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

Jihoon Kwon[*]
Seoul National University
kog0712@snu.ac.kr

Jiwon Choi[*]
Yonsei University
jiii111@yonsei.ac.kr

Jy-yong Sohn[†]
Yonsei University
jysohn1108@yonsei.ac.kr

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 offer a promising alternative by compressing demonstrations into compact hidden-state representations, their quality has been evaluated only through downstream task accuracy. This indirect criterion provides limited insight into how to design more effective task vector extraction methods. In this paper, we posit that inference using task vectors should align their predictive distribution with that of ICL. To quantify this, we introduce d_{NTP}, a metric that measures the discrepancy in next-token probabilities between task vector-based and ICL-based inference. Our empirical analysis reveals that d_{NTP} serves as a 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 and five LLMs, LTV consistently outperforms existing task vector baselines, improving average accuracy by 9.2% while reducing inference latency. We further show that LTV outperforms the baselines on regression tasks. Moreover, we investigate the transferability of LTV across different model scales; an aspect that has remained nascent in task vector research. Specifically, we empirically show that task vectors from a larger model can enhance a smaller model's performance by 6.4%, suggesting a new utility for extracted task representations.

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 [7, 52]. 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 [2, 11]. 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 [35, 30, 14, 16]. 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 [18, 46, 32, 26]. 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 [17, 55, 12]. By applying this vector to zero-shot inference, where no demonstrations are provided, the model achieves substantial performance gains on the task [18]. 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.

[*]Equal Contribution [†]Corresponding author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

In this paper, we posit that distributional alignment with ICL is a desirable property of task vectors, and a useful criterion for designing 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 an 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 linear mapping that estimates the effect of demonstrations, and uses the closed-form solution of a regression problem to extract task vectors.
- In experiments, *LTV* consistently outperforms existing TV methods on eight classification benchmarks and five LLMs, improving average accuracy by 9.2% while reducing inference latency. Furthermore, *LTV* outperforms the baselines on regression tasks. Finally, we extend our study to the transferability of task vectors, a dimension largely unexplored in existing research. By applying task vectors extracted from a larger model to smaller ones – which may have limited capacity or context lengths – we achieve a 6.4% improvement in classification accuracy.

2 Related Work

Task Vectors. A pioneering work Ilharco et al. [21] 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 [38, 57, 27]. Recent works have shown that task vectors can also be extracted from representation spaces, such as activation spaces [18, 55] or soft prompt spaces [5].

In-Context Learning. ICL enables LLMs to adapt to new tasks by simply prepending query-label pairs as demonstrations to the model input [7]. The success of ICL has motivated various theoretical interpretations [60, 49, 3, 37, 56, 4, 29]. A notable line of work interprets ICL as implicit Bayesian inference [53, 41, 59]: 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 [34], a line of work we describe in detail below.

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 [30, 2]. 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) [30, 22, 14, 28]. Another line of work explores task vectors in ICL, offering a training-free alternative that enables task adaptation.

Numerous methods have been proposed for extracting task vectors in ICL, demonstrating performance gains over zero-shot inference [18, 46, 32, 26, 30, 23, 51]. As the model activations span multiple modules, existing methods vary widely in where and how task vectors are extracted. This diversity underscores the need for a direct criterion to evaluate the quality of extracted task vectors.

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 [48] (TF) decoder consisting of L layers, which transforms the sequence of embedding vectors into a sequence of hidden states,

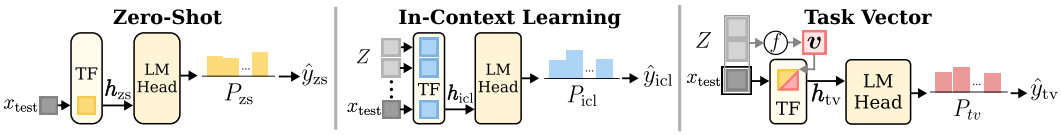

Figure 1: Comparison of three inference modes. In *zero-shot* inference mode (**left**), the model predicts the next token $\hat{y}_{\text{zs}}$ solely based on the test query x_{test}. In the *In-Context Learning* mode (**middle**), the model predicts the next token $\hat{y}_{\text{icl}}$ based on the 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.

- 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 [18, 30, 44] 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 $\mathcal{C} \subseteq \mathcal{U}$ 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 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:

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

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

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

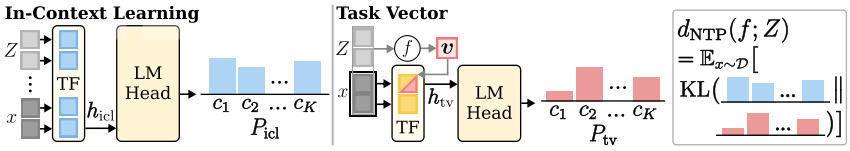

Figure 2: Overview of the proposed metric $d_{\text{NTP}}(f; Z)$ in equation 9, which measures the quality of the task vector extraction method f. In the ICL mode, the model gets demonstrations Z and test query x_{test} together to estimate the probability distribution P_{icl} for the next token (**left**). 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 $\boldsymbol{v} = f(Z)$ is equivalent to the effect of using Z in the input.

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 [53], 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 Z, and $P(y \mid x, \boldsymbol{v})$ denotes the predictive distribution of the label y conditioned on the inferred 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 shown 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

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

For notational simplicity, we suppress the dependence of $\boldsymbol{v}$ on x and Z when it is clear from context.

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[2] 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 an indicator of task vector quality in Sec. 4.3.

[2]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.

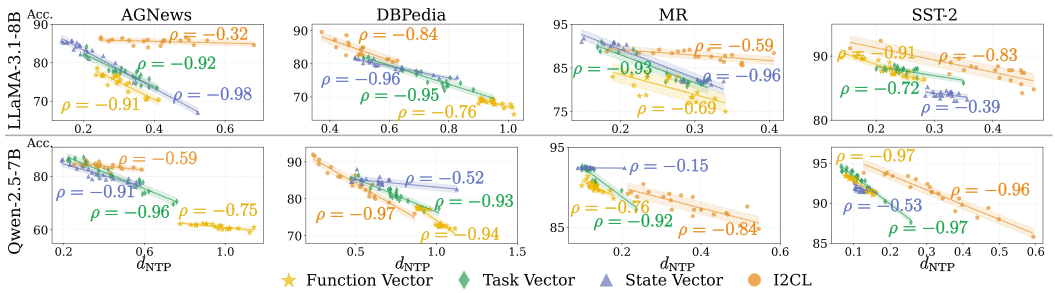

Figure 3: Correlation between the proposed discrepancy metric $d_{\text{NTP}}(f; Z)$ and the test accuracy, measured on various task vector extraction methods f and the demonstrations Z. We test on four variants of f: Function Vector [46], Task Vector [18], State Vector [26], and I2CL [30], each of which 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.

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*. This goal means that, in the formulation of Sec. 3.3, adding $\boldsymbol{v}$ to $\boldsymbol{h}_{\text{zs}}$ should approximate the effect of prepending demonstrations Z to the query x. The resulting distribution P_{tv} in equation 8 is thus intended to approximate P_{icl} in equation 5. Therefore, 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.

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

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

where D_{KL} represents the Kullback-Leibler (KL) divergence [24] 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 of 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 [58], MR [39], SST-2 [45]. We compute the metric $d_{\text{NTP}}(f)$ and the test accuracy for four task vector extraction methods f – Function Vector [46], Task Vector [18], State Vector [26], and I2CL [30]. We test on LLaMA-3.1-8B [13] and Qwen-2.5-7B [43] models using 30 demonstrations Z, across 20 independent runs with randomly sampled demonstrations Z. See Appendix A.1 for further details.

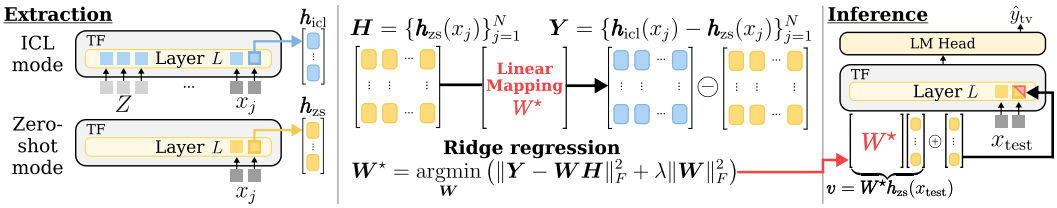

Figure 4: Overview of our Linear Task Vector (*LTV*) method. Our method employs a linear mapping $\boldsymbol{W}$ that estimates the effect of demonstrations in the hidden space ($\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{zs}}$) from the hidden state $\boldsymbol{h}_{\text{zs}}$ of the zero-shot 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 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}.

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 the 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.

With these goals in mind, we briefly explain the intuition behind our approach before formulating the optimization problem. Note that the only difference in computing P_{icl} and P_{tv} is the input hidden state, $\boldsymbol{h}_{\text{icl}}$ and $\boldsymbol{h}_{\text{tv}}$, respectively. We therefore expect that making $\boldsymbol{h}_{\text{tv}}$ close to $\boldsymbol{h}_{\text{icl}}$ will make P_{tv} close to P_{icl}. This motivates finding f that extracts a task vector $\boldsymbol{v}$ to bring $\boldsymbol{h}_{\text{tv}}$ close to $\boldsymbol{h}_{\text{icl}}$ for a given test query x_{test}. Since $\boldsymbol{h}_{\text{tv}} = \boldsymbol{h}_{\text{zs}} + \boldsymbol{v}$, this amounts to estimating $\boldsymbol{v} \approx \boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{zs}}$.

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:

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

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:

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

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 (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 $\boldsymbol{h}_{\text{tv}}$:

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

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:

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

A natural question is how this proxy objective $\mathcal{L}_{\text{MSE}}$ is related to 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*

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

We defer the proof of Proposition 5.1 to Appendix A.2. 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 for 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[3] 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:

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

This reformulated problem in equation 14 has a closed-form solution, as described 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 [19]. 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

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

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.

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}.

[3] In Appendix A.3, we empirically compare this linear choice with alternatives, and discuss non-linear extensions.

Table 1: Classification accuracy (%) of *LTV* and four TV methods, tested on eight benchmarks. We also compare with two inference modes (zero-shot and ICL) for reference. *LTV* method achieves the highest average accuracy (Avg.) across all models; check full results on five LLMs in Appendix A.5.

Models	Methods	AGNews	DBPedia	HateSpeech18	MR	SST-2	SST-5	Subj	TREC	Avg.
LLaMA-2-13B [47]	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 [46]	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 [18]	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 [26]	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 [30]	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 [13]	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 [46]	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 [18]	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 [26]	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 [30]	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**

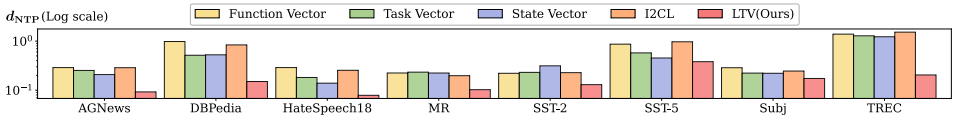

Figure 5: Comparison of d_{NTP} across *LTV* and four baselines on eight benchmarks, tested on LLaMA-3.1-8B. *LTV* consistently achieves the lowest d_{NTP} across all benchmarks.

Table 2: Extraction and inference latency of *LTV* and task vector baselines, evaluated on LLaMA-3.1-8B. Classification accuracy (%) is reported for reference.

	Zero-shot	ICL	Function Vector	I2CL	State Vector	Task Vector	***LTV* (Ours)**
Extraction (s) ↓	-	-	198.965	**1.347**	27.756	27.773	42.664
Inference (s) ↓	0.0220	0.2238	0.0614	0.0246	0.0292	0.0236	**0.0226**
Accuracy (%) ↑	63.48	82.66	66.75	66.42	70.37	70.75	**75.46**

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 [30, 14], we evaluate *LTV* on a diverse set of text classification benchmarks. We consider eight widely used datasets: SST-2, SST-5 [45], MR [40], AGNews, DBPedia [58], TREC [50], SUBJ [39], HateSpeech18 [10]. Unless otherwise specified, 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.1.

Models. We conduct experiments on five widely used LLMs: LLaMA-2-7B [47], LLaMA-2-13B [47], LLaMA-3.1-8B [13], Qwen-2.5-7B [43], and Qwen-3-8B [54].

Baselines. We compare *LTV* with existing task vector methods in a training-free setting, which can be categorized into three groups based on the module from which task vectors are extracted. First, *Task Vector* [18] extracts the task vector from the outputs of a single *decoder layer*. Second, *Function Vector* [46] and *State Vector* [26] extract task vectors from a subset of *attention heads*. Third, *I2CL* [30] extracts task vectors from both attention heads and MLP modules. In I2CL, the task vector is injected after scaling, where the scaling coefficient is *trained* using a validation set with true labels. For fair comparison in a training-free setting, we use the default coefficient selected in [30], instead of training the coefficient. We also compare with standard zero-shot and ICL baselines. Further details on each method are provided in Appendix A.4.

6.2 Results

***LTV* outperforms baselines.** Table 1 reports classification accuracy of *LTV* and baseline task vector methods across eight benchmarks. We present results on two representative models here, with full results on all five LLMs provided in Appendix A.5. As shown in the rightmost column, *LTV* achieves the highest average accuracy on all five models. Notably, *LTV* outperforms the best

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 columns indicating default settings.

	N			λ		
	64	128	256	1.0	5.0	10.0
Avg. Acc. (%) ↑	72.2	74.7	**75.2**	75.2	**75.2**	75.2
d_{NTP} ↓	0.229	0.178	**0.147**	0.149	**0.147**	0.145

Table 4: Regression performance of *LTV* and task vector baselines, measured by mean squared error (MSE), on linear regression and ReLU regression tasks. Experiments are conducted on LLaMA-3.1-8B. Zero-shot and ICL are shown for reference.

	Zero-shot	ICL	Function Vector	I2CL	State Vector	Task Vector	***LTV* (Ours)**
Linear Regression ↓	5.51	3.97	5.23	5.29	5.35	5.97	**5.13**
ReLU Regression ↓	3.82	3.33	3.63	3.68	3.80	4.05	**3.45**

Table 5: Classification accuracy (%) of *LTV* when task vectors obtained from a larger model are transferred and applied to a smaller model for task vector-based inference. We use Qwen-2.5-72B as the larger model and Qwen-2.5-7B as the smaller model. As a reference, we report zero-shot, ICL, and *LTV* inference of each model using its own task vectors. Values in parentheses indicate the change relative to the *LTV* performance of the smaller model.

Models	Methods	AGNews	DBPedia	HateSpeech18	MR	SST-2	SST-5	Subj	TREC	Avg.
Smaller Model (*Qwen-2.5-7B* [43])	Zero-shot	74.20	73.20	61.20	62.20	55.40	20.00	56.80	67.20	58.78
	ICL	83.24 (±2.7)	83.48 (±4.1)	80.60 (±1.3)	92.88 (±0.2)	95.00 (±0.9)	46.60 (±2.9)	76.00 (±6.8)	87.04 (±1.5)	80.61
	LTV	78.48 (±4.8)	77.24 (±1.9)	72.28 (±2.7)	87.56 (±4.1)	89.00 (±7.4)	33.16 (±2.6)	66.52 (±8.4)	84.32 (±4.0)	73.57
Larger Model (*Qwen-2.5-72B* [43])	Zero-shot	72.00	68.00	70.40	92.00	90.60	36.00	53.00	73.00	69.38
	ICL	88.08 (±1.1)	95.64 (±0.6)	82.80 (±1.4)	93.88 (±0.3)	96.44 (±0.2)	46.12 (±1.6)	95.80 (±0.7)	75.44 (±7.1)	84.28
	LTV	81.40 (±1.4)	88.92 (±1.9)	77.20 (±1.6)	91.48 (±0.4)	94.36 (±1.0)	37.16 (±2.8)	91.92 (±1.3)	77.24 (±3.8)	79.96
Smaller Model (*Qwen-2.5-7B* [43])	Transferred *LTV* (72B → 7B)	**83.24** (±1.2) (+4.76)	**92.68** (±1.8) (+15.44)	**76.48** (±1.9) (+4.20)	**91.36** (±0.5) (+3.80)	**92.36** (±1.1) (+3.36)	**42.48** (±2.7) (+9.32)	**90.24** (±1.5) (+23.72)	71.16 (±5.9) (-13.16)	**80.00** (+6.43)

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. A comparison with I2CL using its original validation and tuning protocol across all five models is provided in Appendix A.5.

***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 evaluate the computational efficiency of *LTV* in terms of one-time extraction cost and per-sample inference latency (Table 2). Although *LTV* requires a higher extraction time compared to some TV baselines, this overhead is a one-time pre-computation cost that is effectively amortized over the entire test set. More importantly, *LTV* achieves the most competitive inference latency (0.0226s per sample) among all considered vector-based methods, matching the speed of zero-shot inference. This efficiency stems from the fact that *LTV* operates solely on the final layer through a single matrix-vector multiplication, providing a significant accuracy boost (75.46%) without compromising the model's original inference throughput.

***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.

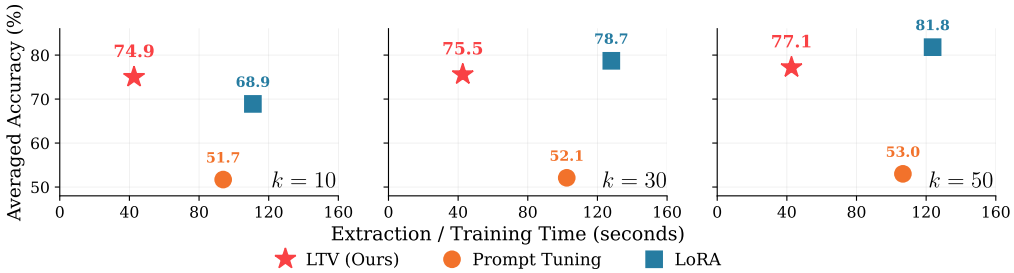

Figure 6: Extraction (training for PEFT) time versus downstream accuracy of *LTV* and PEFT-based methods (prompt tuning [25] and LoRA [20]), evaluated under three numbers of demonstrations ($k = 10, 30, 50$). All methods are tested on LLaMA-3.1-8B.

***LTV* also remains effective on regression tasks.** Most prior studies on task vectors have focused exclusively on classification, leaving it unclear whether existing TV methods are effective in settings with continuous outputs. We address this gap by evaluating TV methods on regression tasks, following the experimental setup of prior studies [55, 15]. Specifically, we conduct linear regression and ReLU [1] regression tasks, and report the mean squared error (MSE) of *LTV* and task vector baselines in Table 4. *LTV* achieves the lowest MSE among all task vector baselines on both tasks (5.13 on linear regression; 3.45 on ReLU regression), showing that our approach is also effective on regression tasks. Detailed setups for the regression experiments are provided in Appendix A.6.

***LTV* can transfer task vectors across model scales.** One might be concerned whether task vector methods remain useful even when the inference model cannot perform ICL effectively. Such cases arise, for instance, when the context length is too short to fit enough demonstrations or when the inference model has limited ICL capability. *LTV* can address this concern through cross-model transfer, which is enabled by a simple extension (see Appendix A.7 for details). This transfer allows us to extract task vectors from a larger, more capable model and apply them to a smaller one. Notably, as shown in Table 5, transferring *LTV* from the 72B model to the 7B model improves the average classification accuracy of the smaller model by 6.43%, even matching the accuracy of the 72B model (79.96%). The change in accuracy after transfer tends to follow the difference in the two models' accuracy under ICL. For example, on Subj and DBPedia, the larger model outperforms the smaller model by the largest margins under ICL, and transfer produces the largest accuracy improvements. On TREC, in contrast, the larger model is less accurate under ICL than the smaller model (75.44% vs. 87.04%), and transfer decreases accuracy. Overall, these results suggest that *LTV* remains useful even when the inference model itself cannot perform ICL effectively, since task vectors can be extracted from a more capable model and transferred to it.

***LTV* achieves higher accuracy than PEFT-based approaches at lower cost in the few-shot regime.** We further examine alternative ways of minimizing our proposed surrogate $\mathcal{L}_{\text{MSE}}$ by employing parameter-efficient fine-tuning (PEFT) methods, instead of the closed-form linear mapping used in *LTV*. Specifically, we consider Prompt Tuning [25] and LoRA [20], two widely used PEFT approaches, and train them to minimize the same $\mathcal{L}_{\text{MSE}}$ objective in equation 11. We evaluate all methods under three demonstration budgets ($k = 10, 30, 50$), and report the extraction (training for PEFT) time and the classification accuracy averaged across 8 benchmarks in Fig. 6. In the few-shot regime ($k = 10$), *LTV* outperforms LoRA in both accuracy and time efficiency. When more demonstrations are available ($k = 30, 50$), LoRA achieves higher accuracy than *LTV*, but its training time is roughly 3 times *LTV*'s extraction time. These results indicate that our method provides a competitive edge compared to PEFT-based methods, particularly in the low-data regime.

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 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 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 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.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00345351, No. RS-2024-00408003); by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2025-02283048, Developing the Next-Generation General AI with Reliability, Ethics, and Adaptability); by the IITP under the Leading Generative AI Human Resources Development (IITP-2026-RS-2026-25544647) grant funded by the Korea government (MSIT); and by the National Institute for Mathematical Sciences (NIMS) grant funded by the Korea government (MSIT) (No. S26120000).

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[2] Rishabh Agarwal, Avi Singh, Lei Zhang, Bernd Bohnet, Luis Rosias, Stephanie Chan, Biao Zhang, Ankesh Anand, Zaheer Abbas, Azade Nova, et al. Many-shot in-context learning. *Advances in Neural Information Processing Systems*, 37:76930–76966, 2024.

[3] Kwangjun Ahn, Xiang Cheng, Hadi Daneshmand, and Suvrit Sra. Transformers learn to implement preconditioned gradient descent for in-context learning. *Advances in Neural Information Processing Systems*, 36:45614–45650, 2023.

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

[5] Robert Belanec, Simon Ostermann, Ivan Srba, and Maria Bielikova. Task prompt vectors: Effective initialization through multi-task soft prompt transfer. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 77–94. Springer, 2025.

[6] Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, et al. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the ninth workshop on statistical machine translation*, pages 12–58, 2014.

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

[8] Ankush Chatterjee, Kedhar Nath Narahari, Meghana Joshi, and Puneet Agrawal. Semeval-2019 task 3: Emocontext contextual emotion detection in text. In *Proceedings of the 13th international workshop on semantic evaluation*, pages 39–48, 2019.

[9] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems, 2021. *URL https://arxiv. org/abs/2110.14168*, 9, 2021.

[10] Ona De Gibert, Naiara Perez, Aitor García-Pablos, and Montse Cuadros. Hate speech dataset from a white supremacy forum. *arXiv preprint arXiv:1809.04444*, 2018.

[11] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, et al. A survey on in-context learning. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pages 1107–1128, 2024.

[12] Yuxin Dong, Jiachen Jiang, Zhihui Zhu, and Xia Ning. Understanding task vectors in in-context learning: Emergence, functionality, and limitations. *arXiv preprint arXiv:2506.09048*, 2025.

[13] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407, 2024.

[14] Boyan Gao, Xin Wang, Yibo Yang, and David A. Clifton. 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.

[15] Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in neural information processing systems*, 35:30583–30598, 2022.

[16] Shahriar Golchin, Yanfei Chen, Rujun Han, Manan Gandhi, Tianli Yu, Swaroop Mishra, Mihai Surdeanu, Rishabh Agarwal, Chen-Yu Lee, and Tomas Pfister. Towards compute-optimal many-shot in-context learning. In *Second Conference on Language Modeling*, 2025.

[17] Seungwook Han, Jinyeop Song, Jeff Gore, and Pulkit Agrawal. 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.

[18] Roee Hendel, Mor Geva, and Amir Globerson. In-context learning creates task vectors. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[19] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[20] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

[21] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*, 2023.

[22] Josip Jukić and Jan Šnajder. Disentangling latent shifts of in-context learning with weak supervision. *arXiv preprint arXiv:2410.01508*, 2024.

[23] Joonseong Kang, Soojeong Lee, Subeen Park, Sumin Park, Taero Kim, Jihee Kim, Ryunyi Lee, and Kyungwoo Song. Adaptive task vectors for large language models. *arXiv preprint arXiv:2506.03426*, 2025.

[24] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[25] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pages 3045–3059, 2021.

[26] Dongfang Li, Zhenyu Liu, Xinshuo Hu, Zetian Sun, Baotian Hu, and Min Zhang. In-context learning state vector with inner and momentum optimization. *Advances in Neural Information Processing Systems*, 37:7797–7820, 2024.

[27] Hongkang Li, Yihua Zhang, Shuai Zhang, Meng Wang, Sijia Liu, and Pin-Yu Chen. When is task vector provably effective for model editing? a generalization analysis of nonlinear transformers. *arXiv preprint arXiv:2504.10957*, 2025.

[28] Jiaqian Li, Yanshu Li, Ligong Han, Ruixiang Tang, and Wenya Wang. Towards generalizable implicit in-context learning with attention routing. *arXiv preprint arXiv:2509.22854*, 2025.

[29] Yingcong Li, Muhammed Emrullah Ildiz, Dimitris Papailiopoulos, and Samet Oymak. Transformers as algorithms: Generalization and stability in in-context learning. In *International conference on machine learning*, pages 19565–19594. PMLR, 2023.

[30] Zhuowei Li, Zihao Xu, Ligong Han, Yunhe Gao, Song Wen, Di Liu, Hao Wang, and Dimitris N. Metaxas. Implicit in-context learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=G7u4ue6ncT.

[31] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[32] Sheng Liu, Haotian Ye, Lei Xing, and James Y Zou. In-context vectors: Making in context learning more effective and controllable through latent space steering. In *Forty-first International Conference on Machine Learning*, 2024.

[33] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[34] Sarthak Mittal, Eric Elmoznino, Leo Gagnon, Sangnie Bhardwaj, Guillaume Lajoie, and Dhanya Sridhar. Does learning the right latent variables necessarily improve in-context learning? In *Forty-second International Conference on Machine Learning*, 2025.

[35] Jesse Mu, Xiang Li, and Noah Goodman. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36:19327–19352, 2023.

[36] Shashi Narayan, Shay B Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 1797–1807, 2018.

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

[38] Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. Task arithmetic in the tangent space: Improved editing of pre-trained models. *Advances in Neural Information Processing Systems*, 36:66727–66754, 2023.

[39] Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. *arXiv preprint cs/0409058*, 2004.

[40] Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. *arXiv preprint cs/0506075*, 2005.

[41] Madhur Panwar, Kabir Ahuja, and Navin Goyal. In-context learning through the bayesian prism. *arXiv preprint arXiv:2306.04891*, 2023.

[42] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[43] A Yang Qwen, Baosong Yang, B Zhang, B Hui, B Zheng, B Yu, Chengpeng Li, D Liu, F Huang, H Wei, et al. Qwen2. 5 technical report. *arXiv preprint*, 2024.

[44] Baturay Saglam, Xinyang Hu, Zhuoran Yang, Dionysis Kalogerias, and Amin Karbasi. Learning task representations from in-context learning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6634–6663, 2025.

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

[46] Eric Todd, Millicent Li, Arnab Sen Sharma, Aaron Mueller, Byron C Wallace, and David Bau. Function vectors in large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

[47] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

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

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

[50] Ellen M Voorhees and Dawn M Tice. Building a question answering test collection. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 200–207, 2000.

[51] Futing Wang, Jianhao Yan, Yue Zhang, and Tao Lin. Elicit: Llm augmentation via external in-context capability. In *The Thirteenth International Conference on Learning Representations*, 2025.

[52] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[53] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*, 2021.

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

[55] Liu Yang, Ziqian Lin, Kangwook Lee, Dimitris Papailiopoulos, and Robert D Nowak. Task vectors in in-context learning: Emergence, formation, and benefit. *CoRR*, 2025.

[56] Kayo Yin and Jacob Steinhardt. Which attention heads matter for in-context learning? *arXiv preprint arXiv:2502.14010*, 2025.

[57] Frederic Z Zhang, Paul Albert, Cristian Rodriguez-Opazo, Anton van den Hengel, and Ehsan Abbasnejad. Knowledge composition using task vectors with learned anisotropic scaling. *Advances in Neural Information Processing Systems*, 37:67319–67354, 2024.

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

[59] Yufeng Zhang, Fengzhuo Zhang, Zhuoran Yang, and Zhaoran Wang. What and how does in-context learning learn? bayesian model averaging, parameterization, and generalization. *arXiv preprint arXiv:2305.19420*, 2023.

[60] Yuxiang Zhou, Jiazheng Li, Yanzheng Xiang, Hanqi Yan, Lin Gui, and Yulan He. 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*, pages 14365–14378, 2024.

A Appendix

Table 6: Overview of the benchmark datasets used in our experiments, including their prompting templates, label sets, and licenses. {Sentence} and {Label} are placeholders for input query x and label y, respectively. All datasets are publicly available and used under their original licenses or standard research-use terms.

Dataset	Prompt Template	Labels	License
AGNews	News: {Sentence} Type: {Label}	World / Sports / Business / Technology	Publicly available for research purposes
DBPedia	Input: {Sentence} Label: {Label}	company / school / artist / athlete / politics / transportation / building / nature / village / animal / plant / album / film / book	CC BY-SA 3.0
HateSpeech18	Text: {Sentence} Label: {Label}	neutral / hate	CC BY-SA 3.0
MR	Review: {Sentence} Sentiment: {Label}	negative / positive	Publicly available for research purposes
SST-2	Review: {Sentence} Sentiment: {Label}	negative / positive	CC BY 4.0
SST-5	Sentence: {Sentence} Sentiment: {Label}	terrible / negative / neutral / positive / great	CC BY 4.0
SUBJ	Sentence: {Sentence} Label: {Label}	objective / subjective	Publicly available for research purposes
TREC	Question: {Sentence} Answer Type: {Label}	Abbreviation / Entity / Person / Location / Number	Publicly available for research purposes

A.1 Implementation Details on Evaluation

We provide detailed information about our evaluation setup. Following experiments conducted in Li et al. [30], we employ the same prompt templates and label sets, as summarized in Table 6. However, EmoC [8] 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. [18], Todd et al. [46], Li et al. [30]. All experiments were conducted using a single NVIDIA GPU (either RTX 6000 Ada Generation or GeForce RTX 4090).

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.2 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 $D_{\mathrm{KL}}(\boldsymbol{p}\|\boldsymbol{q})$.

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,

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

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

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

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

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

This completes the proof. □

Table 7: 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. 6and our proposed d_{NTP} metric. Experiments are conducted on LLaMA-3.1-8B [13].

Design Choices	$f(x, Z)$	Avg. Acc. ↑	d_{NTP} ↓
Constant Mapping	$\boldsymbol{c}$	55.72	0.694
Linear Mapping (*LTV*)	$\boldsymbol{W}\boldsymbol{h}_{\text{zs}}(x)$	75.21	0.147
2-layer MLP based Mapping	$\boldsymbol{W}_2\sigma_{\text{ReLU}}(\boldsymbol{W}_1\boldsymbol{h}_{\text{zs}}(x))$	77.74	0.132

A.3 Design Choices for the Task Vector

In Sec. 5.1 of the main body, 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

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

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

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

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

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 three mapping designs within our proposed framework: a constant mapping, a linear mapping, 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 $\boldsymbol{h}_{\text{icl}} - \boldsymbol{h}_{\text{zs}}$, defined as

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

  where N unlabeled train queries $\{x_i\}_{i=1}^N$ are used.
- **Linear Mapping (*LTV*):** 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 [1] 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 parameters and σ_{ReLU} denotes the 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 [33] 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 7, the constant mapping achieves only 55.72% accuracy, performing significantly worse than our proposed linear mapping (75.21% accuracy) by a margin of 19.49% in accuracy. This confirms that a query-dependent mapping is essential for approximating the effect of demonstrations, as a fixed hidden state vector cannot capture how this effect varies across queries.

On the other hand, the 2-layer MLP-based mapping improves accuracy to 77.74%, indicating that more expressive mappings can yield additional gains within our framework. This observation suggests that our optimization formulation in equation 12 naturally extends beyond the linear case. In this paper, we focus on the linear mapping as the simplest instantiation that admits an efficient closed-form

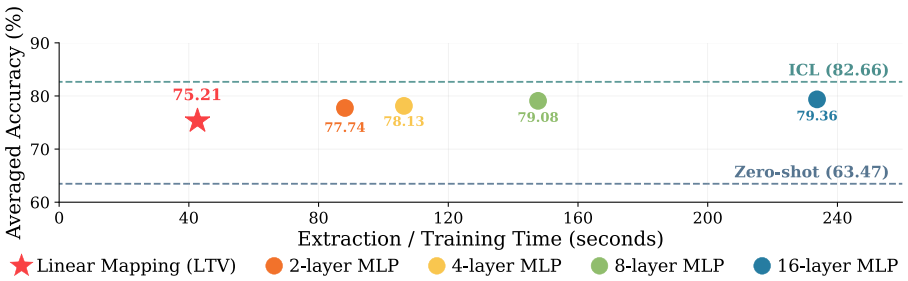

Figure 7: Extraction (training for MLP) time versus downstream accuracy of *LTV* and MLP-based mappings with varying depth. All methods are tested on LLaMA-3.1-8B, and accuracy is averaged over eight classification benchmarks used in the main paper in Sec. 6.

solution and preserves the training-free nature of ICL, and leave non-linear extensions that require iterative optimization for future work. Below, we provide a preliminary cost-benefit analysis of such non-linear mappings when training is allowed, to characterize the trade-off between expressiveness and extraction cost.

Further discussion on scaling of the non-linear mappings. To further explore how non-linear mappings scale with model capacity, we train deeper MLP-based mappings (4, 8, and 16 layers), each with skip connections every two layers to preserve the residual stream. Fig. 7 compares *LTV* and these MLP-based variants in terms of both extraction (training for MLP) time cost and classification accuracy. While deeper MLPs achieve higher absolute accuracy (up to 79.36% with 16 layers), the gain comes with substantially increased extraction cost: a 16-layer MLP requires roughly $5.5\times$ the extraction time of *LTV* for 4.15% absolute accuracy improvement.

A.4 Implementation 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. Note that these baselines typically require additional *labeled* examples for layer or head selection (*e.g.,* Task Vector, Function Vector, and State Vector use a labeled validation set of 32 examples), whereas our method only uses *unlabeled* queries.

- **Task Vector** [18]. 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** [46]. 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** [26]. 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, 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** [30]. 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`.

Table 8: Complete results for Table 1 in the main paper. We report the classification accuracy (%) of LTV and four baseline task vector methods on eight benchmarks across all five LLMs evaluated in this work. The accuracy of Zero-shot and ICL inference is provided as reference inference modes. *LTV* consistently achieves the highest average accuracy (Avg.) across all models.

Models	Methods	AGNews	DBPedia	HateSpeech18	MR	SST-2	SST-5	Subj	TREC	Avg.
Qwen-2.5-7B [43]	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 [46]	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 [18]	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 [26]	**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 [30]	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 [54]	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 [46]	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 [18]	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 [26]	**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 [30]	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 [47]	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 [46]	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 [18]	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 [26]	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 [30]	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 [47]	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 [46]	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 [18]	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 [26]	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 [30]	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 [13]	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 [46]	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 [18]	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 [26]	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 [30]	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**

Table 9: Classification accuracy (%) of *LTV* and I2CL [30] evaluated using its original validation and tuning protocol, tested on eight benchmarks across five models. Avg. reports the average accuracy across benchmarks.

Models	Methods	AGNews	DBPedia	HateSpeech18	MR	SST-2	SST-5	Subj	TREC	Avg.
Qwen-2.5-7B [43]	I2CL (Optimized) [30]	**83.12** (±2.9)	83.08 (±1.5)	53.92 (±3.4)	87.88 (±1.2)	91.04 (±1.7)	**34.96** (±4.1)	50.52 (±1.1)	65.56 (±9.5)	68.76
	***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 [54]	I2CL (Optimized) [30]	**81.48** (±2.9)	87.48 (±0.3)	73.60 (±2.4)	**88.68** (±1.3)	**90.76** (±1.6)	**40.00** (±2.3)	58.72 (±8.6)	71.44 (±8.6)	72.10
	***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 [47]	I2CL (Optimized) [30]	82.88 (±3.3)	**89.60** (±2.4)	**72.04** (±2.4)	90.40 (±1.7)	88.40 (±1.7)	**46.24** (±2.2)	**81.64** (±3.1)	66.76 (±8.6)	**77.25**
	***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 [47]	I2CL (Optimized) [30]	85.80 (±4.2)	91.92 (±0.8)	**75.27** (±3.4)	**90.35** (±1.6)	**92.24** (±1.1)	**46.72** (±1.7)	**86.76** (±2.4)	73.48 (±9.0)	**80.46**
	***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 [13]	I2CL (Optimized) [30]	**84.60** (±2.5)	84.60 (±4.2)	50.08 (±0.2)	87.52 (±1.4)	87.92 (±0.4)	**38.80** (±4.8)	**70.80** (±2.9)	70.28 (±9.1)	71.83
	***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**

A.5 Further Experiments Related to Table 1 in Section 6.2

Extended results of Table 1. Table 8 provides the complete results across all five LLMs evaluated in this work. Consistent with the trend reported in the main body, *LTV* achieves the highest average accuracy on all models, demonstrating that its effectiveness generalizes across different model families and scales.

Comparison with I2CL under Its Original Validation and Tuning Protocol. In the main experiments, we use the default I2CL scaling coefficient to compare task vector methods without gradient-based training. Here, we additionally evaluate I2CL using the validation and tuning protocol described in its original paper [30], while keeping $N = 256$ and $\lambda = 5$ fixed for *LTV*. Table 9 reports the comparison across the five models and eight classification benchmarks. *LTV* achieves higher average accuracy on Qwen-2.5-7B, Qwen-3-8B, and LLaMA-3.1-8B, whereas I2CL achieves higher average accuracy on LLaMA-2-7B and LLaMA-2-13B.

Table 10: Classification accuracy (%) on LLaMA-3.1-8B when all task vector methods use the same 256 training queries. *LTV* does not use query labels, whereas the other task vector methods use the corresponding labels. Zero-shot and ICL are included for reference.

Methods	AGNews	DBPedia	HateSpeech18	MR	SST-2	SST-5	Subj	TREC	Avg.
Zero-shot	75.00	69.00	60.60	82.20	86.80	25.60	59.20	49.40	63.47
ICL	87.16 (±1.1)	97.68 (±0.7)	74.32 (±7.2)	94.52 (±0.4)	94.20 (±1.0)	48.32 (±1.0)	85.96 (±4.7)	79.08 (±6.5)	82.66
Function Vector [46]	76.48 (±0.8)	70.20 (±0.3)	64.76 (±0.9)	82.76 (±0.5)	87.44 (±0.4)	25.60 (±0.9)	59.80 (±0.4)	68.92 (±0.7)	67.00
Task Vector [18]	82.84 (±3.4)	83.76 (±0.7)	**71.16** (±2.2)	87.00 (±3.0)	88.32 (±1.5)	35.36 (±2.1)	61.20 (±0.5)	68.92 (±0.9)	72.32
State Vector [26]	**82.96** (±2.1)	80.96 (±1.5)	67.40 (±2.4)	**90.72** (±0.7)	85.84 (±2.1)	36.44 (±1.4)	**68.68** (±0.3)	67.40 (±1.8)	72.55
I2CL [30]	76.76 (±0.3)	72.60 (±0.3)	62.24 (±0.4)	85.24 (±0.2)	**90.80** (±0.2)	32.48 (±0.3)	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**

Table 11: Extended results of Table 1 on LLaMA-3.1-8B under the number of demonstrations $k = 10$ setting. The experimental setup is otherwise identical to Table 1.

Methods	AGNews	DBpedia	HateSpeech18	MR	SST-2	SST-5	Subj	TREC	Avg.
Zero-shot	75.00	69.00	60.60	82.20	86.80	25.60	59.20	49.40	63.48
ICL	87.52	96.44	73.00	94.64	94.32	46.20	81.84	80.20	81.77
Function Vector [46]	75.72	69.80	63.80	82.04	86.80	25.56	59.56	67.40	66.34
Task Vector [18]	80.16	81.84	**68.68**	84.24	87.08	31.68	59.64	66.32	69.95
State Vector [26]	**84.28**	80.80	67.24	**90.80**	79.16	33.48	67.20	60.64	70.45
I2CL [30]	76.64	72.44	62.00	85.20	**90.64**	32.12	61.52	48.48	66.13
LTV (Ours)	82.20	**90.04**	64.44	89.44	88.92	**41.60**	**69.60**	**72.92**	**74.89**

Table 12: Extended results of Table 1 on LLaMA-3.1-8B under the number of demonstrations $k = 50$ setting. The experimental setup is otherwise identical to Table 1.

Methods	AGNews	DBPedia	HateSpeech18	MR	SST-2	SST-5	Subj	TREC	Avg.
Zero-shot	75.00	69.00	60.60	82.20	86.80	25.60	59.20	49.40	63.48
ICL	87.84	97.24	78.92	94.60	94.52	50.28	90.68	85.68	84.97
Function Vector [46]	76.56	69.92	63.56	83.00	87.48	26.16	59.48	67.08	66.66
Task Vector [18]	77.60	82.36	65.60	81.72	88.24	32.24	59.44	67.60	69.35
State Vector [26]	83.56	81.12	65.60	84.28	79.04	33.64	61.48	67.72	69.56
I2CL [30]	77.16	72.84	62.32	85.52	**90.64**	32.44	61.96	49.40	66.53
LTV (Ours)	**83.92**	**92.44**	**74.52**	**87.92**	89.80	**41.80**	**69.88**	**76.72**	**77.12**

Comparison using the same training queries. In our main experiments, each task vector method uses a different number of training queries, following its original setup, as described in Section 6.1. One might wonder whether the performance advantage of *LTV* persists when all task vector methods are given the same training queries. To examine this, we compare all methods using the same pool of 256 training queries on LLaMA-3.1-8B across the eight classification benchmarks. Importantly, *LTV* does not use the query labels, whereas the other task vector methods use the corresponding labels to construct their task vectors. As shown in Table 10, *LTV* still achieves the highest average accuracy among task vector methods when the training queries are shared.

Effect of the number of demonstrations. To verify the robustness of *LTV* to the number of demonstrations k, we also report results under $k = 10$ (Table 11) and $k = 50$ (Table 12) on LLaMA-3.1-8B. In both settings, *LTV* consistently achieves the highest average accuracy among task vector baselines across both settings, demonstrating its robustness to the number of demonstrations k.

A.6 Implementation Details on Regression Experiments

We provide the detailed experimental setup for the regression experiments reported in Table 4. Following prior work in the in-context learning literature [15, 55], we evaluate *LTV* on synthetic regression tasks. For each task instance, the model is given 30 demonstration pairs $(\boldsymbol{x}, \boldsymbol{y})$ and is asked to predict the continuous scalar output for a query input. We use a simple textual template where each demonstration is formatted as two lines: $x : [x_1, \ldots, x_m]$ followed by $y : y_i$, with demonstrations separated by blank lines.

Each input vector is sampled as $\boldsymbol{x} \sim \mathcal{N}(0, \boldsymbol{I}_m)$ with $m = 6$. We consider two regression settings: linear regression, where $\boldsymbol{y} = \boldsymbol{w}^\top \boldsymbol{x}$ for $\boldsymbol{w} \in \mathbb{R}^d$, and ReLU [1] regression, where $\boldsymbol{y} = \boldsymbol{\alpha}^\top \mathrm{ReLU}(\boldsymbol{V}\boldsymbol{x})$ for $\boldsymbol{V} \in \mathbb{R}^{r \times d}$, $\boldsymbol{\alpha} \in \mathbb{R}^r$, and $r = 100$. For linear regression, the parameter is sampled as $w \sim \mathcal{N}(0, I_d)$, following the isotropic Gaussian prior of Garg et al. [15]. For ReLU regression, we sample $V \sim \mathcal{N}(0, I)$ and $\alpha \sim \mathcal{N}(0, \frac{1}{r} I)$. The performance is measured by mean squared error (MSE).

Unlike classification, where each label is a single token from a discrete label set, the label $\boldsymbol{y}$ here is a real-valued scalar that the LLM must generate as a multi-token sequence (digits and decimal point). To extend *LTV* to this multi-token generation setting, we apply the linear mapping $\boldsymbol{W}^\star$ at every generation step. At each step, the model takes the query and the tokens generated so far as

Table 13: Performance on generation tasks with LLaMA-3.1-8B. Results are averaged over three runs with different demonstration sets.

Methods	Regression positions	Injection positions	WMT14 En–Fr	XSum	GSM8K
Zero-shot	–	–	28.35 (±0.7)	22.89 (±0.1)	22.33 (±1.3)
ICL	–	–	35.68 (±0.7)	36.12 (±0.5)	49.67 (±1.9)
$W_{0,0}$ (*LTV*)	First position	First position	**30.46** (±0.8)	**26.01** (±0.6)	20.44 (±1.0)
$W_{\text{all},0}$	All positions	First position	30.11 (±1.1)	25.81 (±0.5)	20.44 (±1.0)
$W_{\text{all,all}}$	All positions	All positions	29.90 (±0.7)	25.85 (±0.3)	**28.56** (±1.2)
$W_{0,\text{all}}$	First position	All positions	17.17 (±0.6)	20.99 (±0.9)	1.56 (±0.3)

input, and computes the zero-shot hidden state $\boldsymbol{h}_{\text{zs}}$ of the last token. We then add the task vector $\boldsymbol{W}^\star \boldsymbol{h}_{\text{zs}}$ to it before feeding the result to the LM head. The sampled token is appended to the input for the next step. Note that $\boldsymbol{W}^\star$ is estimated once during extraction and reused across all generation steps, preserving the inference efficiency.

A.7 Implementation Details on Cross-Model Transfer Experiments

We provide the detailed experimental setup and method for the cross-model transfer experiments reported in Table 5. The proposed *LTV* method in Sec. 5 is defined within a single model. In practice, however, when the capacity of the model is not enough, ICL alone may not extract sufficient task information from the demonstrations, leading to limited quality of the task vector.

We thus consider transferring task vectors extracted from a larger model – which captures richer ICL representations from the same demonstrations – to a smaller model, in order to improve its performance without requiring additional demonstrations or larger inference cost. This transfer is non-trivial because the two models have different hidden state dimensions and different LM heads, so the linear mapping $\boldsymbol{W}^\star$ in Sec. 5 cannot be directly reused.

Method. We extend *LTV* to the cross-model setting with a single modification. We replace the hidden-space regression in equation 15 with a logit-space regression. The intuition is that, while hidden state spaces differ across models, two models that share the same tokenizer share the vocabulary as a common coordinate system, making the logit space a natural target for alignment. To be specific, let $\boldsymbol{z}^L_{\text{icl}} \coloneqq \boldsymbol{W}^L_{\text{lm}} \boldsymbol{h}^L_{\text{icl}}$ denote the logit produced by the larger model under ICL, and $\boldsymbol{z}^S_{\text{zs}} \coloneqq \boldsymbol{W}^S_{\text{lm}} \boldsymbol{h}^S_{\text{zs}}$ the logit produced by the smaller model under zero-shot inference, where the superscripts L and S denote quantities from the larger and smaller model, respectively.

We solve a ridge regression analogous to equation 15, mapping $\boldsymbol{h}^S_{\text{zs}}$ to the logit-space difference $\boldsymbol{z}^L_{\text{icl}} - \boldsymbol{z}^S_{\text{zs}}$, which yields a closed-form solution $\tilde{\boldsymbol{W}}^\star \in \mathbb{R}^{N_{\mathcal{U}} \times d_S}$, where d_S is the hidden dimension of the smaller model and $N_{\mathcal{U}}$ is the vocabulary size. At inference, the corrected logit is $\boldsymbol{z}^S_{\text{zs}}(\boldsymbol{x}_{\text{test}}) + \tilde{\boldsymbol{W}}^\star \boldsymbol{h}^S_{\text{zs}}(\boldsymbol{x}_{\text{test}})$. When the source and target models are identical, this reduces to the original formulation in equation 16 up to the linear LM head transformation, ensuring that our extension preserves the design principle of *LTV*.

Setup. We use Qwen2.5-72B as the larger model and Qwen2.5-7B as the smaller model, which share the same tokenizer. We follow the eight classification benchmarks and prompt templates used in the main experiments in Sec. 6. To target the regime where the smaller model cannot extract sufficient task information from demonstrations alone, we use $k = 10$ demonstrations, compared to $k = 30$ in the main experiments. For DBPedia, we use $k = 14$ since it has 14 classes. We use $N = 256$ unlabeled training queries and $\lambda = 5.0$ for the ridge regression, following the default values in the main experiments. We report the average accuracy over five independent runs.

A.8 Additional Experiment on Generation Tasks

We explore extending *LTV* from single-token classification to multi-token generation. Recall that, in the classification setting, *LTV* fits the linear mapping $\boldsymbol{W}$ using the final-layer hidden states at the position immediately before the first output token is predicted. At inference, it adds $\boldsymbol{W}\boldsymbol{h}_{\text{zs}}$ to the hidden state at this position before computing the predictive distribution. Thus, both regression and task vector injection concern the first output-token prediction.

For generation, the model must predict a sequence of tokens over the full vocabulary. Aligning its predictive distribution with ICL therefore requires considering subsequent decoding steps as well. To extend our proposed method to generation tasks, we consider two dimensions: the positions whose hidden states are used to fit $\boldsymbol{W}$, and the decoding steps at which the task vector is injected.

We denote the resulting variants by $W_{\mathrm{fit},\mathrm{inject}}$, where 0 indicates only the first output-token prediction and all indicates every output-token prediction. This gives four variants:

- $W_{0,0}$**: Regression and injection only for the first output token.** We fit $\boldsymbol{W}$ using hidden states at the position preceding the first output token and inject the task vector only before predicting that token. Subsequent tokens are generated without further task vector injection. This applies the original classification procedure to generation.
- $W_{0,\mathrm{all}}$**: Regression for the first output token and injection at every decoding step.** We fit $\boldsymbol{W}$ using hidden states at the position preceding the first output token, but apply the mapping to the current hidden state before each output-token prediction. This extends only the injection positions.
- $W_{\mathrm{all},0}$**: Regression over all output positions and injection only for the first output token.** We fit $\boldsymbol{W}$ using hidden states from all decoding positions of the ICL generation, but inject the task vector only before predicting the first output token. This extends only the regression positions.
- $W_{\mathrm{all},\mathrm{all}}$**: Regression over all output positions and injection at every decoding step.** We fit $\boldsymbol{W}$ using hidden states from all decoding positions of the ICL generation and apply the mapping to the current hidden state before each output-token prediction. This extends both the regression and injection positions.

In all four variants, $\boldsymbol{W}$ is obtained once through a single ridge regression. When injection is applied at every decoding step, the fitted mapping is reused to compute the task vector from the current hidden state. Table 13 compares these four choices.

We evaluate on LLaMA-3.1-8B using WMT14 English-to-French translation [6], XSum summarization [36], and GSM8K mathematical reasoning [9] with 8-shot chain-of-thought prompting. We use 256 unlabeled training queries to fit $\boldsymbol{W}$ and average the results over three runs with different demonstration sets. We report BLEU [42] for translation, ROUGE-1 [31] for summarization, and accuracy (%) for mathematical reasoning.

As shown in Table 13, fitting and injecting only at the first position improves translation and summarization over zero-shot inference. Mathematical reasoning instead benefits from fitting and injecting at all positions. Fitting only at the first position and injecting at every decoding step performs worse than zero-shot inference on all three tasks. Thus, the choice of regression and injection positions matters for generation, and none of the four variants matches ICL. Extending *LTV* to longer generation remains a direction for future work.

A.9 Limitation and Future Work

We describe several limitations of our work along with corresponding directions for future work. First, we adopt a linear mapping for *LTV* and empirically validate its effectiveness across a wide range of benchmarks and models. At the same time, our experiments in Appendix A.3 show that more expressive mappings can further improve downstream accuracy, suggesting clear potential beyond the linear case. Therefore, designing more expressive mappings that retain the efficiency of a closed-form solution would thus be a promising direction for future work.

Second, while our work investigates *why* a linear mapping is a reasonable choice and *how* effective it is, the question of *when* the linear approximation is effective remains unexplored. A deeper theoretical characterization of these conditions would not only strengthen the foundation of our work, but also guide the design of mappings tailored to different settings.

Third, our experiments primarily focus on tasks with relatively short outputs, such as single-token classification and scalar regression. Extending *LTV* to inference tasks that generate longer and more diverse token sequences – such as multi-token reasoning, multi-turn conversation, or summarization – would broaden its applicability and is a promising direction for future work. We provide the preliminary generation experiments in Appendix A.8.

Finally, although we demonstrate in Sec. 6.2 that *LTV* enables task vectors to be transferred from a larger model to a smaller one, this transfer currently relies on the assumption that the source and target models share the same tokenizer, which serves as a common coordinate system for the logit-space regression. Extending *LTV* to enable transfer across models with different tokenizers would substantially broaden its applicability and is another valuable direction for future work.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction consistently reflect the paper's key idea (distributional alignment as a design criterion) and clearly present the proposed metric (d_{NTP}) and method (*LTV*). The experimental scope (multiple benchmarks and transfer settings) and claims are consistent with the results presented in Section 6.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses the limitations in Appendix A.9, including the adoption of a linear mapping and the focus on short-output tasks.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The assumptions are clearly stated in Proposition 5.1, and a complete proof is provided in Appendix A.2, with all necessary steps to ensure correctness.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper provides sufficient details to reproduce the experimental results. The datasets (eight classification and two regression benchmarks), models, and experimental setup are described in Section 6. Additional implementation details (prompts, baselines, and hyperparameters) are provided in Appendix A.1–A.7. Code is also provided, enabling reproducibility of the main results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code is publicly available at this GitHub repository, as also referenced in Section 6. The experiments use standard benchmark datasets. The repository provides sufficient instructions to reproduce the main results, including task vector extraction and inference for classification and regression tasks, as well as transfer experiments.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Experimental settings, including number of demonstrations, evaluation setup, run configurations, and data sources are detailed in Section 6.1 and Appendix A.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Table 1 reports results as mean and standard deviation over 5 runs with different random seeds, where the variability arises from the random selection of demonstrations and evaluation. This provides information about the statistical variability of the reported results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper specifies the hardware used (a single NVIDIA RTX 6000 Ada Generation or NVIDIA GeForce RTX 4090) in Appendix A.1, and reports extraction and inference time in Table 2, enabling for estimation of computational latency.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research involves evaluation on standard benchmarks, without human subjects or sensitive data, and thus conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [N/A]

Justification: The paper primarily focuses on a general methodological contribution without a specific application, and therefore does not have identifiable societal impacts.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper proposes a general methodological contribution and uses standard benchmark datasets, which do not pose a high risk for misuse, and therefore no specific safeguards are required.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper uses standard benchmark datasets and pretrained models that are widely adopted in the literature. All assets are properly cited and are used under their original licenses or standard research-use terms. Dataset details are summarized in Table 6.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The implementation of the proposed method will be released via a public repository. Documentation will be provided together with the code.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The research does not involve any crowdsourcing experiments or studies involving human participants.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The research does not involve research with human subjects, and therefore IRB approval is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are not a core methodological component and are used as standard pretrained models for evaluation.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.