]: as long as the next predicted token belongs to the outgoing edge set of the current DFA state, it is considered a correct prediction; otherwise, it is incorrect. Intuitively, as k increases, more transitions are presented to the transformer. This requires the model to approximate a larger n-gram distribution, leading to a degradation in performance.

As shown in the Figure 8, models trained with TVP-loss show a slight improvement in ICL performance (as shown in (a)) and a significant improvement in TVP performance (b). Furthermore, (c) and (d) demonstrate that models trained with task vectors are more robust to label noise and out-of-distribution (OOD) in-context samples. Specifically, these models maintain better performance compared to vanilla training when in-context examples include label noise or OOD samples.

6 Discussion and Conclusions

Factors Affecting Task Vector Emergence. The emergence of the task vector in pre-trained large language models may be attributed to multiple factors, such as the model's capacity, scaling laws, and the diverse tasks encountered during pre-training. While Hendel et al. [2023] and related studies identify and analyze the existence of task vectors, they do not study the factors that may effect their emergence and performance. This gap in understanding motivates our investigation: we reproduce this phenomenon in the small-scale setting when trained from scratch, gaining insight into the factors influencing task vector emergence, such as input format, model depth, and context length. For instance, in the linear regression task, when the model is trained with prompts where x and y alternate closely, a model with moderate depth encourages the emergence of task vectors, which is most evident before the in-context learning loss plateaus.

Training with TVP-Loss. While task vectors can emerge through normal training processes, adding the TVP-loss to the training process encourages the formation of task vectors at prescribed locations and leads to improved accuracy and robustness. Across various benchmark datasets, we have demonstrated that prescribing the task vector to an intermediate or later layer of the model results in comparable in-context learning performance while forming strong task vectors. Depending on the task, task vectors formed at specific layers can enhance the model's robustness to out-of-distribution (OOD) prompts, most likely at the intermediate or later layer.

Potential Applications of Task Vectors Task vectors can serve as a soft prompt, effectively compressing the entire context into a single vector representation. In practical scenarios, where the number and nature of tasks are unknown during pre-training and only demonstrations are provided to the model, task vectors enable both context summarization and task identification. Specifically, a model with task vectors formed gains the ability to identify the underlying task from demonstrations, cluster tasks using the extracted task vectors, and subsequently perform zero-shot inference with these extracted vectors.

Another practical benefit is during inference, attention computations over earlier tokens can be masked out after the prescribed l-th layer, restricting attention to only the task vector and the query. This approach mirrors the findings of Sia et al. [2024], who demonstrated a 45% reduction in computation for pre-trained LLMs. Looking ahead, incorporating the TVP-loss as an auxiliary objective during training offers a promising approach to enhance task-specific representation and overall performance.

7 Acknowledgement

The work of Robert Nowak is supported in part by NSF awards DMS-2134140, CIF-2427440, and ONR MURI N00014-20-1-2787. The work of Dimitris Papailiopoulos is supported in part by ONR Grant No. N00014-21-1-2806 and No. N00014-23-1-2848. The work of Kangwook Lee is supported in part by NSF CAREER Award CCF-2339978, Amazon Research Award, and a grant from FuriosaAI.

References

E. Akyürek, D. Schuurmans, J. Andreas, T. Ma, and D. Zhou. What learning algorithm is in-context learning? investigations with linear models. *ArXiv*, abs/2211.15661, 2022. URL https://api.semanticscholar.org/CorpusID:254043800.

E. Akyürek, B. Wang, Y. Kim, and J. Andreas. In-context language learning: Architectures and algorithms. *ArXiv*, abs/2401.12973, 2024. URL https://api.semanticscholar.org/CorpusID:267095070.

Y. Bai, F. Chen, H. Wang, C. Xiong, and S. Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. *ArXiv*, abs/2306.04637, 2023. URL https://api.semanticscholar.org/CorpusID:259095794.

Y. Bai, H. Huang, C. S.-D. Piano, M.-A. Rondeau, S. Chen, Y. Gao, and J. C. K. Cheung. Identifying and analyzing task-encoding tokens in large language models. 2024. URL https://api.semanticscholar.org/CorpusID:267068913.

R. Belanec, S. Ostermann, I. Srba, and M. Bieliková. Task prompt vectors: Effective initialization through multi-task soft-prompt transfer. *ArXiv*, abs/2408.01119, 2024. URL https://api.semanticscholar.org/CorpusID:271693681.

I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

S. Bhattamishra, A. Patel, P. Blunsom, and V. Kanade. Understanding in-context learning in transformers and llms by learning to learn discrete functions. *ArXiv*, abs/2310.03016, 2023. URL https://api.semanticscholar.org/CorpusID:263620583.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. J. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020.

S. C. Y. Chan, I. Dasgupta, J. Kim, D. Kumaran, A. K. Lampinen, and F. Hill. Transformers generalize differently from information stored in context vs in weights. *ArXiv*, abs/2210.05675, 2022a. URL https://api.semanticscholar.org/CorpusID:252846775.

S. C. Y. Chan, A. Santoro, A. K. Lampinen, J. X. Wang, A. K. Singh, P. H. Richemond, J. Mcclelland, and F. Hill. Data distributional properties drive emergent in-context learning in transformers. *ArXiv*, abs/2205.05055, 2022b. URL https://api.semanticscholar.org/CorpusID:248665718.

D. Dai, Y. Sun, L. Dong, Y. Hao, S. Ma, Z. Sui, and F. Wei. Why can gpt learn in-context? language models implicitly perform gradient descent as meta-optimizers. 2022. URL https://api.semanticscholar.org/CorpusID:258686544.

G. Dar, M. Geva, A. Gupta, and J. Berant. Analyzing transformers in embedding space. *arXiv preprint arXiv:2209.02535*, 2022.

N. Elhage, N. Nanda, C. Olsson, T. Henighan, N. Joseph, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, N. DasSarma, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

E. Elmoznino, T. Marty, T. Kasetty, L. Gagnon, S. Mittal, M. Fathi, D. Sridhar, and G. Lajoie. In-context learning and occam's razor. *arXiv preprint arXiv:2410.14086*, 2024.

S. Garg, D. Tsipras, P. Liang, and G. Valiant. What can transformers learn in-context? a case study of simple function classes. *ArXiv*, abs/2208.01066, 2022.

M. Geva, A. Caciularu, K. Wang, and Y. Goldberg. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. *ArXiv*, abs/2203.14680, 2022a. URL https://api.semanticscholar.org/CorpusID:247762385.

M. Geva, A. Caciularu, K. R. Wang, and Y. Goldberg. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. *arXiv preprint arXiv:2203.14680*, 2022b.

A. Giannou, S. Rajput, J. yong Sohn, K. Lee, J. D. Lee, and D. Papailiopoulos. Looped transformers as programmable computers. *ArXiv*, abs/2301.13196, 2023.

S. Goyal, Z. Ji, A. S. Rawat, A. K. Menon, S. Kumar, and V. Nagarajan. Think before you speak: Training language models with pause tokens. *ArXiv*, abs/2310.02226, 2023. URL https://api.semanticscholar.org/CorpusID:263608983.

S. Han, J. Song, J. Gore, and P. Agrawal. Emergence of abstractions: Concept encoding and decoding mechanism for in-context learning in transformers. 2024. URL https://api.semanticscholar.org/CorpusID:274789679.

M. Hanna, S. Pezzelle, and Y. Belinkov. Have faith in faithfulness: Going beyond circuit overlap when finding model mechanisms. *ArXiv*, abs/2403.17806, 2024. URL https://api.semanticscholar.org/CorpusID:268691935.

Y. He, H. S. Zheng, Y. Tay, J. Gupta, Y. Du, V. Aribandi, Z. Zhao, Y. Li, Z. Chen, D. Metzler, H.-T. Cheng, and E. H. Chi. Hyperprompt: Prompt-based task-conditioning of transformers. In *International Conference on Machine Learning*, 2022. URL https://api.semanticscholar.org/CorpusID:247218062.

R. Hendel, M. Geva, and A. Globerson. In-context learning creates task vectors. *ArXiv*, abs/2310.15916, 2023. URL https://api.semanticscholar.org/CorpusID:264439386.

A. Hojel, Y. Bai, T. Darrell, A. Globerson, and A. Bar. Finding visual task vectors. *ArXiv*, abs/2404.05729, 2024. URL https://api.semanticscholar.org/CorpusID:269005382.

S.-C. Huang, P.-Z. Li, Y.-C. Hsu, K.-M. Chen, Y. T. Lin, S.-K. Hsiao, R. T.-H. Tsai, and H. yi Lee. Chat vector: A simple approach to equip llms with instruction following and model alignment in new languages. In *Annual Meeting of the Association for Computational Linguistics*, 2023. URL https://api.semanticscholar.org/CorpusID:268253329.

G. Ilharco, M. T. Ribeiro, M. Wortsman, S. Gururangan, L. Schmidt, H. Hajishirzi, and A. Farhadi. Editing models with task arithmetic. *ArXiv*, abs/2212.04089, 2022. URL https://api.semanticscholar.org/CorpusID:254408495.

P. Kalyan, P. Rao, P. Jyothi, and P. Bhattacharyya. Emotion arithmetic: Emotional speech synthesis via weight space interpolation. *Interspeech 2024*, 2024. URL https://api.semanticscholar.org/CorpusID:271741273.

J.-H. Kang, J. Lee, M.-H. Lee, and J.-H. Chang. Whisper multilingual downstream task tuning using task vectors. *Interspeech 2024*, 2024. URL https://api.semanticscholar.org/CorpusID:272332568.

J. Kim, H. J. Kim, H. Cho, H. Jo, S.-W. Lee, S. goo Lee, K. M. Yoo, and T. Kim. Ground-truth labels matter: A deeper look into input-label demonstrations. *ArXiv*, abs/2205.12685, 2022. URL https://api.semanticscholar.org/CorpusID:249062718.

S. Kobayashi, S. Schug, Y. Akram, F. Redhardt, J. von Oswald, R. Pascanu, G. Lajoie, and J. Sacramento. When can transformers compositionally generalize in-context? *ArXiv*, abs/2407.12275, 2024. URL https://api.semanticscholar.org/CorpusID:271244618.

J. Kossen, Y. Gal, and T. Rainforth. In-context learning learns label relationships but is not conventional learning. 2023. URL https://api.semanticscholar.org/CorpusID:260125327.

J. Lee, A. Xie, A. Pacchiano, Y. Chandak, C. Finn, O. Nachum, and E. Brunskill. Supervised pretraining can learn in-context reinforcement learning. *ArXiv*, abs/2306.14892, 2023. URL https://api.semanticscholar.org/CorpusID:259262142.

B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. In *Conference on Empirical Methods in Natural Language Processing*, 2021. URL https://api.semanticscholar.org/CorpusID:233296808.

D. Li, Z. Liu, X. Hu, Z. Sun, B. Hu, and M. Zhang. In-context learning state vector with inner and momentum optimization. 2024a. URL https://api.semanticscholar.org/CorpusID:269187789.

Y. Li, M. E. Ildiz, D. Papailiopoulos, and S. Oymak. Transformers as algorithms: Generalization and stability in in-context learning. 2023.

Z. Li, Z. Xu, L. Han, Y. Gao, S. Wen, D. Liu, H. Wang, and D. N. Metaxas. Implicit in-context learning. *ArXiv*, abs/2405.14660, 2024b. URL https://api.semanticscholar.org/CorpusID:269983389.

L. Lin, Y. Bai, and S. Mei. Transformers as decision makers: Provable in-context reinforcement learning via supervised pretraining. *ArXiv*, abs/2310.08566, 2023. URL https://api.semanticscholar.org/CorpusID:263909278.

Z. Lin and K. Lee. Dual operating modes of in-context learning. *arXiv preprint arXiv:2402.18819*, 2024.

S. Liu, H. Ye, L. Xing, and J. Y. Zou. In-context vectors: Making in context learning more effective and controllable through latent space steering. *ArXiv*, abs/2311.06668, 2023. URL https://api.semanticscholar.org/CorpusID:265149781.

X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang. Gpt understands, too. *ArXiv*, abs/2103.10385, 2021. URL https://api.semanticscholar.org/CorpusID:232269696.

Q. Long, Y. Wu, W. Wang, and S. J. Pan. Decomposing label space, format and discrimination: Rethinking how llms respond and solve tasks via in-context learning. *ArXiv*, abs/2404.07546, 2024. URL https://api.semanticscholar.org/CorpusID:269042687.

C. Lou, Z. Jia, Z. Zheng, and K. Tu. Sparser is faster and less is more: Efficient sparse attention for long-range transformers. *arXiv preprint arXiv:2406.16747*, 2024.

G. Luo, T. Darrell, and A. Bar. Task vectors are cross-modal. 2024. URL https://api.semanticscholar.org/CorpusID:273661988.

J. Merullo, C. Eickhoff, and E. Pavlick. Language models implement simple word2vec-style vector arithmetic. *ArXiv*, abs/2305.16130, 2023a. URL https://api.semanticscholar.org/CorpusID:258887799.

J. Merullo, C. Eickhoff, and E. Pavlick. A mechanism for solving relational tasks in transformer language models. 2023b.

S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In *Conference on Empirical Methods in Natural Language Processing*, 2022.

S. Mittal, E. Elmoznino, L. Gagnon, S. Bhardwaj, D. Sridhar, and G. Lajoie. Does learning the right latent variables necessarily improve in-context learning? *ArXiv*, abs/2405.19162, 2024. URL https://api.semanticscholar.org/CorpusID:270095225.

J. Mu, X. L. Li, and N. D. Goodman. Learning to compress prompts with gist tokens. *ArXiv*, abs/2304.08467, 2023. URL https://api.semanticscholar.org/CorpusID:258179012.

S. Muller, N. Hollmann, S. P. Arango, J. Grabocka, and F. Hutter. Transformers can do bayesian inference. *ArXiv*, abs/2112.10510, 2021.

nostalgebraist. interpreting gpt: the logit lens., 2020. URL https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens.

J. Pan, T. Gao, H. Chen, and D. Chen. What in-context learning "learns" in-context: Disentangling task recognition and task learning. In *Annual Meeting of the Association for Computational Linguistics*, 2023. URL https://api.semanticscholar.org/CorpusID:258740972.

Y. Peng, C. Hao, X. Yang, J. Peng, X. Hu, and X. Geng. Learnable in-context vector for visual question answering. *ArXiv*, abs/2406.13185, 2024. URL https://api.semanticscholar.org/CorpusID:270619372.

M. Pham, K. O. Marshall, C. Hegde, and N. Cohen. Robust concept erasure using task vectors. *ArXiv*, abs/2404.03631, 2024. URL https://api.semanticscholar.org/CorpusID:268889354.

J. Phang. Investigating the effectiveness of hypertuning via gisting. *ArXiv*, abs/2402.16817, 2024. URL https://api.semanticscholar.org/CorpusID:268032098.

S. T. Piantadosi, D. C. Muller, J. S. Rule, K. Kaushik, M. I. Gorenstein, E. R. Leib, and E. Sanford. Why concepts are (probably) vectors. *Trends in Cognitive Sciences*, 28:844–856, 2024. URL https://api.semanticscholar.org/CorpusID:271721102.

J. Qiu, H. Ma, O. Levy, S. W.-t. Yih, S. Wang, and J. Tang. Blockwise self-attention for long document understanding. *arXiv preprint arXiv:1911.02972*, 2019.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

A. Raventos, M. Paul, F. Chen, and S. Ganguli. The effects of pretraining task diversity on in-context learning of ridge regression. In *ICLR 2023 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2023.

S. Ren, Q. Jia, and K. Q. Zhu. Context compression for auto-regressive transformers with sentinel tokens. In *Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://api.semanticscholar.org/CorpusID:263909553.

B. Saglam, Z. Yang, D. Kalogerias, and A. Karbasi. Learning task representations from in-context learning. In *ICML 2024 Workshop on In-Context Learning*.

S. Schug, S. Kobayashi, Y. Akram, M. Wolczyk, A. Proca, J. von Oswald, R. Pascanu, J. Sacramento, and A. Steger. Discovering modular solutions that generalize compositionally. *ArXiv*, abs/2312.15001, 2023. URL https://api.semanticscholar.org/CorpusID:266551854.

S. Sia, D. Mueller, and K. Duh. Where does in-context translation happen in large language models. *ArXiv*, abs/2403.04510, 2024. URL https://api.semanticscholar.org/CorpusID:268264275.

A. K. Singh, T. Moskovitz, F. Hill, S. C. Y. Chan, and A. M. Saxe. What needs to go right for an induction head? a mechanistic study of in-context learning circuits and their formation. *ArXiv*, abs/2404.07129, 2024. URL https://api.semanticscholar.org/CorpusID:269033179.

Z. Tao, I. Mason, S. Kulkarni, and X. Boix. Task arithmetic through the lens of one-shot federated learning. 2024. URL https://api.semanticscholar.org/CorpusID:274306177.

E. Todd, M. Li, A. S. Sharma, A. Mueller, B. C. Wallace, and D. Bau. Function vectors in large language models. *ArXiv*, abs/2310.15213, 2023. URL https://api.semanticscholar.org/CorpusID:264439657.

J. von Oswald, E. Niklasson, E. Randazzo, J. Sacramento, A. Mordvintsev, A. Zhmoginov, and M. Vladymyrov. Transformers learn in-context by gradient descent. *ArXiv*, abs/2212.07677, 2022.

J. von Oswald, E. Niklasson, M. Schlegel, S. Kobayashi, N. Zucchet, N. Scherrer, N. Miller, M. Sandler, B. A. y Arcas, M. Vladymyrov, R. Pascanu, and J. Sacramento. Uncovering mesa-optimization algorithms in transformers. *ArXiv*, abs/2309.05858, 2023. URL https://api.semanticscholar.org/CorpusID:261696852.

K. Wang, A. Variengien, A. Conmy, B. Shlegeris, and J. Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *ArXiv*, abs/2211.00593, 2022. URL https://api.semanticscholar.org/CorpusID:253244237.

L. Wang, L. Li, D. Dai, D. Chen, H. Zhou, F. Meng, J. Zhou, and X. Sun. Label words are anchors: An information flow perspective for understanding in-context learning. In *Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://api.semanticscholar.org/CorpusID:258841117.

N. Wies, Y. Levine, and A. Shashua. The learnability of in-context learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 36637–36651. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/73950f0eb4ac0925dc71ba2406893320-Paper-Conference.pdf.

S. M. Xie, A. Raghunathan, P. Liang, and T. Ma. An explanation of in-context learning as implicit bayesian inference. *ArXiv*, abs/2111.02080, 2021.

Z. Xu, Z. Liu, B. Chen, Y. Tang, J. Wang, K. Zhou, X. Hu, and A. Shrivastava. Compress, then prompt: Improving accuracy-efficiency trade-off of llm inference with transferable prompt. *ArXiv*, abs/2305.11186, 2023. URL https://api.semanticscholar.org/CorpusID:258823240.

S. Yadlowsky, L. Doshi, and N. Tripuraneni. Pretraining data mixtures enable narrow model selection capabilities in transformer models. *ArXiv*, abs/2311.00871, 2023a. URL https://api.semanticscholar.org/CorpusID:264935329.

S. Yadlowsky, L. Doshi, and N. Tripuraneni. Can transformer models generalize via in-context learning beyond pretraining data? In *NeurIPS 2023 Workshop on Distribution Shifts: New Frontiers with Foundation Models*, 2023b.

Q. Yu, J. Merullo, and E. Pavlick. Characterizing mechanisms for factual recall in language models. *ArXiv*, abs/2310.15910, 2023. URL https://api.semanticscholar.org/CorpusID:264439114.

J. Yun and J. Choo. Scaling up personalized aesthetic assessment via task vector customization. *ArXiv*, abs/2407.07176, 2024. URL https://api.semanticscholar.org/CorpusID:271088904.

Y. Zhang, F. Zhang, Z. Yang, and Z. Wang. What and how does in-context learning learn? bayesian model averaging, parameterization, and generalization. *ArXiv*, abs/2305.19420, 2023. URL https://api.semanticscholar.org/CorpusID:258987402.

H. Zhao, H. Zhao, B. Shen, A. Payani, F. Yang, and M. Du. Beyond single concept vector: Modeling concept subspace in llms with gaussian distribution. 2024. URL https://api.semanticscholar.org/CorpusID:273022717.

Y. Zhuang, C. Singh, L. Liu, J. Shang, and J. Gao. Vector-icl: In-context learning with continuous vector representations. 2024. URL https://api.semanticscholar.org/CorpusID:273227918.

Organization of the Appendix

A Related Works

Understanding In-Context Learning Behavior in the Pre-trained Large Language Model. Since the launch of GPT-3 [Brown et al., 2020] and the observation that, during inference, the model can leverage few-shot examples to improve performance, researchers have been keenly interested in understanding the mechanisms of this in-context learning behavior. Several studies have explored the significance of labeling in context demonstrations [Min et al., 2022, Kim et al., 2022, Kossen et al., 2023, Long et al., 2024], while others have examined the phenomenon through circuit mechanisms [Elhage et al., 2021, Wang et al., 2022, Singh et al., 2024, Hanna et al., 2024, Wang et al., 2023]. Researchers have also attempted to decode the logits to reveal how transformers refine their output during the forwarding process [nostalgebraist, 2020, Dar et al., 2022, Geva et al., 2022b]. Research by Dai et al. [2022], Geva et al. [2022a], and Merullo et al. [2023b] links in-context learning behavior to implicit internal weight and activation updates within the model. To better understand "learning" in-context, several works [Pan et al., 2023, Sia et al., 2024] have studied its two distinct phases–task recognition and task learning–using controlled experiments designed to disentangle these phases. These studies focus on pre-trained large language models without access to the pre-training procedure. The literature on embedding editing and task vector will be discussed in detail later.

Understanding In-Context Learning Behavior in the Controlled Small-Scale Setting on Data-Fitting Problems. Researchers have also examined in-context learning behavior in small-scale models, where models are trained from scratch to perform in-context learning tasks. Garg et al. [2022], Akyürek et al. [2022], von Oswald et al. [2022, 2023] investigated the ability of transformers to learn regression problems, interpreting the model as performing a single step of gradient descent. This line of research was extended further in different pre-training mixtures of tasks [Raventos et al., 2023, Yadlowsky et al., 2023a], the ability to generalize to unseen tasks Yadlowsky et al. [2023b], explanations via the Bayesian optimal estimator [Zhang et al., 2023, Bai et al., 2023, Muller et al., 2021], the viewpoint of generalization error [Li et al., 2023], and the perspectives of task retrieval and task learning [Lin and Lee, 2024, Wies et al., 2023].

Beyond regression, several studies have explored in-context learning abilities in other domains, such as reinforcement learning [Lee et al., 2023, Lin et al., 2023], discrete function learning [Bhattamishra et al., 2023], factorial hidden Markov chains [Xie et al., 2021], and deterministic finite automata (DFAs) [Akyürek et al., 2024]. Additionally, Chan et al. [2022a,b] investigated the behavior of in-context learning and in-weight learning on the Omniglot datasets.

Task Representation The notion of a "task vector" was first introduced by Ilharco et al. [2022]. Subsequently, Hendel et al. [2023], Merullo et al. [2023a], Yu et al. [2023], Liu et al. [2023], Saglam et al. and Li et al. [2024b] demonstrated that a single vector in the model's activation space can encode learned functions in a pretrained model. Specifically, Liu et al. [2023] showed that, when presented with an in-context learning demonstration, the activation at each layer points to a subspace encoding the task information. Additionally, Todd et al. [2023] identified the "function vector," another form of task vector, by averaging the causal attention heads, which can also guide the pre-trained language model's performance towards desired tasks. Task vectors have also been identified under different modality [Luo et al., 2024, Hojel et al., 2024, Peng et al., 2024], and from different perspective, such as cognitive science [Piantadosi et al., 2024], and through theory perspective [Tao et al., 2024].

A recent study by Mittal et al. [2024] investigates a setting similar to ours, focusing on whether learning appropriate latent representations—achieved through a bottleneck architecture—can enhance robustness in in-context learning. In contrast, our approach facilitates the natural emergence of latent representations within the model's architecture, aligning with the phenomenon of task vector emergence. Building on this idea, Kobayashi et al. [2024] further examined bottleneck architectures, demonstrating that models incorporating bottlenecks exhibit improved compositional generalization. Elmoznino et al. [2024] use the bottleneck architecture to examine the prequential ICL performance. Han et al. [2024] investigate the training dynamics of task vector formation, demonstrating that as task vectors emerge, the model simultaneously develops conditional decoding algorithms, thereby enhancing its in-context learning (ICL) performance.

Task Vector Applications Li et al. [2024a] utilized this task vector to enhance test-time adaptation, while Pham et al. [2024] employed it to erase malicious concepts during pre-training. Beyond activations, task information can also be encoded in a task token Bai et al. [2024], or a pause token

can be used to gain extra computation time Goyal et al. [2023]. Zhao et al. [2024] propose to present each task concept with a Gaussian distribution. Following the notion of "task vector" in the weight space, several works utilize the weight space difference for chatbot on new language [Huang et al., 2023, Kang et al., 2024], emotion transfer [Kalyan et al., 2024], aesthetic assessment [Yun and Choo, 2024], and soft prompt initialization [Belanec et al., 2024].

Soft Prompt and Efficient Adaptation of Transformers. In this work, we primarily follow the definition of a task vector as described by Hendel et al. [2023]. This task vector effectively compresses the information in the context into a single vector, which can be injected into the model to perform zero-shot inference. The notion of a task vector is also related to the concept of a soft prompt [Lester et al., 2021, He et al., 2022, Liu et al., 2021, Xu et al., 2023, Kang et al., 2024], where the context information is similarly compressed. Beyond learning soft prompts that encode instructions, researchers have also explored context compression by modifying the attention mask to ensure the summarization of context at a specific token Ren et al. [2023], Mu et al. [2023], Phang [2024]. In the format of ICL, the researcher also study the continuous representation in context [Zhuang et al., 2024], as a reminiscent of the soft encoding.

Although we propose a training algorithm to encourage the formation of task vectors, our primary goal is not to compare our method of encoding task vectors with these approaches in soft prompt learning. Instead, we aim to understand the benefits of incorporating task vectors into in-context learning.

B Supplements on the Trained-from-Scratch Model

B.1 Effects of Input Formats

In this section, we examine different input formats for in-context learning prompts and their effects on the emergence of task encoding. The various prompt formats are detailed in Table 1. For clarity, we refer to the token positions where task vector presence is examined as the *task token locations*.

We focus on the case where the problem dimension is $d = 6$ and the model depth is $L = 3$, as it exhibits the most pronounced task encoding, as shown in Figure 3. In Figure 9, we evaluate various task vector extraction methods (detailed in Appendix B.2), with their respective task vector prompting performance shown as line with triangular markers. Notably, only the model with the input format `(*xy)` demonstrates task encoding that surpasses random baselines. By default, we use the format `(*xy)` throughout the main paper unless specified otherwise.

In the pre-trained LLM [Hendel et al., 2023], the input format uses `x->y`. There is a clear difference between the pre-trained LLM and the trained-from-scratch transformer in how task information is encoded. In the pre-trained LLM, task information is primarily stored in the "maps-to" token (`->`). Conversely, in the trained-from-scratch transformer, task information is stored in the y token (i.e., the "label" token), but only when x and y alternate closely in the input format. We further analyze the attention maps and PCA of the activations when input format is `x->y`, following the same setup as in Figure 2, and present the results in Figure **??**. Across different layers, the attention maps do not exhibit meaningful task information encoded in the patterns. Consequently, we showcase only one example attention map to illustrate the absence of distinct task encoding in this setup. Additionally, the PCA of the activations reveals that the encoded information is nearly identical across tasks.

We hypothesize that this discrepancy arises because pre-trained LLMs learn the semantic meaning of the "maps-to" symbol (`->`) from their extensive pre-training corpus, enabling them to use `->` as an anchor for task summarization. Additionally, this delimiter `->` is needed to separate the x and y entries. In contrast, for the trained-from-scratch transformer, the `->` token functions more like a `<pause>` token [Goyal et al., 2023], providing additional computational resources rather than semantic significance. Moreover, the training procedure ensures that x and y tokens occupy fixed positions, removing the need for a delimiter to separate them.

Furthermore, Wang et al. [2023] show that label words themselves can serve as anchors for aggregating task information in context. This observation aligns with our finding that task information is encoded in the y tokens. While this result is not explicitly framed within the task vector framework, it highlights the critical role of label tokens in task encoding.

Table 1: *Different prompt formats examined.* We analyze the input format based on the settings described in Garg et al. [2022] and Hendel et al. [2023]. Here, z represents a special token, P_{query} denotes the test query format, and P_k refers to the k-shot context. We examine the activations at Λ_f, referred to as the task token location.

	P_{query}	k-shot examples P_k	Task Token Location Λ_f
`(*xy)` Garg et al. [2022]	$[z, x_{\text{test}}]$	$[z, x_1, f(x_1), \cdots, x_k, f(x_k), x_{k+1}]$	P_k`[0::2]`: $\{f(x_i)\}$ and z
`(x->y)` Hendel et al. [2023]	$[x_{\text{test}}, z]$	$[x_1, z, f(x_1), \cdots, x_k, z, f(x_k), x_{k+1}, z]$	P_k`[1::3]`: z
`(*x->y)` Hendel et al. [2023]	$[z, x_{\text{test}}, z]$	$[z, x_1, z, f(x_1), \cdots, x_k, z, f(x_k), x_{k+1}, z]$	P_k`[0::3]`: $\{f(x_i)\}$ and z

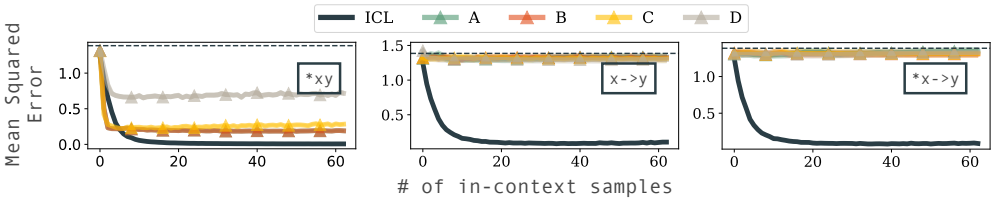

Figure 9: *In-context learning (ICL) and task vector prompting (TVP) performance across input formats and task vector extraction methods.* This figure compares task vector prompting performance across four task vector extraction methods (detailed in Appendix B.2) and three input formats: `(*xy)`, `(x->y)`, and `(*x->y)`. Method (A) is the default task vector extraction method, and `(*xy)` is the default input format used throughout the main paper. As shown, for the `(*xy)` input format, noticeable task encoding is observed, with performance exceeding random predictions, regardless of the extraction method. In contrast, for the `(x->y)` and `(*x->y)` input formats, no task encoding can be reliably extracted by any method, resulting in task vector prompting performance that is nearly random.

B.2 Various Task Vector Locating Methods

In this section, we analyze different methods for extracting task vectors. The main question we address is: *If task encoding is detectable by one task vector extraction method, will it also be detectable by another method?* Our focus is not on the quality of the extracted task vector but rather on whether the method can detect task encoding that is non-random. Below, we present task vector extraction methods inspired by Hendel et al. [2023], Liu et al. [2023], Li et al. [2024b], which are representative approaches in the task vector literature.

Following the definition in Section 2, we study the following task vector extractor methods:

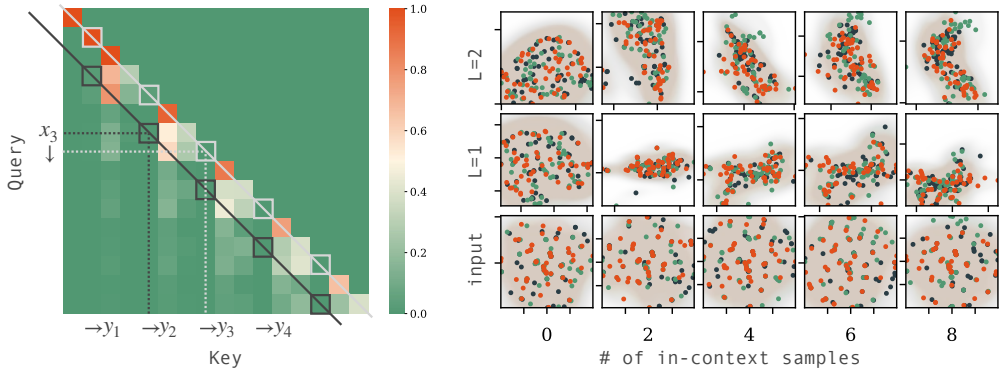

Figure 10: *Attention map and PCA visualization of activations for three linear functions with input format `(x->y)`.* Following the same setup as in Figure 2, we plot the attention map and PCA visualization of the `->` token's activations. (*Left*) The attention map at the 2nd layer highlights interactions between task tokens (white boxes) and the attention from x_i tokens to task tokens (black boxes). Unlike the format `(*xy)`, this format does not exhibit a meaningful attention pattern where x_i tokens attend to previous task tokens, or task tokens attend to prior task tokens. (*Right*) PCA visualizations show that the activations of the `->` token encode nearly identical information across different context lengths and different tasks, suggesting the absence of distinct task encoding dynamics in this input format.

(A) Task Token's Output Embedding Hendel et al. [2023] have observed that in pre-trained language models, the l-th layer output embedding of the k-th task token, denoted as h_k^l, encodes the task vector. We follow this setup and define the task vector extractor g as follows: Let $\boldsymbol{\tau} = h_k^l$, then

$$M(P_{\text{query}}; h_k^l) \coloneqq \texttt{replace task token's l-th layer embedding with } h_k^l.$$

This notion of the task vector indicates the independence of the encoded task information from the demonstrated prompt.

(B) Task Token's Output Embedding Difference Liu et al. [2023] pointed out that when inputting two sequences x and y, the difference in the hidden space corresponds to the mapping from x to y in the pre-trained model. Though their setting is slightly different from the in-context learning format, we leverage this idea and propose the following way to extract the task embedding: Following the definition of h_k^l from above, and let $\tilde{h}_k^l$ be the hidden state of the model when inputting the test prompt P_{query}, then $\boldsymbol{\tau} = h_k^l - \tilde{h}_k^l = \Delta h_k^l$, indicating the difference between the k-th context embedding and an uninformative context, then

$$M(P_{\text{query}}; \Delta h_k^l) \coloneqq \texttt{add } \Delta h_k^l \texttt{ to task token's l-th layer embedding.}$$

This notion of the task vector loosens the constraint of absolute independence of the task from the demonstrated prompt.

(C) Principal Direction of Task Token's Output Embedding Difference Liu et al. [2023] also observed that taking the principal direction along the difference of hidden states aligns more closely with the task. Motivated by their observation and setting, we propose the following task vector extractor: Following the definition of Δh_k^l from above, we set $\boldsymbol{\tau} = \texttt{PCA}(\Delta h_k^l)$, then

$$M(P_{\text{query}}; \texttt{PCA}(\Delta h_k^l)) \coloneqq \texttt{add PCA}(\Delta h_k^l) \texttt{ to task token's l-th layer embedding.}$$

We also re-scale the updated hidden states to preserve the model's original capabilities. This approach differs slightly from the original setup in Liu et al. [2023], where PCA is applied to the concatenated hidden states across all layers. Their method is specifically designed for alignment tasks in natural language settings, where the context x_i and y_i are input separately. In our case, as the model is trained in an in-context learning (ICL) format, we adhere to the input prompt structure where x_i and y_i are provided alternately in one prompt. We also attempted to follow their setup by applying PCA to the concatenated hidden states across all layers; however, this approach resulted in near-random task vector prompting performance.

(D) Linear Combination of Task Token's Output Embedding to the Query Output Embedding Li et al. [2024b] propose to learn the coefficient of the linear combination between the task token's output embedding to the query output embedding. Motivated by this, we let $\boldsymbol{\tau} = \{h_k^l\}_{l \in [L]}$, then

$$M(P_{\text{query}}; \{h_k^l\}_{l \in [L]}) \coloneqq \texttt{replace task token's l-th layer embedding with } \alpha_l^h h_k^l + \beta_l^h h_t^l \quad \forall l \in [L].$$

where h_t^l is the output embedding at l-th layer for the test query, and α_l^h, and β_l^h are learnable parameters. To learn these coefficients, we use a constant learning rate of 0.01, the AdamW optimizer, and train for a total of 100 epochs. Following the initialization setup in Li et al. [2024b], the coefficients are initialized as $\alpha_l^h = 0.1$ and $\beta_l^h = 1$.

B.2.1 Task Vector Extractor Performance

We evaluate the performance of various task vector extraction methods across different input formats, as described in B.1, and present the results in Figure 9. When using the `*xy` input format, the task vector extraction methods demonstrate noticeable task encoding. However, for the other input formats, no informative task encoding is detected, regardless of the extraction method used.

It is worth noting that the learnable task vector extraction method (D) performs slightly worse than the other three methods, possibly due to its initialization. We did not focus on further improving this method, as the primary goal of this section is to demonstrate the detectability of task encoding rather than to optimize its quality.

B.3 Effects of Model Capacity

In this section, we examine the model with same depth, but different embedding size, to investigate the effect of model capacity. Specifically, for the three tasks mentioned in Section 3.1, we study the trained-from-scratch model's performance on model depth 4, 6 or 8, with embedding size 64 (with notation "S") or 256 (with notation "L"), and present the result in Figure 11.

As illustrated in the figure, increasing the embedding size allows the model to discover shortcut solutions for regression tasks. Specifically, the model approximates the least squares solution in a single forward pass, eliminating the need to store intermediate task encodings. Conversely, for the discrete token offset task, the model operates on tokens represented in a learned embedding space. To solve $f(x) = a \times x + b$, the model must map one token to another using the task-specific parameters a and b. We hypothesize that the nature of token embeddings makes it less straightforward for the model to learn a shortcut solution for this task. As a result, the model is compelled to store the task information explicitly. Consequently, increasing the model capacity provides more space for storing and organizing task encodings, leading to clearer task representations as capacity grows.

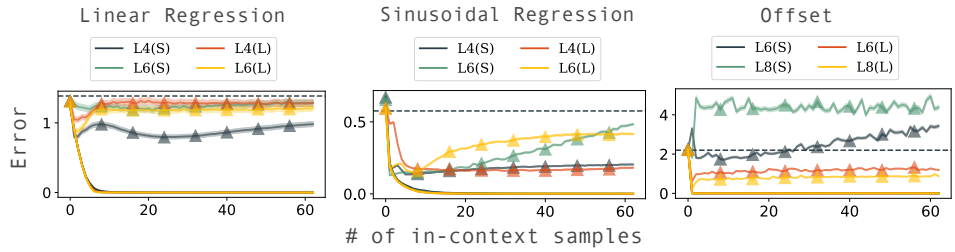

Figure 11: *Effects of model embedding dimension on task vector prompting performance.* This figure shows the task vector prompting performance of models with different embedding sizes (64 for notation "S" and 256 for notation "L"). For the regression tasks (Linear and Sinusoidal), larger embedding sizes make task encoding less prominent, likely because the increased capacity allows the model to find "shortcut" solutions, such as approximating the least-squares solution directly, without the need to store intermediate task information. In contrast, for the discrete token offset task, a larger embedding size enables the model to store task information in a more separable and structured manner.

B.4 Effects of Sparse Attention

Sparse attention [Lou et al., 2024] has been proposed to improve transformer efficiency with minimal performance degradation. In in-context learning tasks (detailed in Section 3.1), a model could potentially solve these tasks using sparse attention: For each query, the model only needs to attend to itself and the previously stored task information. Specifically, each x_i token needs to attend to two tokens (x_i and y_{i-1}), while each y_i token needs to summarize the task information by attending to three tokens (y_i, y_{i-1}, and x_i).

In this section, we investigate the impact of explicitly enforcing sparse attention constraints on in-context learning and task vector prompting performance. Using the setup described in Section 3.1, we apply a sliding window strategy [Qiu et al., 2019, Beltagy et al., 2020] combined with causal attention, with window sizes s of 3, 5, and 7. We evaluate the transformer with various depth on the linear regression task with $d = 6$ (where task vectors emerge) and $d = 9$ (where task vectors are nearly absent), and present the results in Figure 12.

As shown in the figure, models trained with sparse attention demonstrate better task encoding compared to those with full-window causal attention. For instance, when $d = 6$ and $L = 5$, task encoding in the full-window model fades after the first few in-context examples. In contrast, models trained with sliding window attention (window sizes 3, 5, and 7) maintain stable task encoding across varying context lengths.

However, while the sliding window improves task vector emergence, it slightly degrades in-context learning performance because each query has access to fewer tokens during training. Moreover, task encoding stability is not always guaranteed: larger window sizes improve in-context learning performance but degrade task vector prompting performance.

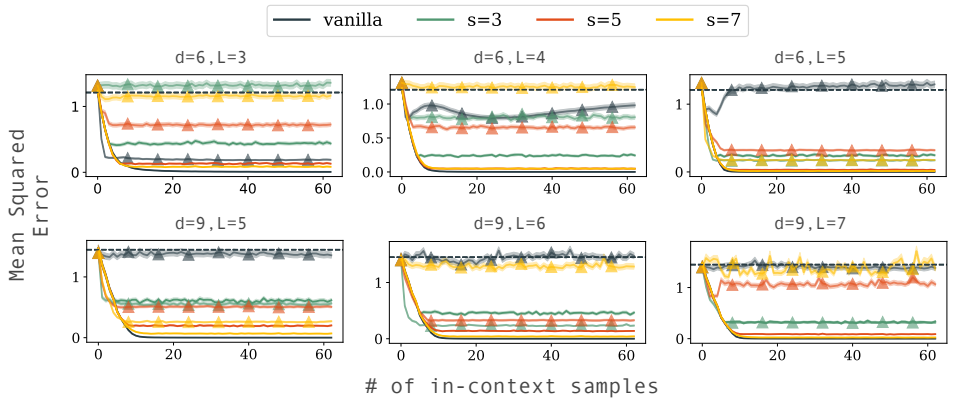

Figure 12: *Task vector prompting performance for models trained with sliding window sparse attention.* We compare models trained with full-window causal attention (vanilla) and sparse attention using sliding window sizes of 3, 5, and 7. Solid lines represent the in-context learning performance, while lines with extra triangular marker indicate the task vector prompting performance. As shown, sparse attention enhances task vector emergence but leads to a slight degradation in in-context learning performance.

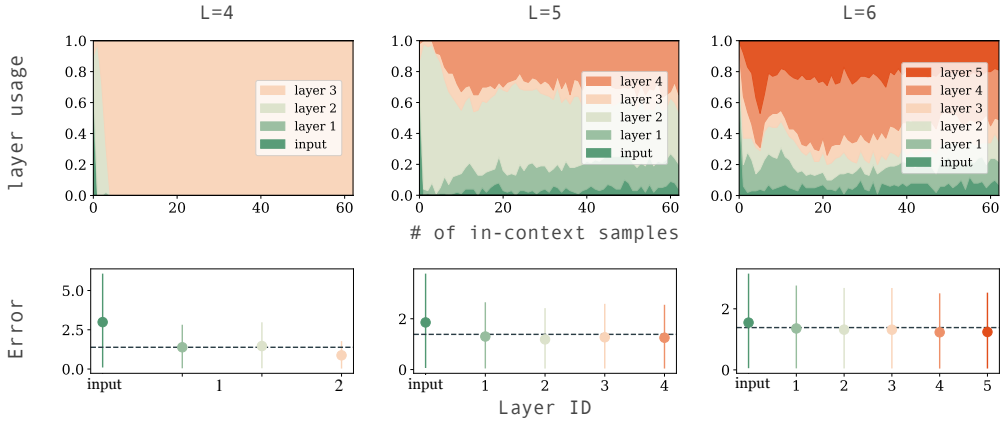

Figure 13: *Layer selection distribution (upper row) and averaged task vector prompting (TVP) performance (lower row) for task vector emergence in Linear Regression with problem dimension $d = 6$ and model depths $L = 4$ (left), $L = 5$ (middle), and $L = 6$ (right).* **Upper row:** The frequency of each layer being selected as the task vector location across varying numbers of in-context examples is shown. As model depth increases, task information becomes less concentrated in any single layer, resulting in a more distributed selection of task vector locations. **Lower row:** The averaged TVP performance for each layer, measured across various context lengths, is presented. The dashed line indicates random-guess performance, representing a scenario where no task information is inferred. These results demonstrate that for deeper models (especially $L = 5, 6$), no single layer consistently encodes significant task information, and task vector performance approaches random, highlighting the distributed nature of task encoding in deeper architectures.

B.5 Supplements on Task Vector Layer Localization

To further support the argument that task vectors in the linear regression task emerge only in the penultimate layer, we extend the analysis from Figure 4 to include the linear regression task with $d = 6$ and model depths $L = 4, 5, 6$. As shown in the figure, for deeper models, particularly with $L = 5$ and $L = 6$, no single layer consistently encodes task information, and the task vector prompting performance appears random. From this analysis, we conclude that in the linear regression task, and under the experimental setup we examined, task vectors either emerge in the penultimate layer when the model is shallow or fail to emerge entirely when the model is deep.

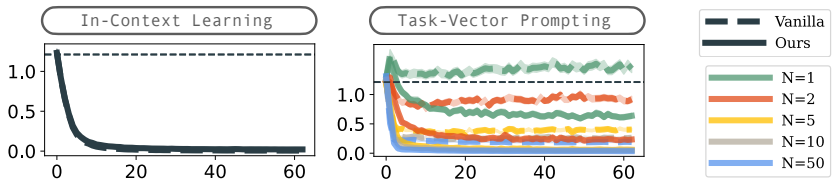

Figure 14: *Impact of the Number of Demonstrated Prompts (N) on Task Vector Prompting Performance.* Increasing N improves task vector prompting performance, as larger N values yield more concentrated task vectors. Across all N settings, model trained with TVP-loss ("Ours") consistently outperforms the vanilla model, demonstrating better task vector encoding. The dashed line represents random-guess performance.

B.6 Effects of Number of Demonstrated Prompts N

Throughout the paper, we use $N = 50$ by default when evaluating task vector prompting performance. Intuitively, larger N values yield more concentrated and purified task vectors. To illustrate the effect of N, we evaluate task vector prompting performance for $N = 1, 2, 5, 10, 50$, following the setup in Section 3.1, and present the results in Figure 14. In this experiment, the transformer is trained on a linear regression task with a problem dimension of $d = 6$ and transformer depth $L = 3$. For the model trained with TVP-loss, the task vector is explicitly formed at the 2nd layer.

As shown in the figure, when $N = 1$, the vanilla model yields random task vector prompting performance. With increasing N, task vector performance improves steadily. Notably, the model trained with TVP-loss (denoted as "Ours" in the figure) demonstrates better task vector encoding compared to the vanilla model across all N values. Because the linear functions in regression tasks have weights that exist in a continuous space and can be very close to each other, separating tasks with only a single demonstration is challenging. As a result, it requires $N = 5$ to achieve task vector prompting performance comparable to its in-context learning performance. For the vanilla model, however, even with $N = 50$, task vector prompting performance remains slightly worse than in-context learning performance.

C Supplements on Training with TVP-Loss

C.1 Full Results for Section 4.2

In Section 4.2, we present the in-context learning and task vector prompting performance at the 63rd context position for models trained with and without the TVP-loss, along with the in-context learning performance on out-of-distribution prompts. Figure 15 extends these results by showing the full performance across varying context lengths. This analysis complements the main results, demonstrating that the observations at the 63rd position are not unique but instead hold consistently across different context lengths.

C.2 Performance on Additional OOD Tasks

We evaluate the model's performance on additional out-of-distribution (OOD) prompts for the regression tasks, specifically: (a) logistic regression, (b) in-context learning with an outlier context presented with a probability of 0.1, where the outlier is defined as $\boldsymbol{x} = 1$ and $y = 1$. (c) in-context learning with noisy labels (noise level 0.3), (d) scaled weights ($\times 3$).

For the linear regression training task, models with task vectors formed at the 7th layer (i.e., the last layer) perform equal to or slightly better than the vanilla-trained model, except when the weights and inputs are scaled. This suggests that models trained with TVP-loss exhibit reduced robustness when the input range is significantly altered. For the sinusoidal regression task, models with task vectors formed at the 3rd layer exhibit equal or improved OOD performance.

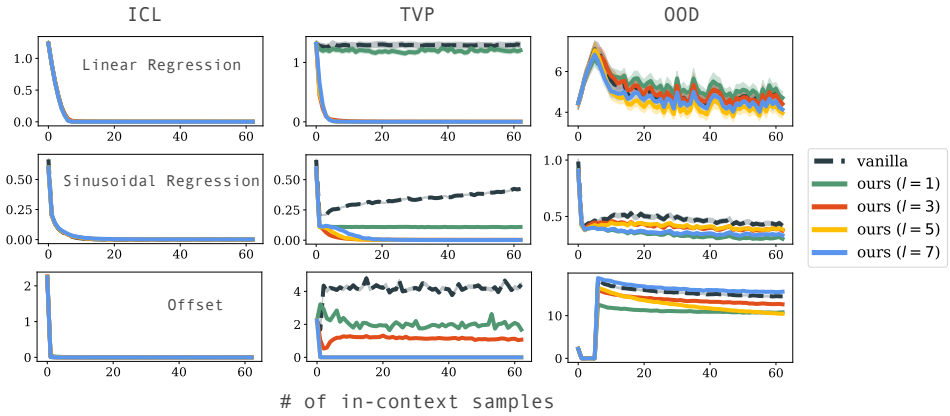

Figure 15: *Model Performance with Vanilla and TVP-Loss Training.* We compare the in-context learning (ICL), task vector prompting (TVP), and out-of-distribution (OOD) performance of models trained with vanilla training (black) and those trained with our TVP-loss, where task vectors are formed at different layers ($l = 1, 3, 5, 7$). Full results are shown across varying numbers of in-context examples for three tasks: linear regression (top row), sinusoidal regression (middle row), and discrete token offset task (bottom row).

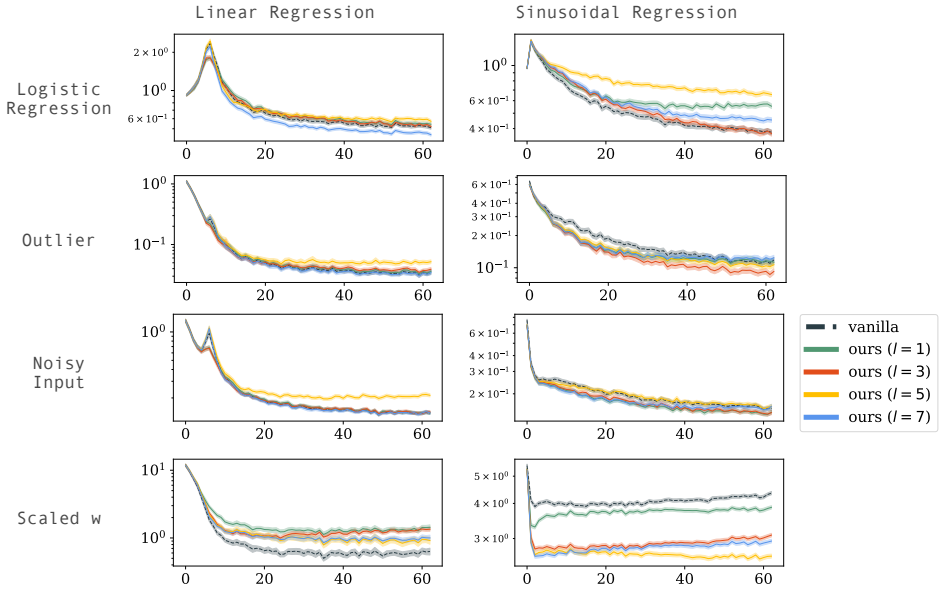

Figure 16: *In-Context Learning Performance on OOD Prompts for Models Trained with Regression Tasks.* We compare the in-context learning performance of the vanilla-trained model (labeled as *vanilla*) and models trained with auxiliary loss to form task vectors at different layers l (labeled as *ours* (l)). We evaluate the models on OOD prompts, including: (a) logistic regression, (b) oulier in context, (c) noisy input labels with a noise level of 0.3, (d) scaled weights ($\times 3$).

C.3 Indentification of task vector in GINC dataset

In GINC, there are no explicit input-output pairs; instead, the entire trajectory performs one task (hidden markov model). Therefore, we encode the task vector into the inserted token "-" in the trajectory. We provide an illustration of our training algorithm in Figure. 17.

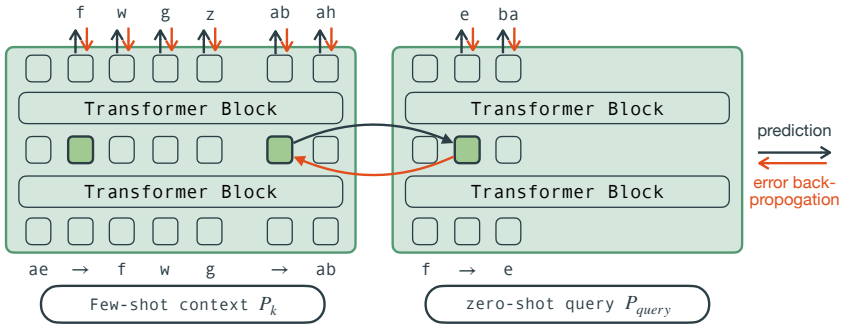

Figure 17: *Demonstration of our training algorithm in GINC dataset.* In contrast to the Meta-ICL format described in Figure 5, in GINC dataset, there are no explicit input-output pairs. Instead, the entire trajectory performs one single task defined by a hidden markov model. Therefore, we randomly insert several $\rightarrow$" tokens into the trajectory in the few-shot mode, while inserting one $\rightarrow$" token into the trajectory in the zero-shot mode. The task vector at the l-th layer is then injected into the embedding in the zero-shot mode.