Distributional Alignment as a Principle for Designing Task Vectors in In-Context Learning

Jihoon Kwon [*1] **Jiwon Choi** [*2] **Jy-yong Sohn** [2]

Abstract

In-context learning (ICL) allows large language models (LLMs) to adapt to new tasks through demonstrations, yet it suffers from escalating inference costs as context length increases. While task vectors (TVs) offer a promising alternative by compressing demonstrations into compact hidden-state representations, the field lacks principled criteria for evaluating the quality of TV, hindering the development of more effective methods. In this paper, we posit that probabilistic alignment with ICL is a key desideratum for effective TVs. To quantify this, we introduce d_{NTP}, a novel metric that measures the discrepancy in next-token probabilities between TV-based and ICL-based inference. Our empirical analysis reveals that d_{NTP} serves as a reliable performance proxy, exhibiting a strong negative correlation with downstream accuracy. Motivated by this, we develop Linear Task Vector (LTV), a method designed to minimize d_{NTP} via a closed-form linear mapping that estimates demonstration effects through regression. Across eight classification benchmarks, LTV consistently outperforms existing training-free baselines – improving average accuracy by up to 9.2% – while achieving the lowest discrepancy on our proposed metric and significantly reducing inference latency.

1. Introduction

In-Context Learning (ICL) has emerged as a powerful paradigm for adapting large language models (LLMs) to new tasks by simply prepending labeled demonstrations before the query (Brown et al., 2020; Wei et al., 2022). ICL has been shown to achieve impressive performance gain across various tasks without requiring any model parameter updates, with performance typically improving as more demonstrations are provided (Agarwal et al., 2024; Dong et al., 2024). However, this improvement comes at a cost: longer demonstrations either increase inference-time computation as the input length grows, or incur memory overhead when caching the activations (Mu et al., 2023; Li et al., 2025c; Gao et al., 2025; Golchin et al., 2025). These computational and memory limitations hinder the practical utility of ICL under resource constraints.

To address these limitations, recent work has proposed *task vectors in ICL* as a training-free approach that enables task adaptation without directly using demonstrations at inference time (Hendel et al., 2023; Todd et al., 2024; Liu et al., 2024; Li et al., 2024). In this approach, a task vector (TV) refers to a condensed vector extracted from the internal activations of an LLM when performing ICL, encapsulating the task information that the LLM implicitly infers from demonstrations (Han et al., 2025; Yang et al., 2025b; Dong et al., 2025). By applying this vector to zero-shot inference, where no demonstrations are provided, the model achieves substantial performance gains on the task (Hendel et al., 2023). While various TV extraction methods have been proposed in the past years, the downstream task performance remains the only established criterion for comparing them, offering limited insight into *why* one method outperforms another and *how* to improve the existing extraction methods.

In this paper, we posit that probabilistic alignment with ICL is a key desideratum for effective TV extraction methods. This perspective is motivated by the following idea: since the role of a task vector is to condense the effect of demonstrations, TV-based inference should naturally produce a predictive distribution that closely aligns with that of ICL. From this perspective, we make the following contributions:

- We propose d_{NTP}, a metric that quantifies the quality of TV methods, by measuring the *discrepancy* between the predictive distribution under TV-based inference and that under ICL-based inference, in terms of *next-token probability (NTP)*. We empirically show that d_{NTP} has a strong negative correlation with the downstream performance, serving as a reliable indicator of the quality of TV.
- We develop the Linear Task Vector (*LTV*) method, which is designed to reduce d_{NTP}. Specifically, *LTV* employs a

[1]College of Engineering, Seoul National University, Seoul, Republic of Korea [2]College of Commerce and Economics, Yonsei University, Seoul, Republic of Korea. Correspondence to: Jy-yong Sohn <jysohn1108@yonsei.ac.kr>.

Preprint. March 7, 2026.

linear mapping that estimates the effect of demonstrations, and uses the closed-form solution of a regression problem to extract task vectors.

- Through experiments across eight classification benchmarks and five LLMs, we demonstrate that *LTV* consistently outperforms existing training-free TV extraction methods by up to 9.2% on average, while achieving the lowest d_{NTP} and inference latency.

2. Related Work

Task Vectors. A pioneering work Ilharco et al. (2023) introduces the concept of task vectors, defined as the difference in parameter space between a pre-trained model and the model fine-tuned for a specific task. The core idea is that shifting the weights of a model in the task vector direction improves performance on that task without additional fine-tuning (Ortiz-Jimenez et al., 2023; Zhang et al., 2024; Li et al., 2025a). Recent studies have shown that task vectors can also be extracted from other representation spaces, such as activation spaces (Hendel et al., 2023; Yang et al., 2025b) or soft prompt spaces (Belanec et al., 2025).

In-Context Learning (ICL). ICL enables LLMs to adapt to new tasks by simply prepending query-label pairs as demonstrations to the model input (Brown et al., 2020). Despite its simplicity, ICL yields significant performance gains, making it a compelling paradigm for task adaptation without parameter updates (Dong et al., 2024). The success of ICL has motivated various theoretical interpretations (Zhou et al., 2024), including perspectives based on gradient descent (Von Oswald et al., 2023; Ahn et al., 2023), mechanistic interpretability (Olsson et al., 2022; Yin & Steinhardt, 2025), and algorithmic learning (Akyürek et al., 2022; Li et al., 2023). Among these, a notable line of research interprets ICL as implicit Bayesian inference (Xie et al., 2021; Panwar et al., 2023; Zhang et al., 2023): as the model processes demonstrations, it implicitly infers a *latent task concept* from demonstrations and conditions its predictions on the resulting posterior. This view provides the theoretical foundation for task vectors in ICL, approaches that seek to explicitly extract this latent concept as a vector and apply it during zero-shot inference (Mittal et al., 2025).

Task Vectors in ICL. A key limitation of ICL is that it incurs substantial inference-time computation and memory overhead as the input length increases (Li et al., 2025c; Agarwal et al., 2024). To reduce these overheads, recent works aim to internalize the task adaptation induced by ICL, by adjusting the model parameters or activations so that the effects of demonstrations are encoded within the model itself. One line of work achieves this through few-shot parameter-efficient fine-tuning (PEFT) (Li et al., 2025c; Jukić & Šnajder, 2024; Gao et al., 2025; Li et al., 2025b), but these methods require additional training. In contrast, task vectors in ICL offer a training-free alternative that enables task adaptation without increasing the input length.

Numerous methods have been proposed for extracting task vectors in ICL, demonstrating performance gains over zero-shot inference (Hendel et al., 2023; Todd et al., 2024; Liu et al., 2024; Li et al., 2024; 2025c). As the model activations span multiple modules, existing methods vary widely in both where task vectors are extracted – such as from attention modules (Todd et al., 2024; Li et al., 2024; 2025c) or decoder layers (Hendel et al., 2023; Liu et al., 2024) – and how they are extracted, even within the same module. To the best of our knowledge, no existing metric directly measures the quality of task vectors in a way that serves as a guideline for principled extraction.

Several prior studies have proposed metrics that measure the separability between task vectors corresponding to different tasks (Han et al., 2025; Jiang et al., 2025). However, these metrics describe observed phenomena in ICL rather than guide how to improve extraction. Other lines of work aim to improve task vector performance through different means. ATV (Kang et al., 2025) relies on a small auxiliary LLM to generate query-dependent task vectors, while ELICIT (Wang et al., 2025) pre-constructs a pool of task vectors and retrieves the most suitable one at inference time. In contrast, we focus on improving how task vectors are extracted from ICL. Specifically, we propose an interpretable and optimizable metric for evaluating task vector quality, and introduce a method that explicitly reduces this metric.

3. Backgrounds

In this section, we first review relevant concepts and notations used in our paper. Sec. 3.1 describes how LLMs predict the next token, Sec. 3.2 defines our target classification task, and Sec. 3.3 introduces three inference modes for LLMs – zero-shot, ICL, and using task vectors.

3.1. Model: Large Language Models (LLMs)

We consider pre-trained auto-regressive LLMs which predict the next token u given an input prompt p, a sequence of tokens. The model consists of three components:

- the embedding layer that converts each token in the prompt p into an embedding vector,
- the transformer (Vaswani et al., 2017) (TF) decoder consisting of L layers, which transforms the sequence of embedding vectors into a sequence of hidden states,
- the language modeling (LM) head that predicts the probability of the next token u based on the output of TF.

Let $[\boldsymbol{h}_1(p), \boldsymbol{h}_2(p), \ldots, \boldsymbol{h}_l(p)]$ denote the hidden states at the final layer of the TF decoder, where l is the sequence length

and $\boldsymbol{h}_l(p) \in \mathbb{R}^d$ is a d-dimensional vector. The LM head predicts the next token $u \in \mathcal{U}$ by applying a linear projection to the last hidden state $\boldsymbol{h}_l(p)$, where $\mathcal{U} = \{1, 2, \ldots, N_{\mathcal{U}}\}$ is the vocabulary set; each token is represented by its index. To be specific, the probability of the next token is computed as:

$$P(u \mid p) = \sigma(\boldsymbol{W}_{\text{lm}} \boldsymbol{h}_l(p))[u], \quad u \in \mathcal{U}, \tag{1}$$

where $\boldsymbol{W}_{\text{lm}} \in \mathbb{R}^{N_{\mathcal{U}} \times d}$ denotes the weight matrix of the LM head, and $\sigma(\cdot)$ denotes the softmax function.

For notational simplicity, we hereafter write $\boldsymbol{h}(p)$ for $\boldsymbol{h}_l(p)$, as we only use the hidden state of the last token. We also write $\text{TF}(p)$ to denote the hidden state $\boldsymbol{h}(p)$ obtained by embedding the prompt p and passing it through TF.

3.2. Task: Classification

While large language models (LLMs) can be applied to a wide range of downstream tasks, prior work on task vectors in ICL (Hendel et al., 2023; Li et al., 2025c; Saglam et al., 2025) has primarily focused on classification settings. Following this line of work, we also restrict our attention to classification tasks.

We define a classification task by a distribution $\mathcal{D}$ over query–label pairs (x, y), where the query x is a text sequence and the label y belongs to a task-specific label set $\mathcal{C} \subseteq \mathcal{V}$ which contains $|\mathcal{C}| = K$ classes. Given a query x, the goal is to predict its corresponding label y. We consider the next-token distribution *restricted* to the label set $\mathcal{C}$:

$$P(c \mid p; \mathcal{C}) = \frac{P(c \mid p)}{\sum_{c' \in \mathcal{C}} P(c' \mid p)}, \quad c \in \mathcal{C}. \tag{2}$$

For notational simplicity, we hereafter write $P(c \mid p)$ to denote this label-restricted distribution. In greedy decoding, the predicted label $\hat{y}$ is determined by selecting the class with the highest probability:

$$\hat{y} = \text{argmax}_{c \in \mathcal{C}} P(c \mid p). \tag{3}$$

3.3. Methods: Inference Modes for LLMs

We introduce three inference modes for LLMs: zero-shot inference in Sec. 3.3.1, in-context learning (ICL) in Sec. 3.3.2, and task vectors in Sec. 3.3.3.

3.3.1. Zero-shot Inference Mode

In the zero-shot inference mode, the LLM predicts y_{test} for the test query x_{test}, without being provided any labeled examples (x, y) for the target task. The leftmost part of Fig. 1 shows the detailed process. First, the test query x_{test} is passed through the TF to obtain the hidden state $\boldsymbol{h}_{\text{zs}} = \text{TF}(x_{\text{test}})$. Then, the LM head computes the probability $P(c \mid x_{\text{test}})$ for each class c from $\boldsymbol{h}_{\text{zs}}$. Finally, the predicted label is selected via greedy decoding:

$$\hat{y}_{\text{zs}} = \text{argmax}_{c \in \mathcal{C}} P(c \mid x_{\text{test}}). \tag{4}$$

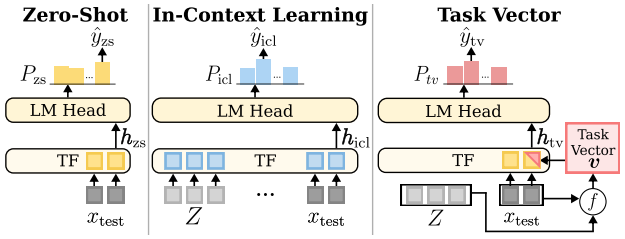

Figure 1. Comparison of three inference modes. In *zero-shot* inference mode (**left**), the model predicts the next token $\hat{y}_{\text{zs}}$ solely based on the test query x_{test}. In the *In-Context Learning* mode (**middle**), the model predicts the next token $\hat{y}_{\text{icl}}$ based on the concatenation of demonstrations Z and the query x_{test}. In the *task vector* mode (**right**), the model predicts the next token $\hat{y}_{\text{tv}}$ based on not only the query x_{test}, but also an injected task vector $\boldsymbol{v} = f(Z)$, which is added to the model activation. Here, the task vector $\boldsymbol{v}$ is constructed by a function f using the demonstrations Z.

Throughout the paper, we use the subscript 'zs' to note that the quantity is for the zero-shot inference mode.

3.3.2. In-Context Learning (ICL) Mode

Suppose we are given k demonstrations $Z = \{(x_i, y_i)\}_{i=1}^k$ sampled from the task distribution $\mathcal{D}$. As shown in the middle of Fig. 1, ICL prepends Z to the test query x_{test} and passes them through the model, which outputs $\boldsymbol{h}_{\text{icl}} = \text{TF}([Z||x_{\text{test}}])$, where $||$ represents the concatenation operator. The LM head then computes the probability $P(c \mid [Z||x_{\text{test}}])$ for each class c from $\boldsymbol{h}_{\text{icl}}$. We denote this probability by $P_{\text{icl}}(c \mid x_{\text{test}}, Z)$ to indicate the ICL inference mode. Lastly, the predicted label is then obtained as

$$\hat{y}_{\text{icl}} = \text{argmax}_{c \in \mathcal{C}} P_{\text{icl}}(c \mid x_{\text{test}}, Z). \tag{5}$$

3.3.3. Task Vector (TV) Mode

Inference using task vectors is a variant of ICL. This mode is motivated by the implicit Bayesian view (Xie et al., 2021), which posits that during ICL, the LLM predicts the label y conditioned on a latent task concept $\boldsymbol{v}$ inferred from demonstrations Z. Under this view, the predictive distribution computed by the LLM under the ICL can be expressed as

$$P(y \mid x, Z) = \int_{\boldsymbol{v}} P(y \mid x, \boldsymbol{v})\, P(\boldsymbol{v} \mid Z)\, d\boldsymbol{v}. \tag{6}$$

where $P(\boldsymbol{v} \mid Z)$ denotes the posterior over the latent task concept $\boldsymbol{v}$ inferred from the demonstrations Z, and $P(y \mid x, \boldsymbol{v})$ denotes the predictive distribution of the label y conditioned on the inferred task concept $\boldsymbol{v}$.

TV mode explicitly makes use of the decomposition of the predictive distribution as in the right-hand side of equation 6. In other words, the task vector mode is composed of two stages: (1) the *extraction* stage, which extracts a task vector

$\boldsymbol{v}$ from the model activation induced by the demonstrations Z, and (2) the *inference* stage, which applies the extracted vector $\boldsymbol{v}$ to the model activation, in the zero-shot inference. Below we formally describe each stage, which is illustrated in the rightmost column of Fig 1.

Extraction of task vector. Let f denote a task vector extraction function that takes demonstrations Z and query x as input, and outputs a task vector $\boldsymbol{v}$. Formally, we represent the task vector $\boldsymbol{v}$ as

$$\boldsymbol{v} = f(x, Z). \tag{7}$$

Inference using task vector. Recall that in the zero-shot inference mode (specified in Sec. 3.3.1), TF outputs the hidden state of the last token $\boldsymbol{h}_{\text{zs}}$, when the input is set to the test query x_{test}. In the TV mode, the task vector $\boldsymbol{v}$ extracted in equation 7 is used to update[1] the hidden state from $\boldsymbol{h}_{\text{zs}}$ to task-conditioned hidden state $\boldsymbol{h}_{\text{tv}} = \boldsymbol{h}_{\text{zs}} + \boldsymbol{v}$. For a given $\boldsymbol{h}_{\text{tv}}$, the LM head computes the probability $P(c \mid x_{\text{test}}, \boldsymbol{v})$ for each class c; we denote this probability by $P_{\text{tv}}(c \mid x_{\text{test}}, \boldsymbol{v})$ to indicate that this inference mode uses task vectors. The predicted label is then determined as

$$\hat{y}_{\text{tv}} = \text{argmax}_{c \in \mathcal{C}} P_{\text{tv}}(c \mid x_{\text{test}}, \boldsymbol{v}). \tag{8}$$

4. Measuring the Quality of Task Vector

We propose a metric that measures the quality of the task vector extracted by f. We first provide the motivation of our approach in Sec. 4.1, and then propose our metric in Sec. 4.2. Finally, we show that our metric serves as a reliable indicator of task vector quality in Sec. 4.3.

4.1. Motivation

Recall that the goal of using task vectors is to condense the task information inferred during ICL into a compact vector $\boldsymbol{v}$, enabling predictions similar to ICL without the overhead of processing demonstrations. However, prior work has relied solely on downstream task accuracy to evaluate the quality of the task vector. This evaluation practice offers limited insight into *why* one method outperforms another, and provides little guidance on *how* to design better task vector methods.

To address this, we propose a metric grounded in the goal of using task vectors: *enabling predictions similar to ICL*. If the task vector successfully captures the task information inferred during ICL, its inference should yield a predictive distribution *aligned* with that of ICL. We thus suggest to measure the discrepancy between the two distributions, formally defined as below.

[1]While our method additively updates the output of TF, there exist other task vector-based methods that combine a model activation and $\boldsymbol{v}$ in different ways. We focus on this additive case for notational simplicity.

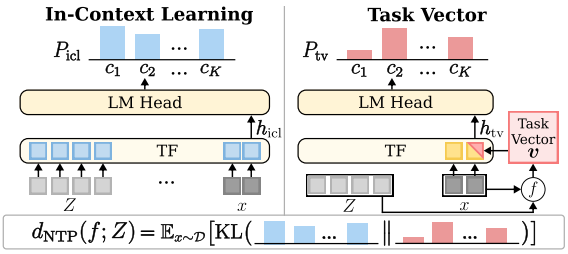

Figure 2. Overview of the proposed metric $d_{\text{NTP}}(f; Z)$ in equation 9, which measures the quality of the task vector extraction method f. In the ICL mode, the model gets demonstrations Z and test query x_{test} together to estimate the probability distribution P_{icl} for the next token (**left**). In the TV mode, the task vector $\boldsymbol{v}$ is injected in the hidden layer (instead of putting demonstrations Z in the input layer) to get the distribution P_{tv} for the next token (**right**). Our proposed metric measures the expected Kullback-Leibler (KL) divergence between these two distributions (P_{icl} and P_{tv}), thus checking whether the effect of using $\boldsymbol{v} = f(Z)$ in the hidden layer is equivalent to the effect of using Z in the input layer.

4.2. Proposed Metric

Recall that P_{icl} and P_{tv} are the probabilities of the next token computed by the model, for the ICL mode and the TV mode, respectively. Given demonstrations Z, we measure the quality of the task vector extraction method f as the discrepancy of the ICL mode and the TV mode in terms of the next token probability (NTP), denoted by

$$d_{\text{NTP}}(f; Z) = \mathbb{E}_{x \sim \mathcal{D}}\Big[\text{KL}\big(P_{\text{icl}}(\cdot \mid x, Z) \parallel P_{\text{tv}}(\cdot \mid x, f(Z))\big)\Big], \tag{9}$$

where KL represents the Kullback-Leibler (KL) divergence (Kullback & Leibler, 1951) operator. Here, P_{icl} serves as the reference distribution, and the metric quantifies how far P_{tv} deviates from the reference. See Fig. 2 for the illustration on our proposed metric.

A lower d_{NTP} indicates that P_{tv} is more closely aligned with P_{icl}. Thus, the task vector extraction method f with lower $d_{\text{NTP}}(f)$ is considered a more desirable method. Notably, our proposed metric evaluates the quality of task vector extraction methods without requiring labels for the test set.

4.3. Correlation of Proposed Metric With Test Accuracy

Now, a valid question is, whether our proposed metric $d_{\text{NTP}}(f)$ in equation 9 is a good indicator for the quality of the task vector extraction method f, in the practical sense. In this section, we empirically validate that $d_{\text{NTP}}(f)$ strongly correlates with the test accuracy when task vector $\boldsymbol{v} = f(Z)$ is used, across diverse classification benchmarks.

Experimental Setup. We evaluate on four classification benchmarks – AGNews, DBPedia (Zhang et al., 2015), MR (Pang & Lee, 2004), SST-2 (Socher et al., 2013). We

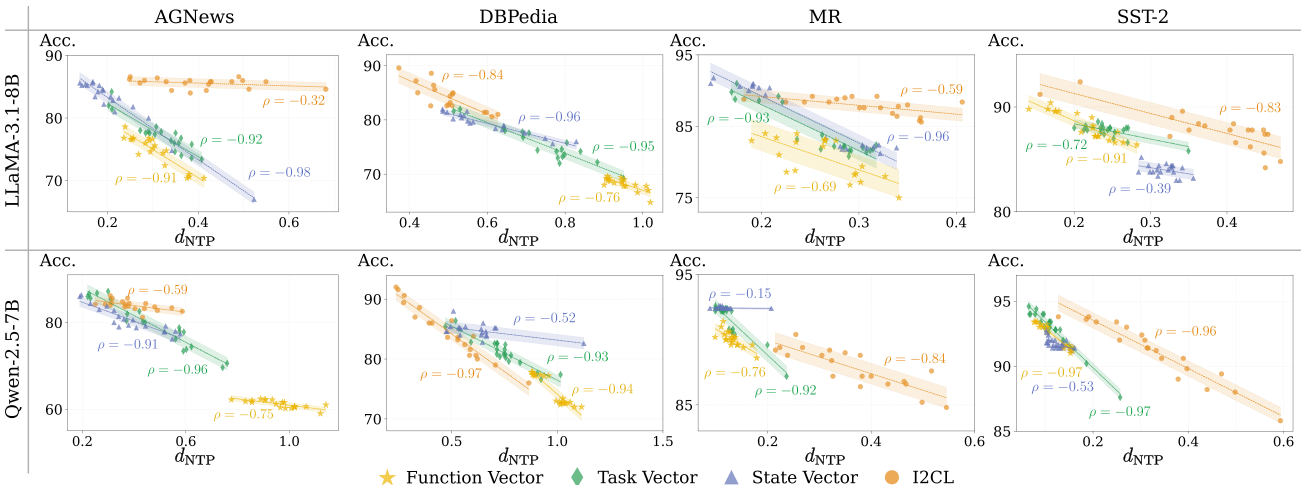

Figure 3. Correlation between the proposed discrepancy metric $d_{\text{NTP}}(f; Z)$ and the test accuracy, measured on various task vector extraction methods f and the demonstrations Z. We test on four variants of f: Function Vector (Todd et al., 2024), Task Vector (Hendel et al., 2023), State Vector (Li et al., 2024) and I2CL (Li et al., 2025c), each of which is shown in different colors. Here, each point corresponds to the result for different demonstrations Z. Pearson correlation coefficients (ρ) are reported for each method f. Across four classification benchmarks (columns) and two models (rows), lower $d_{\text{NTP}}(f; Z)$ consistently correlates with higher accuracy, validating our metric as a principled criterion for the quality of the task vector.

compute the metric $d_{\text{NTP}}(f)$ and the test accuracy for four task vector extraction methods f – Function Vector (Todd et al., 2024), Task Vector (Hendel et al., 2023), State Vector (Li et al., 2024), and I2CL (Li et al., 2025c). We test on LLaMA-3.1-8B (Dubey et al., 2024) and Qwen-2.5-7B (Qwen et al., 2024) models using 30 demonstrations Z, across 20 independent runs with randomly sampled demonstrations Z. Further details are provided in Appendix A.2.

Results. Fig. 3 presents scatter plots of $d_{\text{NTP}}(f)$ versus accuracy $\text{Acc}(f)$, where each point corresponds to each result obtained with different demonstrations Z. Across all tasks and models, lower $d_{\text{NTP}}(f)$ consistently correlates with higher accuracy, with most Pearson correlation coefficients exceeding 0.6 in magnitude. This strong correlation demonstrates that lower $d_{\text{NTP}}(f)$ indicates superior task vector quality. This result motivates developing a task vector extraction method aimed at reducing $d_{\text{NTP}}(f)$, as proposed in the next section.

5. Proposed Method: Linear Task Vector

In Sec. 4, we empirically observed that in the TV mode, the test accuracy negatively correlates with the discrepancy $d_{\text{NTP}}(f)$ measured for the task vector extraction method f. This motivates us to devise a task vector extraction method that achieves a small $d_{\text{NTP}}(f)$, which is expected to improve test accuracy when the task vector is used.

Based on this motivation, we propose Linear Task Vector (*LTV*), which leverages a linear mapping to compute task vectors that enable TV mode inference to closely resemble that of ICL. We first present the rationale behind our approach in Sec. 5.1, then formally describe *LTV* in Sec. 5.2.

5.1. Rationale Behind the Proposed Method

The proposed *LTV* method is designed to achieve the following two goals simultaneously. First, we aim to design a method f that achieves a small $d_{\text{NTP}}(f)$, meaning that the next token probability P_{tv} under TV mode closely replicates P_{icl} under ICL mode. Second, we seek to preserve the key advantage of ICL, which does not require updating parameters of a model. These goals lead us to formulate a proxy optimization problem which satisfies two conditions: (1) the proxy objective is provably related to the original objective d_{NTP}, and (2) the proxy problem has a closed-form solution.

Formulation of the proxy optimization problem. Given demonstrations Z, our goal is to find the optimal f that minimizes the discrepancy $d_{\text{NTP}}(f)$ between P_{icl} and P_{tv}. As depicted in Fig. 2, the predictive distributions P_{icl} and P_{tv} are obtained by applying the LM head to the respective hidden states $\boldsymbol{h}_{\text{icl}}$ and $\boldsymbol{h}_{\text{tv}}$. Thus, the metric $d_{\text{NTP}}(f)$ in equation 9 can be expressed as:

$$d_{\text{NTP}}(f) = \mathbb{E}\Big[\text{KL}\big(\sigma(\boldsymbol{W}_{\text{lm}}\boldsymbol{h}_{\text{icl}}) \parallel \sigma(\boldsymbol{W}_{\text{lm}}(\boldsymbol{h}_{\text{zs}} + f(Z)))\big)\Big],$$

where $\boldsymbol{h}_{\text{tv}} = \boldsymbol{h}_{\text{zs}} + \boldsymbol{v} = \boldsymbol{h}_{\text{zs}} + f(Z)$ is the hidden state of TV mode. Thus, minimizing $d_{\text{NTP}}(f)$ with respect to f leads to the following optimization problem:

$$\min_f \ \mathbb{E}\Big[\text{KL}\big(\sigma(\boldsymbol{W}_{\text{lm}}\boldsymbol{h}_{\text{icl}}) \parallel \sigma(\boldsymbol{W}_{\text{lm}}(\boldsymbol{h}_{\text{zs}} + f(Z)))\big)\Big]. \quad (10)$$

However, solving equation 10 requires iteratively updating the task vector extraction model f, as no closed-form solution exists. In order to preserve the key advantage of ICL

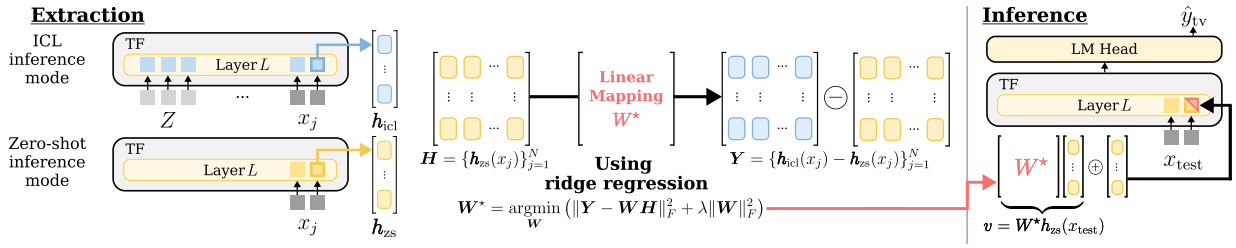

Figure 4. Overview of the proposed Linear Task Vector (*LTV*) method. Our method employs a linear mapping $\boldsymbol{W}$ that estimates the effect of demonstrations in the hidden space $(\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{zs}})$ from the hidden state $\boldsymbol{h}_{\text{zs}}$ of the zero-shot inference mode via ridge regression. In the extraction phase **(left)**, we use N unlabeled training queries $\{x_j\}_{j=1}^N$ to define (1) the regression target matrix $\boldsymbol{Y}$ as the concatenation of N column vectors $\{\boldsymbol{h}_{\text{icl}}(x_j) - \boldsymbol{h}_{\text{zs}}(x_j)\}_{j=1}^N$ and (2) the variable matrix $\boldsymbol{H}$ as the concatenation of N column vectors $\{\boldsymbol{h}_{\text{zs}}(x_j)\}_{j=1}^N$. Subsequently, a closed-form solution for the optimal linear mapping $\boldsymbol{W}^\star$ from $\boldsymbol{H}$ to $\boldsymbol{Y}$ is obtained. In the inference phase **(right)**, we use the optimal linear mapping $\boldsymbol{W}^\star$ to inject the task vector $\boldsymbol{v} = \boldsymbol{W}^\star \boldsymbol{h}_{\text{zs}}(x_{\text{test}})$ in the hidden state, for a given test query x_{test}.

(which does not require any model updates), we instead introduce a proxy objective in a way that the corresponding minimization problem has a closed-form solution on f. To this end, we define the proxy objective as the mean squared error (MSE) between the hidden states $\boldsymbol{h}_{\text{icl}}$ and $\boldsymbol{h}_{\text{tv}}$:

$$\mathcal{L}_{\text{MSE}}(f) = \mathbb{E}\left[\left\|\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{tv}}\right\|_2^2\right] = \mathbb{E}\left[\left\|\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{zs}} - f(Z)\right\|_2^2\right]. \tag{11}$$

Recall that d_{NTP} in equation 9 is defined as the expectation over query x. With explicit dependence on x and demonstrations Z, the proxy optimization problem becomes:

$$\min_f \mathbb{E}_{x\sim\mathcal{D}}\left[\left\|\boldsymbol{h}_{\text{icl}}(x, Z) - \boldsymbol{h}_{\text{zs}}(x) - f(x, Z)\right\|_2^2\right]. \tag{12}$$

A natural question is how this proxy objective $\mathcal{L}_{\text{MSE}}$ is related with the original objective d_{NTP}. The following proposition answers this question:

Proposition 5.1 (Relationship between d_{NTP} and $\mathcal{L}_{\text{MSE}}$)**.** *Assume the language modeling head $\boldsymbol{W}_{lm}$ has a bounded spectral norm $\|\boldsymbol{W}_{lm}\|_2 \leq C_1$ and the log-softmax function is C_2-Lipschitz in the ℓ_2-norm. Then, for any function f, the metric $d_{NTP}(f)$ is upper-bounded by the square root of $\mathcal{L}_{MSE}(f)$:*

$$d_{NTP}(f) \leq C_1 C_2 \sqrt{\mathcal{L}_{MSE}(f)}. \tag{13}$$

We defer the proof of Prop. 5.1 to Appendix A.1. This proposition indicates that we can reduce $d_{\text{NTP}}(f)$ by finding f which has small $\mathcal{L}_{\text{MSE}}(f)$.

Using a linear mapping to solve the proxy problem. We now focus on solving the proxy problem in equation 12, *i.e.,* finding f which outputs the task vector $\boldsymbol{v} = f(x, Z)$ that resembles $\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{zs}}$. In other words, the purpose of task vector $\boldsymbol{v}$ is to compensate the effect of demonstrations in the ICL mode (compared with the zero-shot inference mode which does not use demonstrations) in the hidden state. Thus, a natural approach to solve this problem is to find a mapping from $\boldsymbol{h}_{\text{zs}}$ (the hidden state for the zero-shot inference mode) to the target $\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{zs}}$. In order to get a closed-form solution for the optimization problem, we consider the simplest case[2] of using a *linear* mapping $\boldsymbol{W}$, which results in the following formulation:

$$f(x, Z) = \boldsymbol{W}(Z)\boldsymbol{h}_{\text{zs}}(x).$$

This leads to the following optimization problem:

$$\min_{\boldsymbol{W}} \mathbb{E}_{x\sim\mathcal{D}}\left[\left\|\boldsymbol{h}_{\text{icl}}(x, Z) - \boldsymbol{h}_{\text{zs}}(x) - \boldsymbol{W}\boldsymbol{h}_{\text{zs}}(x)\right\|_2^2\right]. \tag{14}$$

This reformulated problem in equation 14 has a closed-form solution, the details of which are given below.

5.2. Formal Description of Proposed Method

We now formalize Linear Task Vector (*LTV*), a task vector extraction method f which estimates a linear mapping $\boldsymbol{W}^\star$ that minimizes equation 14. Specifically, in the extraction stage, *LTV* estimates $\boldsymbol{W}^\star$ via ridge regression (Hoerl & Kennard, 1970). In the inference stage, the estimated optimal mapping $\boldsymbol{W}^\star$ is applied to compute the task vector.

Extraction Stage. Given demonstrations Z, we estimate $\boldsymbol{W} \in \mathbb{R}^{d\times d}$ by solving a regression problem mapping $\boldsymbol{h}_{\text{zs}}$ to $(\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{zs}})$, where d is the dimension of the hidden state. As shown in the leftmost column of Fig. 4, we first sample N unlabeled queries $\{x_j\}_{j=1}^N$ from the training set, and use them to compute the hidden states h. Let $\boldsymbol{H} \in \mathbb{R}^{d\times N}$ be the matrix whose j-th column is $\boldsymbol{h}_{\text{zs}}(x_j)$, and let $\boldsymbol{Y} \in \mathbb{R}^{d\times N}$ be the matrix whose j-th column is $(\boldsymbol{h}_{\text{icl}}(x_j) - \boldsymbol{h}_{\text{zs}}(x_j))$. To prevent ill-conditioning of $\boldsymbol{H}\boldsymbol{H}^\top$ when d is large relative to N, we employ the ridge regression

$$\boldsymbol{W}^\star = \underset{\boldsymbol{W}}{\text{argmin}} \left(\|\boldsymbol{Y} - \boldsymbol{W}\boldsymbol{H}\|_F^2 + \lambda\|\boldsymbol{W}\|_F^2\right), \tag{15}$$

where λ is the regularization parameter, and $\|\cdot\|_F$ denotes the Frobenius norm. This yields a closed-form solution, computed by solving a linear system

$$(\boldsymbol{H}\boldsymbol{H}^\top + \lambda\boldsymbol{I})\boldsymbol{W}^{\star\top} = \boldsymbol{H}\boldsymbol{Y}^\top, \tag{16}$$

where $\boldsymbol{I}$ denotes the identity matrix.

[2] In Appendix A.4, we empirically validate that this simple choice is both practical and effective compared to alternatives.

Table 1. Comparison of classification accuracy (%) between *LTV* and four baseline task vector methods, tested on eight benchmarks. Here, we also compare with two additional inference modes (zero-shot and ICL) for reference. The *LTV* method consistently achieves the highest average accuracy (Avg.) across all five models.

Models	Methods	AGNews	DBPedia	HateSpeech18	MR	SST-2	SST-5	Subj	TREC	Avg.
Qwen-2.5-7B (Qwen et al., 2024)	Zero-shot	73.80	74.80	58.40	68.00	54.80	20.20	48.80	66.40	58.15
	ICL	85.40 (±1.6)	97.32 (±0.3)	81.76 (±1.3)	93.44 (±0.5)	94.40 (±0.3)	48.56 (±2.2)	89.68 (±1.9)	87.32 (±2.6)	84.74
	Function Vector (Todd et al., 2024)	73.24 (±0.4)	75.56 (±0.5)	58.64 (±0.5)	58.96 (±1.2)	58.48 (±1.8)	20.20 (±0.0)	58.32 (±3.5)	68.08 (±0.7)	58.94
	Task Vector (Hendel et al., 2023)	74.76 (±2.1)	78.52 (±1.8)	63.92 (±3.3)	87.28 (±2.9)	**93.76** (±1.2)	23.52 (±4.7)	54.96 (±6.9)	74.12 (±3.2)	68.86
	State Vector (Li et al., 2024)	**78.52** (±0.3)	76.88 (±0.6)	57.84 (±1.3)	85.84 (±2.5)	86.64 (±3.6)	30.60 (±0.2)	70.12 (±2.7)	71.64 (±6.8)	69.76
	I2CL (Li et al., 2025c)	75.60 (±2.3)	75.68 (±0.4)	62.80 (±0.7)	79.24 (±0.4)	87.36 (±0.3)	22.96 (±0.3)	42.72 (±0.3)	56.68 (±1.8)	62.88
	***LTV* (Ours)**	77.32 (±6.7)	**90.68** (±2.2)	**75.24** (±1.2)	**90.56** (±1.3)	91.76 (±1.7)	**31.32** (±4.2)	**80.68** (±5.6)	**83.40** (±6.2)	**77.62**
Qwen-3-8B (Yang et al., 2025a)	Zero-shot	57.40	74.60	71.60	92.40	92.20	34.80	66.40	68.20	69.70
	ICL	86.56 (±2.3)	97.31 (±0.6)	83.44 (±1.0)	93.28 (±0.5)	94.48 (±0.5)	53.08 (±2.3)	93.36 (±0.7)	85.24 (±4.8)	85.84
	Function Vector (Todd et al., 2024)	61.68 (±0.8)	77.74 (±0.3)	71.56 (±0.4)	90.24 (±1.0)	91.36 (±0.2)	34.88 (±0.1)	67.20 (±2.2)	75.56 (±0.7)	71.28
	Task Vector (Hendel et al., 2023)	79.24 (±7.0)	81.49 (±1.4)	74.20 (±0.9)	92.36 (±0.2)	93.16 (±0.6)	34.08 (±0.4)	76.00 (±2.5)	73.56 (±7.9)	75.51
	State Vector (Li et al., 2024)	**81.24** (±4.2)	84.40 (±0.9)	69.92 (±2.0)	**92.44** (±0.1)	92.24 (±0.4)	33.92 (±1.1)	75.56 (±6.8)	78.32 (±2.6)	76.01
	I2CL (Li et al., 2025c)	62.92 (±0.2)	74.51 (±0.3)	70.12 (±0.6)	92.40 (±0.1)	**93.24** (±0.2)	34.92 (±0.1)	59.76 (±0.3)	78.36 (±0.9)	70.78
	***LTV* (Ours)**	77.80 (±2.1)	**94.20** (±1.7)	**76.04** (±3.4)	86.72 (±6.3)	90.40 (±2.2)	**39.40** (±7.1)	**90.00** (±2.2)	**81.96** (±4.6)	**79.57**
LLaMA-2-7B (Touvron et al., 2023)	Zero-shot	72.40	73.60	54.00	73.00	80.40	28.40	51.60	50.00	60.43
	ICL	84.48 (±5.1)	94.44 (±2.7)	64.60 (±9.0)	93.72 (±0.6)	93.52 (±1.2)	42.88 (±3.3)	54.48 (±6.5)	79.36 (±4.0)	75.94
	Function Vector (Todd et al., 2024)	71.64 (±0.7)	73.88 (±1.2)	55.80 (±0.6)	74.72 (±0.9)	78.60 (±0.1)	27.84 (±0.8)	50.04 (±0.2)	59.28 (±4.2)	61.48
	Task Vector (Hendel et al., 2023)	76.68 (±2.6)	78.68 (±0.6)	**71.28** (±2.8)	78.64 (±6.8)	81.60 (±5.6)	32.24 (±3.6)	48.72 (±2.6)	58.84 (±6.8)	65.84
	State Vector (Li et al., 2024)	74.36 (±8.4)	**90.36** (±1.0)	64.12 (±3.7)	88.48 (±3.8)	82.56 (±7.2)	33.68 (±8.4)	50.20 (±3.7)	62.20 (±7.5)	68.25
	I2CL (Li et al., 2025c)	72.96 (±1.1)	76.00 (±0.1)	51.48 (±0.4)	74.16 (±1.0)	80.16 (±0.1)	34.72 (±0.6)	**51.16** (±0.1)	60.24 (±0.5)	62.61
	***LTV* (Ours)**	**83.96** (±4.4)	88.84 (±1.6)	55.72 (±5.2)	**91.20** (±2.5)	**90.12** (±0.9)	**38.68** (±2.0)	50.32 (±0.3)	**70.68** (±5.4)	**71.19**
LLaMA-2-13B (Touvron et al., 2023)	Zero-shot	73.80	76.20	55.20	61.20	66.60	20.80	49.80	69.60	59.15
	ICL	88.04 (±1.7)	96.88 (±1.1)	77.40 (±2.2)	94.44 (±0.5)	94.80 (±0.7)	46.52 (±2.7)	82.28 (±7.1)	83.36 (±5.2)	82.97
	Function Vector (Todd et al., 2024)	74.00 (±1.2)	76.40 (±7.5)	54.08 (±0.4)	61.84 (±3.4)	72.44 (±6.0)	21.40 (±1.2)	50.00 (±0.0)	69.96 (±1.1)	60.02
	Task Vector (Hendel et al., 2023)	79.84 (±2.9)	80.72 (±1.9)	71.48 (±6.7)	83.16 (±0.8)	82.80 (±4.2)	33.64 (±2.1)	49.72 (±0.1)	73.84 (±5.3)	69.40
	State Vector (Li et al., 2024)	84.64 (±4.0)	89.48 (±4.9)	58.80 (±8.8)	89.20 (±2.8)	87.20 (±3.6)	36.08 (±3.0)	49.40 (±0.6)	66.96 (±1.0)	70.22
	I2CL (Li et al., 2025c)	78.88 (±0.3)	79.00 (±0.5)	54.48 (±0.1)	60.68 (±0.3)	64.80 (±0.4)	25.96 (±0.1)	50.00 (±0.0)	69.12 (±0.4)	60.37
	***LTV* (Ours)**	**86.68** (±1.7)	**93.20** (±0.5)	**72.08** (±2.2)	**90.20** (±2.0)	**88.96** (±2.1)	**41.88** (±2.2)	**78.76** (±2.7)	**83.92** (±6.3)	**79.46**
LLaMA-3.1-8B (Dubey et al., 2024)	Zero-shot	75.00	69.00	60.60	82.20	86.80	25.60	59.20	49.40	63.48
	ICL	87.16 (±1.2)	97.68 (±0.8)	74.32 (±8.1)	94.52 (±0.4)	94.20 (±1.1)	48.32 (±1.1)	85.96 (±5.3)	79.08 (±7.3)	82.66
	Function Vector (Todd et al., 2024)	76.16 (±0.1)	69.44 (±1.0)	63.00 (±0.2)	83.24 (±0.3)	87.32 (±0.6)	26.20 (±1.3)	59.52 (±0.2)	69.12 (±0.6)	66.75
	Task Vector (Hendel et al., 2023)	81.36 (±3.7)	83.24 (±1.5)	67.64 (±2.0)	83.48 (±3.5)	87.88 (±0.3)	34.76 (±1.9)	61.12 (±1.4)	66.48 (±1.0)	70.75
	State Vector (Li et al., 2024)	80.28 (±4.3)	80.80 (±1.5)	65.40 (±1.3)	85.96 (±4.9)	84.12 (±0.6)	36.60 (±1.6)	62.52 (±3.5)	67.28 (±1.9)	70.37
	I2CL (Li et al., 2025c)	76.76 (±0.4)	72.56 (±0.4)	62.24 (±0.5)	85.24 (±0.3)	**90.80** (±0.2)	32.48 (±0.4)	62.28 (±0.1)	49.00 (±0.5)	66.42
	***LTV* (Ours)**	**82.84** (±2.5)	**93.36** (±1.4)	**70.44** (±6.0)	**88.68** (±1.0)	90.08 (±1.2)	**38.20** (±3.4)	**67.16** (±11.4)	**72.88** (±7.2)	**75.46**

Inference Stage. As shown in the rightmost column of Fig. 4, we compute the task vector for a test query x_{test} as $\boldsymbol{v} = \boldsymbol{W}^{\star}\boldsymbol{h}_{\text{zs}}(x_{\text{test}})$. The task-conditioned hidden state is then obtained as $\boldsymbol{h}_{\text{tv}} = \boldsymbol{h}_{\text{zs}} + \boldsymbol{v}$, from which the LM head computes the predictive distribution P_{tv}.

6. Experiments

In this section, we empirically validate the proposed *LTV* method. Sec. 6.1 describes the evaluation setup, and Sec. 6.2 presents the experimental results. Codes are available at this GitHub repository.

6.1. Experimental Setup

Evaluation. Following prior work (Li et al., 2025c; Gao et al., 2025), we evaluate *LTV* on a diverse set of text classification benchmarks. We consider eight widely used datasets, organized by task type – sentiment analysis: SST-2, SST-5 (Socher et al., 2013), MR (Pang & Lee, 2005); topic classification: AGNews, DBPedia (Zhang et al., 2015); question classification: TREC (Voorhees & Tice, 2000); subjectivity classification: SUBJ (Pang & Lee, 2004); and hate speech detection: HateSpeech18 (De Gibert et al., 2018). We report the average accuracy over five independent runs, each using a different set of demonstrations with $k = 30$. For *LTV*, we report results using $N = 256$ train queries and a regularization parameter $\lambda = 5$ selected via grid search. Further details on evaluation are provided in Appendix A.2.

Models. We conduct experiments on five widely used LLMs: LLaMA-2-7B (Touvron et al., 2023), LLaMA-2-13B (Touvron et al., 2023), LLaMA-3.1-8B (Dubey et al., 2024), Qwen-2.5-7B (Qwen et al., 2024), and Qwen-3-8B (Yang et al., 2025a).

Baselines. We compare *LTV* with existing task vector methods which can be categorized into three groups, based on the module from which task vectors are extracted. First, we compare with *Task Vector* (Hendel et al., 2023), which extracts the task vector from the outputs of a single *decoder layer*. Second, we compare with methods that extract the task vector from a subset of *attention heads*. *Function Vector* (Todd et al., 2024) extracts from the final token of the input, whereas *State Vector* (Li et al., 2024) extracts from the last token of each demonstration and averages them. For all baselines above, the layer index or set of attention heads is selected using a held-out validation set. Third, we compare with *I2CL* (Li et al., 2025c), which extracts task vectors from both attention heads and MLP modules. In

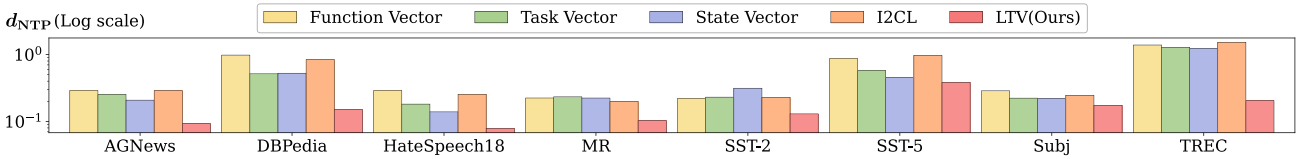

Figure 5. Comparison of d_{NTP} across *LTV* and four baselines on eight benchmarks, tested on LLaMA-3.1-8B. *LTV* consistently achieves the lowest d_{NTP} across all benchmarks, confirming that minimizing the proxy objective $\mathcal{L}_{\text{MSE}}$ using linear mapping effective reduces d_{NTP}.

Table 2. Comparison of the inference time per test sample (in seconds) and performance, evaluated on LLaMA-3.1-8B.

Method	Time (s)↓	Avg. Acc. (%)↑
Zero-shot	0.0220	63.48
Few-shot (ICL)	0.2238	82.66
Function Vector (Todd et al., 2024)	0.0614	66.75
State Vector (Li et al., 2024)	0.0292	70.37
I2CL (Li et al., 2025c)	0.0246	66.42
Task Vector (Hendel et al., 2023)	0.0236	70.75
***LTV* (Ours)**	**0.0226**	**75.46**

I2CL, the task vector is injected after scaling, where the scaling coefficient is *trained* using a validation set with true label. For fair comparison in a training-free setting, we use the default coefficient selected in (Li et al., 2025c), instead of training the coefficient. We also compare with standard zero-shot and ICL baselines. Details on each method are provided in Appendix A.3.

6.2. Results

***LTV* outperforms baselines.** Table 1 reports classification accuracy of *LTV* and baseline task vector methods across eight benchmarks and five LLMs. As shown in the rightmost column, *LTV* achieves the highest average accuracy on all five models. Notably, *LTV* outperforms the best baseline – *State Vector* – on LLaMA-2-13B by 9.2%, achieving a score of 79.46% compared to 70.22%. Moreover, *LTV* ranks first on the majority of individual tasks across all models; specifically, our method achieves the best performance on 31 out of 40 cases measured on 8 benchmarks and 5 models. This strong performance highlights that, among TV methods, *LTV* most effectively extracts task vectors that capture the effect of demonstrations.

***LTV* effectively reduces the proposed metric.** Recall that *LTV* is designed to achieve a small d_{NTP}. Given its strong performance over baselines, a natural follow-up is whether *LTV* indeed attains lower d_{NTP}. Fig. 5 compares d_{NTP} across five methods on eight benchmarks experimented with LLaMA-3.1-8B model. Compared to other baselines, *LTV* achieves the lowest d_{NTP} on all benchmarks. This confirms that *LTV* works as intended, successfully reducing the discrepancy between TV and ICL modes.

***LTV* incurs the lowest inference overhead.** We verify that the strong performance of *LTV* does not come at the expense of inference speed. Table 2 compares the average inference time per test example across methods. *LTV* achieves the shortest inference time among all task vector baselines (0.0226s per sample), comparable to zero-shot inference (0.0220s per sample). This efficiency is natural, as *LTV* intervenes only at the final-layer via a single matrix-vector multiplication.

Table 3. Effect of hyperparameters (the number of unlabeled queries N and the regularization parameter λ) on the performance of *LTV* and d_{NTP}. Experiments are conducted on LLaMA-3.1-8B, with gray rows indicating default settings.

N	Avg. Acc. (%) ↑	d_{NTP} ↓	λ	Avg. Acc. (%) ↑	d_{NTP} ↓
32	69.9	0.266	0.1	54.5	1.035
64	72.2	0.229	1.0	75.2	0.149
128	74.7	0.178	5.0	75.2	0.147
256	75.2	0.155	10.0	75.2	0.145

***LTV* is robust to hyperparameter choices.** One might wonder whether the performance of *LTV* relies on careful hyperparameter tuning. We examine the effect of the number of queries N and the ridge regularization parameter λ on both accuracy and d_{NTP}. Table 3 reports results obtained by varying one hyperparameter at a time. Halving N from the default value of 256 to 128 decreases accuracy by only 0.5% with a slight increase in d_{NTP}. Similarly, varying λ from the default value of 5.0 to 1.0 or 10.0 maintains comparable accuracy and d_{NTP}. These results confirm that *LTV* is robust to hyperparameter choices.

7. Conclusion

In this paper, we investigated the mechanics of task vectors (TVs) as a means of compressing In-Context Learning (ICL) demonstrations into the hidden states of LLMs. We introduced d_{NTP}, a novel metric that quantifies the discrepancy in next-token probability between TV-based and standard ICL-based inference. Our analysis reveals that d_{NTP} serves as a reliable proxy for downstream performance, enabling the evaluation of TV methods without the need for exhaustive task-specific testing. Leveraging these insights, we developed Linear Task Vector (LTV), a method that employs a closed-form linear mapping to minimize d_{NTP}. Experimental results across multiple benchmarks and architectures demonstrate that LTV outperforms existing training-free TV methods. By achieving superior accuracy with reduced inference latency, LTV offers a more efficient and effective framework for task compression in LLMs.

Impact Statement

This paper presents work aimed at advancing the field of machine learning. While our method can be used to improve task vector methods in in-context learning, we do not foresee any specific negative societal or ethical impacts beyond those commonly associated with LLMs.

References

Agarap, A. F. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

Agarwal, R., Singh, A., Zhang, L., Bohnet, B., Rosias, L., Chan, S., Zhang, B., Anand, A., Abbas, Z., Nova, A., et al. Many-shot in-context learning. *Advances in Neural Information Processing Systems*, 37:76930–76966, 2024.

Ahn, K., Cheng, X., Daneshmand, H., and Sra, S. Transformers learn to implement preconditioned gradient descent for in-context learning. *Advances in Neural Information Processing Systems*, 36:45614–45650, 2023.

Akyürek, E., Schuurmans, D., Andreas, J., Ma, T., and Zhou, D. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*, 2022.

Belanec, R., Ostermann, S., Srba, I., and Bielikova, M. Task prompt vectors: Effective initialization through multi-task soft prompt transfer. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 77–94. Springer, 2025.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chatterjee, A., Narahari, K. N., Joshi, M., and Agrawal, P. Semeval-2019 task 3: Emocontext contextual emotion detection in text. In *Proceedings of the 13th international workshop on semantic evaluation*, pp. 39–48, 2019.

De Gibert, O., Perez, N., García-Pablos, A., and Cuadros, M. Hate speech dataset from a white supremacy forum. *arXiv preprint arXiv:1809.04444*, 2018.

Dong, Q., Li, L., Dai, D., Zheng, C., Ma, J., Li, R., Xia, H., Xu, J., Wu, Z., Chang, B., et al. A survey on in-context learning. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 1107–1128, 2024.

Dong, Y., Jiang, J., Zhu, Z., and Ning, X. Understanding task vectors in in-context learning: Emergence, functionality, and limitations. *arXiv preprint arXiv:2506.09048*, 2025.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

Gao, B., Wang, X., Yang, Y., and Clifton, D. A. Optimization inspired few-shot adaptation for large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=rZ2nSt1X58.

Golchin, S., Chen, Y., Han, R., Gandhi, M., Yu, T., Mishra, S., Surdeanu, M., Agarwal, R., Lee, C.-Y., and Pfister, T. Towards compute-optimal many-shot in-context learning. In *Second Conference on Language Modeling*, 2025.

Han, S., Song, J., Gore, J., and Agrawal, P. Emergence and effectiveness of task vectors in in-context learning: An encoder decoder perspective. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=0ysC6VS0y3.

Hendel, R., Geva, M., and Globerson, A. In-context learning creates task vectors. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Hoerl, A. E. and Kennard, R. W. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

Ilharco, G., Ribeiro, M. T., Wortsman, M., Schmidt, L., Hajishirzi, H., and Farhadi, A. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*, 2023.

Jiang, J., Dong, Y., Zhou, J., and Zhu, Z. From compression to expansion: A layerwise analysis of in-context learning. *arXiv preprint arXiv:2505.17322*, 2025.

Jukić, J. and Šnajder, J. Disentangling latent shifts of in-context learning with weak supervision. *arXiv preprint arXiv:2410.01508*, 2024.

Kang, J., Lee, S., Park, S., Park, S., Kim, T., Kim, J., Lee, R., and Song, K. Adaptive task vectors for large language models. *arXiv preprint arXiv:2506.03426*, 2025.

Kullback, S. and Leibler, R. A. On information and sufficiency. *The annals of mathematical statistics*, 22(1): 79–86, 1951.

Li, D., Liu, Z., Hu, X., Sun, Z., Hu, B., and Zhang, M. In-context learning state vector with inner and momentum optimization. *Advances in Neural Information Processing Systems*, 37:7797–7820, 2024.

Li, H., Zhang, Y., Zhang, S., Wang, M., Liu, S., and Chen, P.-Y. When is task vector provably effective for model

editing? a generalization analysis of nonlinear transformers. *arXiv preprint arXiv:2504.10957*, 2025a.

Li, J., Li, Y., Han, L., Tang, R., and Wang, W. Towards generalizable implicit in-context learning with attention routing. *arXiv preprint arXiv:2509.22854*, 2025b.

Li, Y., Ildiz, M. E., Papailiopoulos, D., and Oymak, S. Transformers as algorithms: Generalization and stability in in-context learning. In *International conference on machine learning*, pp. 19565–19594. PMLR, 2023.

Li, Z., Xu, Z., Han, L., Gao, Y., Wen, S., Liu, D., Wang, H., and Metaxas, D. N. Implicit in-context learning. In *The Thirteenth International Conference on Learning Representations*, 2025c. URL https://openreview.net/forum?id=G7u4ue6ncT.

Liu, S., Ye, H., Xing, L., and Zou, J. Y. In-context vectors: Making in context learning more effective and controllable through latent space steering. In *Forty-first International Conference on Machine Learning*, 2024.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Mittal, S., Elmoznino, E., Gagnon, L., Bhardwaj, S., Lajoie, G., and Sridhar, D. Does learning the right latent variables necessarily improve in-context learning? In *Forty-second International Conference on Machine Learning*, 2025.

Mu, J., Li, X., and Goodman, N. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36:19327–19352, 2023.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Ortiz-Jimenez, G., Favero, A., and Frossard, P. Task arithmetic in the tangent space: Improved editing of pre-trained models. *Advances in Neural Information Processing Systems*, 36:66727–66754, 2023.

Pang, B. and Lee, L. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. *arXiv preprint cs/0409058*, 2004.

Pang, B. and Lee, L. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. *arXiv preprint cs/0506075*, 2005.

Panwar, M., Ahuja, K., and Goyal, N. In-context learning through the bayesian prism. *arXiv preprint arXiv:2306.04891*, 2023.

Qwen, A. Y., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al. Qwen2. 5 technical report. *arXiv preprint*, 2024.

Saglam, B., Hu, X., Yang, Z., Kalogerias, D., and Karbasi, A. Learning task representations from in-context learning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6634–6663, 2025.

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

Todd, E., Li, M., Sharma, A. S., Mueller, A., Wallace, B. C., and Bau, D. Function vectors in large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Von Oswald, J., Niklasson, E., Randazzo, E., Sacramento, J., Mordvintsev, A., Zhmoginov, A., and Vladymyrov, M. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pp. 35151–35174. PMLR, 2023.

Voorhees, E. M. and Tice, D. M. Building a question answering test collection. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 200–207, 2000.

Wang, F., Yan, J., Zhang, Y., and Lin, T. Elicit: Llm augmentation via external in-context capability. In *The Thirteenth International Conference on Learning Representations*, 2025.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

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

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Yang, L., Lin, Z., Lee, K., Papailiopoulos, D., and Nowak, R. D. Task vectors in in-context learning: Emergence, formation, and benefit. *CoRR*, 2025b.

Yin, K. and Steinhardt, J. Which attention heads matter for in-context learning? *arXiv preprint arXiv:2502.14010*, 2025.

Zhang, F. Z., Albert, P., Rodriguez-Opazo, C., van den Hengel, A., and Abbasnejad, E. Knowledge composition using task vectors with learned anisotropic scaling. *Advances in Neural Information Processing Systems*, 37: 67319–67354, 2024.

Zhang, X., Zhao, J., and LeCun, Y. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

Zhang, Y., Zhang, F., Yang, Z., and Wang, Z. What and how does in-context learning learn? bayesian model averaging, parameterization, and generalization. *arXiv preprint arXiv:2305.19420*, 2023.

Zhou, Y., Li, J., Xiang, Y., Yan, H., Gui, L., and He, Y. The mystery of in-context learning: A comprehensive survey on interpretation and analysis. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 14365–14378, 2024.

A. Appendix

A.1. Proof of Proposition 5.1

We provide detailed proof for the proposition 5.1 presented in Sec. 5.

Proof. Recall that, throughout the paper, P denotes the next-token distribution *restricted* to the label set $\mathcal{C}$ as defined in equation 2. Accordingly, the P_{icl} and P_{tv} in the KL divergence in $d_{\text{NTP}}(f)$ in equation 9 are also computed over $\mathcal{C}$.

Given demonstrations Z and a query x, recall that

$$\boldsymbol{h}_{\text{icl}} \coloneqq \boldsymbol{h}_{\text{icl}}(x, Z), \qquad \boldsymbol{h}_{\text{tv}} \coloneqq \boldsymbol{h}_{\text{tv}}(x, \boldsymbol{v})$$

denote the hidden states of the last token in the final-layer under ICL and TV inference, respectively.

Restricted logits. For the LM head $\boldsymbol{W}_{\text{lm}}$, define $\boldsymbol{W}_{\mathcal{C}} \in \mathbb{R}^{K\times d}$ as the submatrix of $\boldsymbol{W}_{\text{lm}}$ formed by selecting the rows indexed by the label set $\mathcal{C}$ (where $K = |\mathcal{C}|$). Then

$$\|\boldsymbol{W}_{\mathcal{C}}\|_2 \le \|\boldsymbol{W}_{\text{lm}}\|_2 \le C_1,$$

since removing rows cannot increase the spectral norm.

Define the *restricted* logits

$$\boldsymbol{z} \coloneqq \boldsymbol{W}_{\mathcal{C}}\, \boldsymbol{h}_{\text{icl}}, \qquad \tilde{\boldsymbol{z}} \coloneqq \boldsymbol{W}_{\mathcal{C}}\, \boldsymbol{h}_{\text{tv}},$$

and the corresponding restricted predictive distributions

$$\boldsymbol{p} \coloneqq \mathrm{softmax}(\boldsymbol{z}), \qquad \boldsymbol{q} \coloneqq \mathrm{softmax}(\tilde{\boldsymbol{z}}).$$

Then the discrepancy for each query x appearing in $d_{\text{NTP}}(f)$ in equation 9 is equal to $D_{\text{KL}}(\boldsymbol{p}\|\boldsymbol{q})$.

Step 1: Bounding the restricted logit distance. By sub-multiplicativity and the spectral-norm bound,

$$\|\boldsymbol{z} - \tilde{\boldsymbol{z}}\|_2 = \|\boldsymbol{W}_{\mathcal{C}}(\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{tv}})\|_2 \le \|\boldsymbol{W}_{\mathcal{C}}\|_2\, \|\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{tv}}\|_2 \le C_1\, \|\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{tv}}\|_2. \tag{17}$$

Step 2: Bounding the restricted KL divergence via Lipschitz log-softmax. Let $\phi(\cdot) \coloneqq \log\mathrm{softmax}(\cdot)$ denote the log-softmax map on $\mathbb{R}^K$. By assumption, ϕ is C_2-Lipschitz in ℓ_2:

$$\|\phi(\boldsymbol{z}) - \phi(\tilde{\boldsymbol{z}})\|_2 \le C_2 \|\boldsymbol{z} - \tilde{\boldsymbol{z}}\|_2. \tag{18}$$

For any coordinate $i \in \{1, \dots, K\}$,

$$|\log \boldsymbol{p}_i - \log \boldsymbol{q}_i| = |\phi(\boldsymbol{z})_i - \phi(\tilde{\boldsymbol{z}})_i| \le \|\phi(\boldsymbol{z}) - \phi(\tilde{\boldsymbol{z}})\|_2 \le C_2 \|\boldsymbol{z} - \tilde{\boldsymbol{z}}\|_2.$$

Therefore,

$$\begin{aligned} D_{\text{KL}}(\boldsymbol{p}\|\boldsymbol{q}) &= \sum_{i=1}^{K} \boldsymbol{p}_i (\log \boldsymbol{p}_i - \log \boldsymbol{q}_i) \le \sum_{i=1}^{K} \boldsymbol{p}_i\, |\log \boldsymbol{p}_i - \log \boldsymbol{q}_i| \\ &\le \sum_{i=1}^{K} \boldsymbol{p}_i\, (C_2 \|\boldsymbol{z} - \tilde{\boldsymbol{z}}\|_2) = C_2 \|\boldsymbol{z} - \tilde{\boldsymbol{z}}\|_2. \end{aligned} \tag{19}$$

Step 3: Combining and taking expectation. Combining equation 17 and equation 19 yields

$$D_{\text{KL}}(\boldsymbol{p}\|\boldsymbol{q}) \le C_1 C_2\, \|\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{tv}}\|_2.$$

Taking expectation over $x \sim \mathcal{D}$, we have

$$\begin{aligned} d_{\text{NTP}}(f) = \mathbb{E}_{x\sim\mathcal{D}}\big[D_{\text{KL}}(\boldsymbol{p}\|\boldsymbol{q})\big] &\le C_1 C_2\, \mathbb{E}_{x\sim\mathcal{D}}\big[\|\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{tv}}\|_2\big] \\ &\le C_1 C_2 \sqrt{\mathbb{E}_{x\sim\mathcal{D}}\big[\|\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{tv}}\|_2^2\big]} \\ &= C_1 C_2 \sqrt{\mathcal{L}_{\text{MSE}}(f)}. \end{aligned}$$

This completes the proof. □

Table 4. Overview of the benchmark datasets used in our experiments, including their prompting templates and label sets. {Sentence} and {Label} are placeholders for input query x and label y, respectively.

Dataset	Prompt Template	Labels
AGNews	News: {Sentence} Type: {Label}	World / Sports / Business / Technology
DBPedia	Input: {Sentence} Label: {Label}	company / school / artist / athlete / politics / transportation / building / nature / village / animal / plant / album / film / book
HateSpeech18	Text: {Sentence} Label: {Label}	neutral / hate
MR	Review: {Sentence} Sentiment: {Label}	negative / positive
SST-2	Review: {Sentence} Sentiment: {Label}	negative / positive
SST-5	Sentence: {Sentence} Sentiment: {Label}	terrible / negative / neutral / positive / great
SUBJ	Sentence: {Sentence} Label: {Label}	objective / subjective
TREC	Question: {Sentence} Answer Type: {Label}	Abbreviation / Entity / Person / Location / Number

A.2. Details on Evaluation

We provide detailed information about our evaluation setup. Following experiments conducted in Li et al. (2025c), we employ the same prompt templates and label sets, as summarized in Table 4. However, EmoC (Chatterjee et al., 2019) is excluded as it is no longer publicly available. For labels consisting of multiple tokens, we use the probability of the first token when computing equation 2, following Hendel et al. (2023); Todd et al. (2024); Li et al. (2025c). All experiments were conducted using a single NVIDIA RTX A6000 GPU.

All experiments are conducted with $k = 30$ demonstrations Z, sampled from the training dataset. Note that for classification tasks, the label set $\mathcal{C}$ contains $|\mathcal{C}| = K$ labels. For both task vector extraction and ICL inference, we ensure that each label is represented by an equal number of examples in Z. Specifically, we include the maximum number of examples per label such that all labels are equally represented while staying within the limit of k demonstrations. For example, in the case of AGNews where the number of classes K is 4, we include 7 examples per label under the $k = 30$ setting, resulting in a total of 28 demonstrations. This uniform label distribution is maintained across all experiments involving demonstrations.

A.3. Details on Baselines

In this section, we provide the details of each baseline along with our implementation. For each baseline, we describe (1) the extraction phase, (2) the inference phase, and (3) the implementation details.

- **Task Vector** (Hendel et al., 2023). In the extraction phase, task vectors are extracted from hidden states at the last token position. In the inference phase, the original hidden state of the model at a specific layer is replaced with the extracted vector (*i.e.,* $\boldsymbol{h}_{\text{tv}} = \boldsymbol{v}$). Following the original study, we select the optimal layer independently for each task using a validation set with 32 labeled examples.
- **Function Vector** (Todd et al., 2024). In the extraction phase, task vectors are extracted from the activations of attention heads that are selected based on their causal influence on increasing the probability of the correct label. In the inference phase, the task vector is added to the hidden state at a chosen injection layer. In our implementation, we follow the original paper and identify the top-10 attention heads based on their causal influence. We average their representations over 20 independent trials and select the optimal layer using a validation set with 32 labeled examples. Detailed implementation refers to the official repository at `https://github.com/ericwtodd/function_vectors`.
- **State Vector** (Li et al., 2024). In the extraction phase, task vectors are formed by collecting attention activations at the separator tokens of each demonstration. The activations are gathered across the initial layers. In the inference phase, these stored activations are added to the model activations. We employ the inner optimization strategy from the original paper,

Table 5. Comparison of design choices for the task vector $f(x, Z)$ that maps $\boldsymbol{h}_{\text{zs}}(x)$ to the estimated hidden state difference $\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{zs}}$. We compare three ways of designing such mapping: the *constant mapping* uses a fixed vector $f(x, Z) = \boldsymbol{c}$, the *linear mapping* which uses a linear transformation $f(x, Z) = \boldsymbol{W}\boldsymbol{h}_{\text{zs}}(x)$, and the *2-layer MLP based mapping* which uses a ReLU network $f(x, Z) = \boldsymbol{W}_2\sigma_{\text{ReLU}}(\boldsymbol{W}_1\boldsymbol{h}_{\text{zs}}(x))$. Note that the *linear mapping* is used in our proposed LTV method in Sec. 5. We report the accuracy averaged over 8 classification benchmarks used in Sec. 6, our proposed d_{NTP} metric and $\mathcal{L}_{\text{MSE}}$. Experiments are conducted on LLaMA-3.1-8B (Dubey et al., 2024).

Design Choices	$f(x, Z)$	Avg. Acc. ↑	d_{NTP} ↓	$\mathcal{L}_{\text{MSE}}$ ↓
Constant Mapping	$\boldsymbol{c}$	55.72	0.694	2.84
Linear Mapping (Ours)	$\boldsymbol{W}\boldsymbol{h}_{\text{zs}}(x)$	75.21	0.146	1.22
2-layer MLP based Mapping	$\boldsymbol{W}_2\sigma_{\text{ReLU}}(\boldsymbol{W}_1\boldsymbol{h}_{\text{zs}}(x))$	77.74	0.132	0.64

which averages activations extracted from multiple demonstration examples. The optimal depth of the initial layers is selected for each task via a validation set with 32 labeled examples. The method is implemented using the official code at https://github.com/HITsz-TMG/ICL-State-Vector.

- **I2CL** (Li et al., 2025c). In the extraction phase, task vectors are extracted from the output activations of both attention and MLP modules at the last token position. These activations are averaged over each demonstration example. At inference time, a linear combination of these attention and MLP task vectors is additively injected into the residual stream at every layer. In our experiments, we use the initial coefficient values ($\lambda = 0.1, \beta = 1$) for the linear combination as suggested in the original paper. Code and detailed implementation instructions are available at https://github.com/LzVv123456/I2CL.

A.4. Additional Experiments

In Sec. 5.1 of the main paper, we propose to solve the proxy problem in equation 14 by minimizing $\mathcal{L}_{\text{MSE}}$ in equation 11. We interpret the proxy problem as finding a task vector $\boldsymbol{v} = f(x, Z)$ that compensates for the effect of demonstrations in the hidden state. Specifically, the goal of using task vectors is to approximate the hidden state difference $\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{zs}}$ between ICL and zero-shot inference:

$$\min_f \mathbb{E}_{x\sim\mathcal{D}}\left[\left\|\boldsymbol{h}_{\text{icl}}(x, Z) - \boldsymbol{h}_{\text{zs}}(x) - f(x, Z)\right\|_2^2\right].$$

In the main paper, we consider f as a mapping from $\boldsymbol{h}_{\text{zs}}$ to the target $\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{zs}}$ and adopt a linear mapping $f(x, Z) = \boldsymbol{W}\boldsymbol{h}_{\text{zs}}(x)$ for its closed-form solution:

$$\min_{\boldsymbol{W}} \mathbb{E}_{x\sim\mathcal{D}}\left[\left\|\boldsymbol{h}_{\text{icl}}(x, Z) - \boldsymbol{h}_{\text{zs}}(x) - \boldsymbol{W}\boldsymbol{h}_{\text{zs}}(x)\right\|_2^2\right].$$

Here, we compare this choice against alternative designs and validate its effectiveness.

Design choices for the task vector. Given the strong performance of our proposed *LTV* method as shown in Sec. 6.2, the choice of employing a linear mapping naturally raises two questions: (1) Is it necessary to model f as a mapping, the output of which is dependent on the input $\boldsymbol{h}_{\text{zs}}(x)$, or does modeling f as a simple constant vector suffice? (2) How effective is our linear mapping compared to a more expressive alternative trained via gradient descent? To answer these two questions, we compare our linear mapping approach against a constant mapping baseline and a more expressive alternative (2-layer MLP based mapping):

- **Constant Mapping:** Modeling the output of f as a fixed vector $\boldsymbol{c} \in \mathbb{R}^d$ independent of the query x, *i.e.,* $f(x, Z) = \boldsymbol{c}$. The optimal solution is given by the empirical mean of $\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{zs}}$, represented as

$$\boldsymbol{c}^* = \frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{h}_{\text{icl}}(x_i, Z) - \boldsymbol{h}_{\text{zs}}(x_i)\right)$$

  where N unlabeled train queries $\{x_i\}_{i=1}^N$ are used.
- **Linear Mapping (Ours):** Mapping from $\boldsymbol{h}_{\text{zs}}$ to the target $\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{zs}}$ using a linear transformation, *i.e.,* $f(x, Z) = \boldsymbol{W}\boldsymbol{h}_{\text{zs}}(x)$, where $\boldsymbol{W} \in \mathbb{R}^{d\times d}$. This has a closed-form solution when we formulate it as a ridge regression problem; see Section 5.2.
- **2-layer MLP based Mapping:** Mapping from $\boldsymbol{h}_{\text{zs}}$ to the target $\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{zs}}$ using a 2-layer ReLU (Agarap, 2018) network: $f(x, Z) = \boldsymbol{W}_2\,\sigma_{\text{ReLU}}(\boldsymbol{W}_1\boldsymbol{h}_{\text{zs}}(x))$, where $\boldsymbol{W}_1, \boldsymbol{W}_2 \in \mathbb{R}^{d\times d}$ are learnable parameter and σ_{ReLU} denotes ReLU activation function. This is more expressive than the linear mapping, but requires iterative optimization. We train this network by

minimizing the $\mathcal{L}_{\text{MSE}}$ loss using AdamW (Loshchilov & Hutter, 2017) with learning rate 10^{-3}, cosine scheduler with warmup ratio 0.1, batch size 8, and 20 epochs over $N = 256$ unlabeled train queries with $k = 30$ demonstrations Z.

As shown in Table 5, the constant mapping achieves only 55.72% accuracy with $\mathcal{L}_{\text{MSE}} = 2.84$, performing significantly worse than our proposed linear mapping (75.21% accuracy, $\mathcal{L}_{\text{MSE}} = 1.22$) by a margin of 19.49% in accuracy and 1.62 in MSE. This confirms that a fixed hidden state vector is not sufficient for approximating the effect of demonstrations. While 2-layer MLP based mapping achieves the lowest $\mathcal{L}_{\text{MSE}}$ of 0.64 due to its expressiveness, our proposed linear mapping attains competitive accuracy (75.21% vs 77.74%) and d_{NTP} (0.146 vs 0.132) without requiring iterative optimization. These results demonstrate that linear mapping is a viable choice, balancing both effectiveness and computational efficiency.