Adaptive Task Vectors for Large Language Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

In-Context Learning (ICL) enables Large Language Models (LLMs) to perform tasks without parameter updates by conditioning on a few demonstrations provided in the prompt. Despite its success, ICL suffers from several limitations, including sensitivity to demonstration order, context length constraints, and limited control over internal reasoning mechanisms. To address these challenges, task vector-based approaches compress task information into a single vector. However, these methods typically construct task vectors from fixed sets of demonstrations and reuse them across input queries, without conditioning on the specific input. This limitation reduces their ability to probe or guide the model's internal computation, making adaptation to diverse or misaligned queries difficult and degrading generalization on unseen tasks. To overcome this limitation, we propose **Adaptive Task Vectors (ATV)**, a simple and effective framework that dynamically generates task vectors conditioned on each input query. ATV employs a small language model to generate task vectors, which are then transformed to match the target LLM's architecture and applied to guide its output generation. In contrast to ICL and previous vector-based approaches, which rely on fixed demonstration sets and their corresponding vectors, ATV dynamically generates task vectors tailored to each specific input query and task. As a result, ATV serves as an effective tool for probing and guiding the internal mechanisms of LLMs, enabling strong performance and enhanced insight, even for unseen tasks. Furthermore, we provide a theoretical analysis indicating that ATV is expressively equivalent to LoRA under equal rank budgets and more expressive than Prefix-Tuning, thereby offering formal support for its representational advantage.

1 Introduction

Large Language Models (LLMs) have made remarkable strides in natural language processing, demonstrating impressive performance across various tasks. As LLMs become more powerful, understanding and controlling their internal mechanisms is increasingly important. In-Context Learning (ICL) [1] has become a pivotal method for enhancing LLM performance, enabling models to perform specific tasks by including demonstration samples in prompts without requiring additional training [2].

However, ICL faces several limitations: it operates solely at the input-output level and lacks direct access to or control over the model's internal computation or representations. Performance varies depending on the order and selection of demonstration samples [3–5], the maximum context length constraint of LLMs makes it difficult to handle tasks involving long-context reasoning or diverse demonstration sets, and processing many demonstrations reduces computational efficiency [6, 7].

To mitigate these issues, task vector-based approaches [8–12] have attracted growing interest for improving the efficiency and robustness of ICL. Task vectors [8] are vector representations that compress task-specific information, typically obtained from the hidden state of the last token in

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

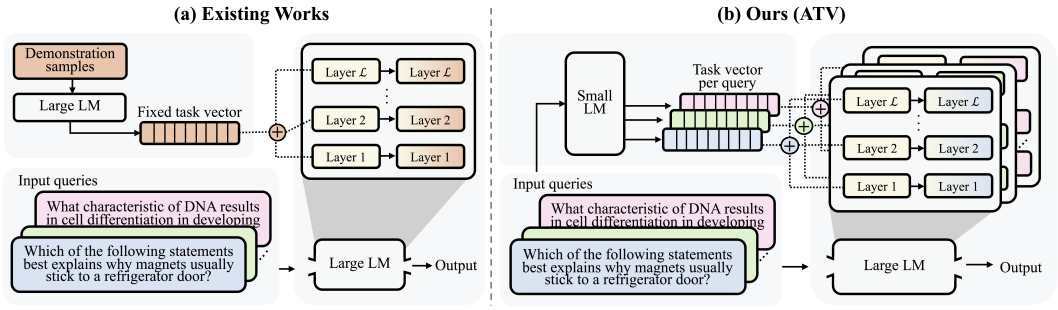

Figure 1: Comparison between task vector methods: a) Prior work uses a fixed task vector for all inputs, whereas b) our method generates a query-specific task vector, enabling adaptive behavior for each input.

the prompt. These vectors are integrated with the input query to modulate the model's output in a task-specific manner. By compressing information from multiple demonstrations into a single vector, these methods overcome context window constraints and reduce performance variability due to demonstration ordering [2, 13–15]. Thus, they preserve the effectiveness of ICL while improving computational efficiency and consistency [16]. However, since these task vectors are static, they provide only limited control over the model's internal mechanisms.

Most prior task vector methods construct vectors from fixed demonstration sets and reuse the same vector for all queries, regardless of their individual characteristics [8, 17]. Some recent approaches retrieve precomputed vectors based on query similarity, but these are not conditioned on the current input and offer only limited ability to steer or manipulate internal representations, which may hinder adaptation for misaligned or unseen tasks.

Motivated by these limitations, we present **Adaptive Task Vectors (ATV)**, a framework for dynamically generating task vectors conditioned on each input query. By producing a query-specific vector, ATV enables more accurate and input-sensitive model guidance. Our method uses a small language model to generate intermediate task representations, which are then transformed to match the target LLM architecture and used to modulate its output. This allows ATV to probe and steer the internal mechanisms of LLMs for each input. Figure 1 contrasts (a) conventional fixed-vector methods [10–12], which apply the same vector to all queries, with (b) our adaptive approach that tailors a vector to each query for more appropriate responses.

In this paper, we establish the effectiveness of ATV through theoretical and empirical evaluation. Theoretically, we prove that ATV is expressively equivalent to LoRA under matched rank budgets and strictly more expressive than Prefix-Tuning, providing a formal basis for its enhanced representational capacity. Empirically, we evaluate ATV on in-domain performance, generalization to unseen tasks, and ablations on model capacity and injection configuration. Across these settings, ATV demonstrates strong in-domain accuracy, generalization, and interpretable insights into model capacity and injection behavior.

Our main contributions are as follows: **(1)** We propose **Adaptive Task Vectors (ATV)**, a simple and effective framework that generates task vectors conditioned on each input query, enabling LLMs to adapt and steer their internal computation and representations in a task-aware manner. **(2)** We provide a theoretical analysis showing that ATV is expressively equivalent to LoRA under equal rank budgets and strictly more expressive than Prefix-Tuning, providing a formal justification for its enhanced capacity to manipulate internal model states. **(3)** We empirically evaluate ATV on both in-domain tasks and generalization to unseen tasks, demonstrating strong performance and interpretable insights into how ATV affects internal model behavior.

2 Related Work

In-Context Learning. In-context learning (ICL) allows LLMs to perform new tasks without parameter updates by conditioning on a few input-output examples in the prompt [1]. While ICL, especially since GPT-3, has shown strong performance across diverse tasks through prompt engineering and model scaling [18–20], it faces key limitations: sensitivity to the selection and order of demonstra-

tions [13, 21], constraints from the maximum context length [6], and high computational cost at inference [7].

Adaptive or retrieval-based ICL methods improve robustness by dynamically selecting examples [22], but they remain reliant on prompt tokens, inheriting context length and explicit demonstration requirements. We instead propose conveying task information through a compact, learned vector, which removes prompt design and length constraints while preserving ICL's adaptability and generalization.

Vector-Based Approaches for Model Steering. Recent work has explored replacing in-context demonstrations with task vectors, which are dense representations encoding task information from a few-shot prompt, typically obtained from the last token's hidden state in a transformer [8]. Methods such as I2CL [11] and ELICIT [12] use either compressed context vectors or retrieved task vectors from a static library to improve efficiency. However, these vectors are fixed for each task and are not conditioned on individual inputs, which limits their adaptability. To our knowledge, no previous approach generates task vectors dynamically for each input. Our framework, ATV, addresses this gap by producing input-conditioned task vectors per query, preserving the efficiency of vector-based methods while enabling the adaptability needed for input-level variation.

3 Methodology

3.1 Background and Preliminaries

Transformer Architecture. The Transformer is a neural architecture based on self-attention and forms the foundation of modern large-scale language models [23]. Each layer consists of a feed-forward network and a self-attention mechanism, which allows tokens to attend to others in the sequence. The self-attention operation is defined as:

$$\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

where Q, K, and V denote the query, key, and value matrices, respectively, and d_k is the dimensionality of the key vectors.

Let h_t^{l-1} be the hidden state of token t at layer $l-1$. A standard Transformer layer updates it as:

$$\tilde{h}_t^l = \text{LayerNorm}\left(h_t^{l-1} + \text{Attn}(Q_t, K, V)\right) \tag{2}$$

$$h_t^l = \text{LayerNorm}\left(\tilde{h}_t^l + \text{MLP}(\tilde{h}_t^l)\right) \tag{3}$$

where $Q_t = W_Q h_t^{l-1}$ and K, V are projections of the previous layer's hidden states. In autoregressive models, the hidden state of the last token h_T^l summarizes the input context and is used for next-token prediction.

Task Vectors. Following prior work [12], we define a *task vector* as the hidden state of the last token at each transformer layer, capturing task-relevant information in a compressed form. Given an input $x = [x_1, x_2, \ldots, x_T]$, the task vector at layer l is:

$$v_{\text{task}}^l = h_T^l \quad \text{(task vector extracted from the last token at layer } l\text{)} \tag{4}$$

To steer the model output in a task-specific direction, we inject the task vector into the hidden state of the last token at each transformer layer. Specifically, for each layer l, the modified hidden state is computed as:

$$\tilde{h}^l = h^l + \lambda v_{\text{task}}^l \tag{5}$$

where h^l denotes the hidden state of the last token at layer l, $v_{\text{task}}^l \in \mathbb{R}^{d_l}$ is the corresponding task vector slice, $\tilde{h}^l$ is the injected version, d_l is the hidden dimensionality at layer l, and λ is a scaling factor controlling the strength of the intervention. For simplicity, we omit the token index and refer to the last token's hidden state simply as h^l. This formulation allows the task vector to modulate the model's behavior in a lightweight and interpretable manner.

Previous methods rely on task vectors extracted from fixed demonstrations, resulting in a static representation shared across inputs. We introduce the **Adaptive Task Vector (ATV)**, which is dynamically generated per input to modulate the model's behavior.

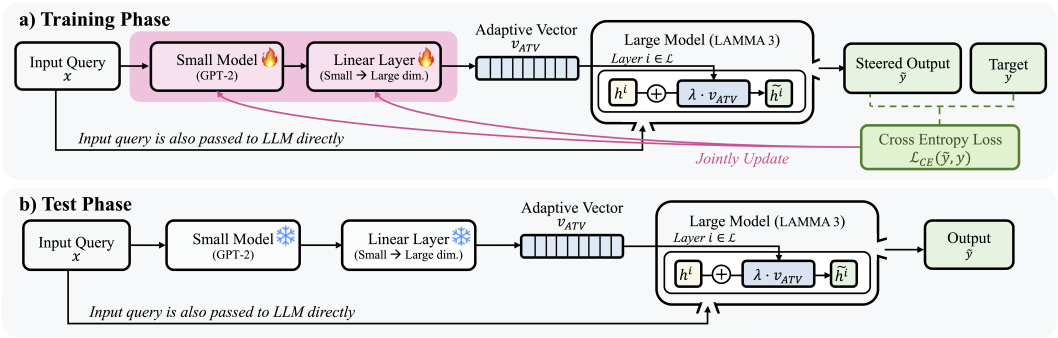

Figure 2: Overview of the Adaptive Task Vector (ATV) framework. Top: During training, the small model and expansion module are updated to minimize the loss from the steered output. Bottom: During inference, both modules are frozen and used to generate query-specific ATV vectors for steering the frozen large model.

3.2 ATV: Adaptive Task Vectors for Large Language Models

Notation and Setup. Let $x = [x_1, x_2, \ldots, x_T]$ be a tokenized input query sequence of length T, and let y denote the corresponding target output. We define two models: a small model $\mathcal{M}_{\text{small}}$ with hidden size d_s, and a large language model $\mathcal{M}_{\text{large}}$ with L layers and hidden size d_l per layer.

From the input x, $\mathcal{M}_{\text{small}}$ produces a hidden representation $v_{\text{small}} \in \mathbb{R}^{d_s}$, extracted from the last token of the last layer. This vector is then expanded via a parameterized function $f_\theta : \mathbb{R}^{d_s} \to \mathbb{R}^{L \times d_l}$ to obtain our **ATV** $v_{\text{ATV}} = f_\theta(v_{\text{small}})$, suitable for injection into the large model. Let $v^l_{\text{ATV}} \in \mathbb{R}^{d_l}$ denote the portion corresponding to the l-th layer. The final output generated by $\mathcal{M}_{\text{large}}$ after ATV injection is denoted $\tilde{y}$. The ATV is scaled by a hyperparameter λ before being added to the hidden state.

Overview of the ATV Framework. Our goal is to steer an LLM without modifying its weights by injecting input-conditioned signals directly into its hidden states. We introduce **ATV**, a lightweight control framework that operates externally to a frozen large model.

ATV consists of two lightweight modules:

(1) **ATV generation.** A small language model produces a compact vector representation from the input query, (2) **ATV expansion.** An expansion module transforms this vector into a set of layer-wise steering signals injected into the large model.

We implement the expansion module as a single linear projection from $\mathbb{R}^{d_s}$ to $\mathbb{R}^{L \cdot d_l}$, followed by reshaping into $\mathbb{R}^{L \times d_l}$ for compatibility with the target model. The generator and expansion modules are trained jointly, while the LLM remains frozen.

By injecting the ATV into the internal layers of the large model, the ATV enables flexible and targeted control over the model's behavior. This allows the large model to better align with desired task objectives, such as answering questions accurately or performing structured reasoning, without modifying its parameters or relying on prompt engineering.

We summarize this process in Figure 2. During training (top), the small model and the expansion module are optimized to produce effective ATVs that steer the large model toward the desired output. During inference (bottom), both modules are frozen and used to dynamically generate ATVs for new input queries. The core idea behind ATV is to adapt the behavior of an LLM to a specific task without modifying its parameters. Instead of prompt-based conditioning, we steer the model by injecting query-specific information directly into its internal hidden states in the form of an ATV.

To generate the ATV, we first encode the input query using a small language model $\mathcal{M}_{\text{small}}$, such as a GPT-2 variant [24]. The model processes the full tokenized sequence and produces hidden representations at each layer. From the last token of the last layer, we extract a compact vector v_{small} that summarizes the semantics of the input query.

This vector is then expanded by f_θ into the **ATV** v_{ATV}, where each slice v^l_{ATV} is designed to modulate computations at the corresponding layer of the large model.

The ATV is injected into the frozen large model by modifying the hidden states of the *last token* during the forward pass. Specifically, for each layer l, the original hidden state h^l of the last token is modified as:

$$\tilde{h}^l = h^l + \lambda v^l_{\text{ATV}} \tag{6}$$

where λ is a scalar hyperparameter that scales the ATV's influence. This additive injection provides lightweight yet effective steering of the model's output behavior.

To enable effective learning, we jointly train $\mathcal{M}_{\text{small}}$ and f_θ using a supervised loss. Let y denote the target output for input query x, and let $\tilde{y}$ be the output of the large model after ATV injection. The objective is to minimize the cross-entropy loss:

$$\min_{\phi,\theta} \ \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\mathcal{L}_{\text{CE}}\left(\tilde{y},\ y\right)\right], \quad \text{where } \tilde{y} = \mathcal{M}_{\text{large}}(x; v_{\text{ATV}}) \tag{7}$$

where $\mathcal{D}$ denotes the training dataset, and ϕ, θ are the parameters of the small model and the expansion module, respectively. Notably, the large model $\mathcal{M}_{\text{large}}$ remains frozen throughout training.

After training, both $\mathcal{M}_{\text{small}}$ and f_θ are frozen. During inference, given a new input query, the system generates a query-specific ATV and injects it into the large model to guide its behavior. This enables ATV to adapt frozen LLMs to diverse downstream tasks in a modular and parameter-efficient manner.

3.3 Theoretical Analysis

We now theoretically analyze the proposed ATV framework to better understand its effect on the behavior of the LLM by comparing it to two prominent parameter-efficient tuning methods: LoRA and Prefix-Tuning. LoRA (Low-Rank Adaptation) injects trainable low-rank matrices into pretrained weights and has demonstrated strong performance in language model fine-tuning [25]. Prefix-Tuning prepends trainable continuous vectors to the input sequence, conditioning the model through modified attention mechanisms without altering the pretrained weights [26].

Specifically, we focus on addressing the following two questions: (1) How does ATV compare to LoRA in expressivity under the same rank budget, and when might ATV offer additional advantages? (2) Does ATV offer a more expressive attention mechanism compared to Prefix-Tuning?

We first show that ATV is never weaker than LoRA when both methods operate under the same rank budget, and then clarify in which respects ATV can be stronger.

Theorem 1. *(ATV-LoRA equivalence under equal rank) Let $h^\ell \in \mathbb{R}^{d_\ell}$ be the last-token hidden state at layer ℓ, and let $\tilde{h}^\ell$, $\hat{h}^\ell$ be the outputs of ATV and LoRA. For $d_s \ll d_\ell$ and a LoRA rank $r = d_s$, ATV and LoRA are expressively equivalent: for any ATV update $\tilde{h}^\ell$, there exists a LoRA configuration yielding $\hat{h}^\ell = \tilde{h}^\ell$, and vice versa (see Appendix A.1 for the full proof).*

Situations in which ATV exceeds LoRA The equivalence theorem 1 ensures that ATV inherits LoRA's expressiveness. Beyond this, there are several aspects in which ATV can exceed LoRA. First, if the auxiliary bottleneck is widened ($d_s > r$), ATV can represent update directions that LoRA cannot capture within its fixed rank-r subspace, thereby increasing its expressive power and enhancing its flexibility to adapt across diverse tasks and input distributions. Second, even with $d_s = r$, ATV enables effective direction injection as it induces input-dependent perturbations. In the ATV-to-LoRA construction, the matrix that plays the role of LoRA's down-projection matrix W_{down}, a factor that remains static during inference, is given by $M := x^{+\top}(\lambda v_{small})$, which, unlike LoRA, varies with the current activation x. Here, λ is a scalar hyperparmeter of ATV and $x^{+\top}$ denotes the Moore-Penrose pseudoinverse of $x^\top$ (see Step 3 of the proof in Appendix A.1.3). This query dependence allows ATV to adapt its update on a per-instance basis, an ability LoRA lacks, which may improve adaptability in dynamic or multi-task environments.

Secondly, under the relaxed linear attention approximation, we argue in Theorem 2 that any representation obtainable by prefix-tuning is also realizable by ATV, while the converse does not hold. To examine the source of expressivity differences under the approximation proposed by prior work [27], we begin by formulating the standard attention as $\text{Attn}(xW_q, CW_k, CW_v)$, where $x \in \mathbb{R}^{T\times d_l}$ is the query, $C \in \mathbb{R}^{m\times d_l}$ is the length-m context, and W_q, W_k, W_v are projection matrices. Prefix-tuning modifies the key and value by concatenating p trainable prefix vectors $P_k, P_v \in \mathbb{R}^{p\times d_l}$, yielding an augmented attention [28]:$\text{Attn}_{\text{prefix}} = \text{Attn}(xW_q, [P_k; CW_k], [P_v; CW_v])$, ATV, in

contrast, injects a trained vector v^l_{ATV} additively to both the query and context: $\text{Attn}_{\text{ATV}} = \text{Attn}\left((x + e_T \cdot (v^l_{ATV})^\top)W_q, (C + e_m \cdot (v^l_{ATV})^\top)W_k, (C + e_m \cdot (v^l_{ATV})^\top)W_v\right)$, where $e_m$is a vector $[0, ...0, 1] \in \mathbb{R}^{m\times 1}$.

Theorem 2. *(ATV is more expressive than Prefix-Tuning) Let $Attn_{ATV}$ and $Attn_{prefix}$ denote the attention outputs from ATV and prefix-tuning, respectively. Then, the representational space $\mathcal{F}$ of $Attn_{ATV}$ includes that of $Attn_{prefix}$:*

$$\mathcal{F}(Attn_{prefix}) \subseteq \mathcal{F}(Attn_{ATV}) \tag{8}$$

Comparison of Attn_{ATV} and $\text{Attn}_{\text{prefix}}$ Under the approximation, $\text{Attn}_{\text{ATV}} \approx \text{Attn}_{\text{prefix}} + \Delta_{\text{cross}}$, where Δ_{cross} encapsulates the six cross-terms that capture the additional interactions between the query and context, modulated by the vector v^l_{ATV} (see Appendix A.2 for the full proof).

4 Experiments

To evaluate ATV, we design experiments examining both in-domain and generalization performance, followed by ablation studies on injection strategies. We also compare ATV with parameter-efficient tuning methods to validate our theoretical analysis and visualize task vector distributions to understand their representational properties. We closely follow ELICIT's [12] experimental design, using identical datasets and evaluation protocols. We evaluate on LLaMA3-8B [29] and Mistral-7B [30], with I2CL [11] as an additional baseline and a separate comparison to LoRA [25] for theoretical validation. Our code is available at `https://anonymous.4open.science/r/ATV-8B5A`.

4.1 Experiment Setup

Models and Baselines. Our primary models are LLaMA3-8B and Mistral-7B. We compare ATV against (i) zero-shot, (ii) 16-shot in-context learning (ICL), (iii) 16-shot BM25 retrieval-based ICL [31], (iv) ELICIT [12], and (v) I2CL [11]. ICL and BM25 baselines use 16 demonstrations, either randomly sampled or retrieved from the task's training set.

Datasets. We evaluate ATV on a diverse collection of 20 tasks spanning five categories: Knowledge, Reasoning, Mathematics, Safety, and Natural Language Understanding. These tasks test various NLP capabilities from reasoning to numerical problem solving. In addition to in-domain tasks, we evaluate ATV on a separate set of unseen tasks to assess its generalization ability.

Evaluation. We adopt the same evaluation strategy as ELICIT to reflect realistic inference scenarios, where test-time query formats differ from those seen during training. Each test query is presented in two additional template variations not used in task vector generation. Task-specific instructions are prepended for all methods to ensure fair comparison. Full implementation and experimental details, including dataset names, template formats, and hyperparameters, are available in Appendix B.

For the ATV generator $\mathcal{M}_{\text{small}}$, we use GPT-2 (137M). As detailed in Appendix C, our additional analysis indicate that this configuration is sufficient for our purposes, with larger or differently trained generators yielding comparable results.

4.2 In-Domain Performance Evaluation

We evaluate ATV on 20 in-domain tasks across five categories, with results summarized in Table 1. ATV consistently achieves the highest average accuracy across all baselines while maintaining strong token efficiency by avoiding additional prompt tokens. ATV performs particularly well on NLU and Reasoning tasks across both LLaMA3 and Mistral, highlighting the benefit of query-specific task vectors in handling semantic and logical variation. These categories often require nuanced understanding of input structure and are sensitive to prompt formulation, limiting the adaptability of fixed vector approaches. Interestingly, BM25 achieves the best result in the Math category. We attribute this to the pattern-based nature of many math problems, where retrieved demonstrations closely resembling the test query provide a direct advantage. In contrast, ATV's focus on semantic-level task modeling may limit its effectiveness in tasks that demand precise procedural alignment.

To further examine the generality of our findings, we conducted additional experiments using Llama-2-13B and Pythia-2.8B [32] as backbone models, both of which showed consistent improvements with

Table 1: **In-domain performance comparison across five categories under LLaMA3 and Mistral.** ATV achieves the highest average accuracy on both models using the same number of tokens as ELICIT and I2CL, while outperforming all baselines across most domains and maintaining superior token efficiency over prompt-based methods. All results except I2CL and ATV are from ELICIT [12].

Model		# Tokens	NLU	Reasoning	Knowledge	Math	Safety	Avg.
Llama3	Zero-shot	108.3 ± 1.4	32.2 ± 1.2	31.6 ± 0.2	42.5 ± 1.2	14.0 ± 1.1	35.5 ± 1.2	31.1 ± 1.0
	16-shot	1883.8 ± 0.9	60.6 ± 0.9	56.0 ± 0.4	70.6 ± 1.0	26.7 ± 2.0	62.1 ± 0.3	55.2 ± 0.9
	BM25	2260.9 ± 21.7	56.8 ± 1.4	56.6 ± 0.3	69.4 ± 0.2	**29.0 ± 1.1**	55.5 ± 1.0	53.4 ± 0.8
	ELICIT	108.3 ± 1.4	41.6 ± 0.4	46.7 ± 0.1	60.6 ± 1.4	19.1 ± 1.4	49.9 ± 2.1	43.5 ± 0.8
	I2CL	108.3 ± 1.4	52.4 ± 4.6	48.4 ± 0.9	52.2 ± 3.1	17.9 ± 2.2	45.6 ± 2.4	43.3 ± 0.7
	ATV	108.3 ± 1.4	**61.0 ± 5.0**	**76.1 ± 1.3**	**73.0 ± 1.6**	25.8 ± 2.0	**74.8 ± 0.4**	**62.1 ± 1.5**
Mistral	Zero-shot	123.5 ± 1.7	29.6 ± 1.2	26.9 ± 0.4	45.5 ± 1.3	2.8 ± 0.1	36.1 ± 0.3	28.2 ± 0.5
	16-shot	2161.3 ± 0.9	55.3 ± 0.5	52.1 ± 0.5	70.8 ± 0.4	23.7 ± 1.7	63.1 ± 0.6	53.0 ± 0.1
	BM25	2655.2 ± 27.3	55.2 ± 0.3	66.0 ± 0.5	70.2 ± 1.9	**24.1 ± 0.4**	62.1 ± 0.5	55.5 ± 0.4
	ELICIT	123.5 ± 1.7	41.9 ± 1.0	48.3 ± 0.3	59.4 ± 0.9	20.3 ± 0.9	48.7 ± 1.8	43.7 ± 0.6
	I2CL	123.5 ± 1.7	48.6 ± 0.9	47.3 ± 1.5	59.6 ± 0.8	17.6 ± 1.9	49.4 ± 1.0	44.5 ± 0.6
	ATV	123.5 ± 1.7	**60.8 ± 2.8**	**69.1 ± 1.6**	**71.4 ± 4.5**	20.8 ± 2.4	**69.4 ± 1.9**	**58.3 ± 1.3**

Table 2: **Performance on unseen tasks not included in the ATV training set, evaluated under LLaMA3 and Mistral.** ATV achieves the highest average accuracy across all methods while using significantly fewer tokens than prompt-based and fixed vector approaches, demonstrating strong generalization. All results except I2CL and ATV are from ELICIT [12].

Model		# Tokens	GLUE COLA	BBQ Religion	Deepmind	MMLU-Psychology	BBH-five-objects	Avg.
Llama3	Zero-shot	103.6 ± 47.7	72.0 ± 0.7	38.6 ± 1.1	17.5 ± 2.6	54.2 ± 0.3	17.1 ± 0.0	39.9 ± 0.8
	BM25	2502.8 ± 26.0	55.4 ± 1.0	64.6 ± 1.3	**30.7 ± 1.7**	**83.0 ± 0.1**	48.3 ± 0.0	56.4 ± 0.4
	ELICIT	103.6 ± 47.7	63.4 ± 0.9	45.0 ± 0.7	23.7 ± 3.4	70.0 ± 0.6	25.7 ± 0.0	45.6 ± 0.4
	I2CL	103.6 ± 47.7	26.1 ± 0.6	39.4 ± 3.1	23.5 ± 3.7	75.0 ± 1.0	27.3 ± 2.5	38.3 ± 2.2
	ATV	103.6 ± 47.7	**77.6 ± 2.7**	**80.8 ± 2.6**	26.4 ± 2.7	80.6 ± 2.3	**51.7 ± 3.1**	**63.4 ± 2.5**
Mistral	Zero-shot	115.4 ± 51.0	43.3 ± 1.1	35.4 ± 3.3	9.0 ± 0.4	57.9 ± 0.7	7.4 ± 0.0	30.6 ± 1.0
	BM25	2804.6 ± 27.6	44.4 ± 2.2	70.7 ± 0.7	**26.6 ± 3.9**	**78.7 ± 1.1**	25.7 ± 0.0	49.2 ± 0.3
	ELICIT	115.4 ± 51.0	41.7 ± 0.8	42.1 ± 2.5	25.1 ± 1.2	65.6 ± 0.6	15.6 ± 0.0	38.0 ± 0.6
	I2CL	115.4 ± 51.0	53.3 ± 1.3	48.4 ± 6.5	22.0 ± 2.6	72.6 ± 0.2	22.9 ± 4.5	43.9 ± 3.0
	ATV	115.4 ± 51.0	**79.8 ± 7.1**	**81.7 ± 2.2**	24.6 ± 5.3	70.7 ± 1.0	**40.3 ± 3.6**	**59.4 ± 2.6**

ATV. Results for these settings are provided in Appendix D. Overall, these results demonstrate the strength of adaptive task representations in most language understanding scenarios, while suggesting that tasks relying on surface-level similarity may benefit more from retrieval-based approaches.

While ATV's initial performance on Math tasks is lower than retrieval-based baselines, further analysis shows that this is not an inherent limitation. To investigate, we compared ATV and LoRA when trained on all tasks versus only on Math datasets. The results demonstrate that ATV's accuracy improves substantially with targeted training, whereas LoRA does not exhibit similar gains. These findings indicate that ATV is fully capable of handling complex procedural domains when training is appropriately allocated. Additional details are provided in Appendix E.

Beyond task accuracy, we also evaluated two key aspects of output reliability: adherence to specified formats and response consistency across paraphrased prompts. As detailed in Appendix F, ATV demonstrates format adherence and consistency scores that are comparable to or exceed ICL-based baselines. These findings indicate that ATV's performance improvements are achieved without compromising structural reliability or coherence.

4.3 Generalization to Unseen Tasks

To assess generalization, we evaluate ATV on a set of unseen tasks held out from training, covering domains such as linguistic acceptability, bias detection, and scientific reasoning.

As shown in Table 2, ATV achieves the highest average accuracy on both LLaMA3 and Mistral, likely due to its query-conditioned task vectors that allow adaptation to novel tasks even without explicit demonstrations. This demonstrates the strength of ATV in generalizing beyond in-domain tasks while maintaining strong token efficiency.

We also evaluated ATV on adversarial generalization using the HANS dataset [33], which is specifically designed to expose heuristic-driven failures in NLI models. While other existing approaches collapse in this setting, ATV maintains strong accuracy, highlighting its robustness to highly dissimilar and adversarial tasks. A detailed analysis and results are provided in Appendix G.

4.4 Layer-wise Analysis of Injection Strategies

We conduct a layer-wise analysis to examine how injection depth influences performance for both ATV and ELICIT, revealing distinct patterns in how the two methods interact with different transformer layers. While ELICIT requires identifying a single best injection layer per task, we perform a uniform layer-wise evaluation for both methods by injecting into the bottom, middle, or top third of the model.

Table 3: **Layer-wise performance comparison between ATV and ELICIT on LLaMA3.** While ATV shows strong performance when injected into bottom layers, ELICIT performs best when applied only to top layers. This contrast highlights the different functional dependencies of the two methods. Reported differences are measured with respect to the full-layer injection setting.

Injected Layer		Avg. acc (ATV)	Diff. (ATV)	Avg. acc (ELICIT)	Diff. (ELICIT)
Llama3	All Layers	**62.1 ± 1.5**	-	30.9 ± 0.8	-
	Bottom $\frac{1}{3}$	60.5 ± 0.7	**-1.6**	23.5 ± 0.9	-17.7
	Middle $\frac{1}{3}$	43.2 ± 1.0	-18.9	17.8 ± 0.2	-18.5
	Top $\frac{1}{3}$	32.6 ± 0.4	-29.5	**32.8 ± 0.3**	**+0.6**

We divide the transformer into bottom, middle, and top thirds, and evaluate each method by restricting injection to a single region. As shown in Table 3, ATV retains strong performance when injected into the bottom layers, exhibiting only marginal degradation relative to the full-layer setting. For ELICIT, the best performance is observed when injecting into the top layers, slightly surpassing its full-layer setting. This divergence suggests that ATV benefits from modulating lower-level representations, while ELICIT relies more on upper-layer reasoning.

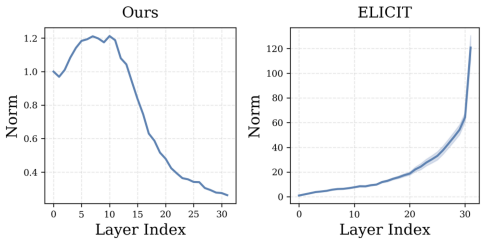

Figure 3: **Layer-wise analysis of vector injection magnitudes.** Left: ATV concentrates vector strength in lower layers, while Right: ELICIT shows a monotonic increase toward top layers. These patterns align with each method's layer-specific performance impact.

Figure 3 visualizes the ℓ_2 norm of the injected vectors across layers. ATV peaks in the bottom layers with balanced magnitudes, whereas ELICIT shows a steep increase in vector strength toward the top, with much larger magnitudes.

This behavioral difference aligns with the performance trends in Table 1: ATV achieves consistently strong results across all five task categories, not just in NLU. The effectiveness of early-layer modulation is consistent with prior studies on transformer specialization [34–36], which show that lower layers primarily encode lexical and syntactic features, while upper layers are responsible for semantic reasoning and task-specific abstraction.

4.5 Efficiency and Fair Comparison with LoRA

To ensure a fair comparison with LoRA and to address concerns regarding computational cost, we evaluate both training- and inference-time efficiency. We additionally introduce **ATV-Light**, a parameter-efficient variant configured to match LoRA's ∼20M trainable parameters by applying LoRA to the generator and introducing a bottleneck in the projection module.

Training Efficiency. We compared trainable parameters, peak GPU memory usage, and training time per epoch. As summarized in Table 4, ATV-Light and LoRA consume nearly identical memory, while ATV incurs only a modest increase. Both ATV and ATV-Light achieve substantially shorter training times than LoRA, since updates are confined to smaller modules rather than requiring backpropagation through the entire large model. These results confirm that ATV maintains training efficiency even when matched to LoRA's parameter budget.

Table 4: **Training efficiency comparison.** We compare the number of trainable parameters, peak GPU memory usage, and training time per epoch for LoRA, ATV-Light, and the full ATV. ATV-Light achieves a substantial reduction in training time compared to LoRA while consuming nearly identical memory.

Method	Params	Memory	Training Time
LoRA	∼20M	23.44 GiB	21m 32s
ATV-Light	∼20M	23.77 GiB	16m 43s
ATV	∼200M	25.47 GiB	16m 29s

Inference Efficiency. We evaluated accuracy, inference latency, and peak GPU memory usage, as shown in Table 5. ATV achieves the highest accuracy on both in-domain and out-of-domain tasks, with ATV-Light also surpassing LoRA. Regarding efficiency, ATV and ATV-Light exhibit memory usage comparable to LoRA and I2CL and are significantly faster than ELICIT, which incurs overhead from per-query retrieval. While ATV and ATV-Light show slightly increased latency compared to LoRA and I2CL, the difference is minor relative to their accuracy gains. These results demonstrate that ATV delivers state-of-the-art performance with minimal inference overhead.

Table 5: **Inference efficiency and accuracy comparison.** We measure inference latency per sample, peak memory usage, and performance on ID and OOD tasks. ATV and ATV-Light achieve state-of-the-art accuracy with only a minor increase in latency compared to LoRA, and are substantially faster than retrieval-based methods like ELICIT.

Method	Latency	Memory	ID Acc.	OOD Acc.
ELICIT	0.077s	15.52 GiB	43.5 ± 0.8	45.6 ± 0.4
I2CL	0.037s	14.96 GiB	43.3 ± 0.7	38.3 ± 2.2
LoRA	0.030s	15.45 GiB	56.0 ± 1.4	52.0 ± 3.2
ATV-Light	0.055s	15.51 GiB	60.6 ± 1.5	60.9 ± 0.9
ATV	0.047s	15.81 GiB	$\mathbf{62.1 \pm 1.5}$	$\mathbf{63.4 \pm 2.5}$

Overall, our results show that ATV offers superior performance while maintaining efficiency in both training and inference, making it a strong alternative to LoRA and retrieval-based methods.

4.6 Visualizing Task Vector Distributions

We use t-SNE to visualize query-specific task vectors and assess how ATV captures input variation.

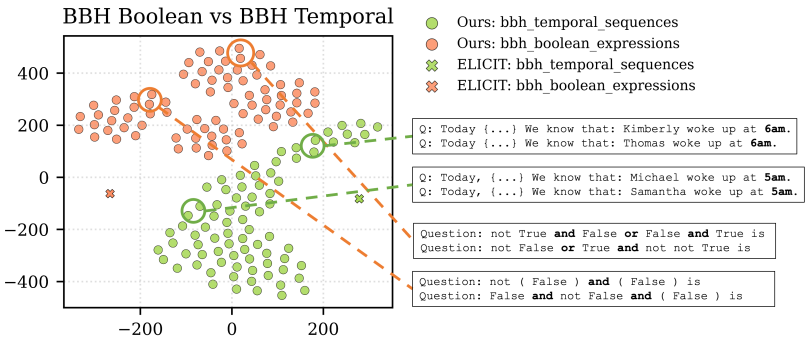

Figure 4: **t-SNE visualization of task vector distributions for two BBH tasks.** Each dot represents a query-specific task vector generated by ATV, while crosses denote the fixed task vectors used by ELICIT. We observe that vectors from similar queries tend to be grouped together, indicating that ATV adapts its representations based on the input, while ELICIT draws from a fixed demonstration pool and captures less query-level variation as a result.

Figure 4 shows the projected vectors for two BBH tasks alongside their ELICIT counterparts. We observe that, in ATV, vectors from similar queries tend to appear closer together in the embedding space, suggesting that the method captures input-specific variation within and across tasks. In contrast, ELICIT assigns a single static vector per task, which fails to reflect such internal diversity.

5 Conclusion

We introduced Adaptive Task Vectors (ATV), a framework that dynamically generates query-specific vectors to causally intervene in LLMs' computational processes without parameter changes. Our theoretical analysis shows that ATV is expressively equivalent to LoRA while offering better adaptability. Empirically, ATV outperforms prior approaches across 20 datasets while maintaining token efficiency and demonstrating strong generalization to unseen tasks, confirming its effectiveness for precise and context-sensitive causal steering of model behavior.

Limitation and Impact statement Although ATV performs well overall, its effectiveness varies by task, and ATV can sometimes underperform compared to retrieval-based approaches. Additionally, because ATV depends on the input data, unintended behaviors may occur if the dataset is poorly curated. Careful dataset understanding is therefore essential when applying this method.

References

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

[2] Q. Dong, L. Li, D. Dai, C. Zheng, J. Ma, R. Li, H. Xia, J. Xu, Z. Wu, T. Liu *et al.*, "A survey on in-context learning," *arXiv preprint arXiv:2301.00234*, 2022.

[3] J. Liu, D. Shen, Y. Zhang, B. Dolan, L. Carin, and W. Chen, "What makes good in-context examples for gpt-3?" *arXiv preprint arXiv:2101.06804*, 2021.

[4] K. Peng, L. Ding, Y. Yuan, X. Liu, M. Zhang, Y. Ouyang, and D. Tao, "Revisiting demonstration selection strategies in in-context learning," *arXiv preprint arXiv:2401.12087*, 2024.

[5] X. Wang, W. Zhu, and W. Y. Wang, "Large language models are implicitly topic models: Explaining and finding good demonstrations for in-context learning," *arXiv preprint arXiv:2301.11916*, vol. 1, p. 15, 2023.

[6] T. Li, G. Zhang, Q. D. Do, X. Yue, and W. Chen, "Long-context llms struggle with long in-context learning," *arXiv preprint arXiv:2404.02060*, 2024.

[7] Y. Kuratov, A. Bulatov, P. Anokhin, I. Rodkin, D. Sorokin, A. Sorokin, and M. Burtsev, "Babilong: Testing the limits of llms with long context reasoning-in-a-haystack," *Advances in Neural Information Processing Systems*, vol. 37, pp. 106 519–106 554, 2024.

[8] R. Hendel, M. Geva, and A. Globerson, "In-context learning creates task vectors," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds., Dec. 2023, pp. 9318–9333.

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

[10] S. Liu, H. Ye, L. Xing, and J. Zou, "In-context vectors: making in context learning more effective and controllable through latent space steering," in *Proceedings of the 41st International Conference on Machine Learning*, 2024, pp. 32 287–32 307.

[11] Z. Li, Z. Xu, L. Han, Y. Gao, S. Wen, D. Liu, H. Wang, and D. N. Metaxas, "Implicit in-context learning," in *The Thirteenth International Conference on Learning Representations*, 2025.

[12] F. Wang, J. Yan, Y. Zhang, and T. Lin, "ELICIT: LLM augmentation via external in-context capability," in *The Thirteenth International Conference on Learning Representations*, 2025.

[13] Z. Zhao, E. Wallace, S. Feng, D. Klein, and S. Singh, "Calibrate before use: Improving few-shot performance of language models," in *International conference on machine learning*. PMLR, 2021, pp. 12 697–12 706.

[14] Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp, "Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity," *arXiv preprint arXiv:2104.08786*, 2021.

[15] K. Zhang, A. Lv, Y. Chen, H. Ha, T. Xu, and R. Yan, "Batch-icl: Effective, efficient, and order-agnostic in-context learning," *arXiv preprint arXiv:2401.06469*, 2024.

[16] H. Li, Y. Zhang, S. Zhang, M. Wang, S. Liu, and P.-Y. Chen, "When is task vector provably effective for model editing? a generalization analysis of nonlinear transformers," *arXiv preprint arXiv:2504.10957*, 2025.

[17] L. Yang, Z. Lin, K. Lee, D. Papailiopoulos, and R. Nowak, "Task vectors in in-context learning: Emergence, formation, and benefit," *arXiv preprint arXiv:2501.09240*, 2025.

[18] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[19] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.

[20] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. Le *et al.*, "Least-to-most prompting enables complex reasoning in large language models," *arXiv preprint arXiv:2205.10625*, 2022.

[21] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer, "Rethinking the role of demonstrations: What makes in-context learning work?" in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022, pp. 11 048–11 064.

[22] O. Rubin, J. Herzig, and J. Berant, "Learning to retrieve prompts for in-context learning," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2022, pp. 2655–2671.

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

[24] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[25] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[26] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," *arXiv preprint arXiv:2101.00190*, 2021.

[27] D. Dai, Y. Sun, L. Dong, Y. Hao, S. Ma, Z. Sui, and F. Wei, "Why can gpt learn in-context? language models implicitly perform gradient descent as meta-optimizers," *arXiv preprint arXiv:2212.10559*, 2022.

[28] J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig, "Towards a unified view of parameter-efficient transfer learning," in *International Conference on Learning Representations*, 2022.

[29] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[30] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mistral 7b," 2023. [Online]. Available: https://arxiv.org/abs/2310.06825

[31] S. Robertson, H. Zaragoza *et al.*, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.

[32] S. Biderman, H. Schoelkopf, Q. G. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M. A. Khan, S. Purohit, U. S. Prashanth, E. Raff *et al.*, "Pythia: A suite for analyzing large language models across training and scaling," in *International Conference on Machine Learning*. PMLR, 2023, pp. 2397–2430.

[33] R. T. McCoy, E. Pavlick, and T. Linzen, "Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, A. Korhonen, D. Traum, and L. Màrquez, Eds. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 3428–3448. [Online]. Available: https://aclanthology.org/P19-1334/

[34] A. Rogers, O. Kovaleva, and A. Rumshisky, "A primer in bertology: What we know about how bert works," *Transactions of the association for computational linguistics*, vol. 8, pp. 842–866, 2021.

[35] I. Tenney, D. Das, and E. Pavlick, "Bert rediscovers the classical nlp pipeline," *arXiv preprint arXiv:1905.05950*, 2019.

[36] N. Elhage, N. Nanda, C. Olsson, T. Henighan, N. Joseph, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly *et al.*, "A mathematical framework for transformer circuits," *Transformer Circuits Thread*, vol. 1, no. 1, p. 12, 2021.

[37] A. Talmor, J. Herzig, N. Lourie, and J. Berant, "Commonsenseqa: A question answering challenge targeting commonsense knowledge," *arXiv preprint arXiv:1811.00937*, 2018.

[38] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal, "Can a suit of armor conduct electricity? a new dataset for open book question answering," *arXiv preprint arXiv:1809.02789*, 2018.

[39] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" *arXiv preprint arXiv:1905.07830*, 2019.

[40] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova, "Boolq: Exploring the surprising difficulty of natural yes/no questions," *arXiv preprint arXiv:1905.10044*, 2019.

[41] M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou *et al.*, "Challenging big-bench tasks and whether chain-of-thought can solve them," *arXiv preprint arXiv:2210.09261*, 2022.

[42] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *arXiv preprint arXiv:1803.05457*, 2018.

[43] A. Amini, S. Gabriel, P. Lin, R. Koncel-Kedziorski, Y. Choi, and H. Hajishirzi, "Mathqa: Towards interpretable math word problem solving with operation-based formalisms," *arXiv preprint arXiv:1905.13319*, 2019.

[44] Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang *et al.*, "Mmlu-pro: A more robust and challenging multi-task language understanding benchmark," in *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[45] N. Nangia, C. Vania, R. Bhalerao, and S. R. Bowman, "Crows-pairs: A challenge dataset for measuring social biases in masked language models," *arXiv preprint arXiv:2010.00133*, 2020.

[46] A. Parrish, A. Chen, N. Nangia, V. Padmakumar, J. Phang, J. Thompson, P. M. Htut, and S. R. Bowman, "Bbq: A hand-built bias benchmark for question answering," *arXiv preprint arXiv:2110.08193*, 2021.

[47] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[48] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[49] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "Superglue: A stickier benchmark for general-purpose language understanding systems," *Advances in neural information processing systems*, vol. 32, 2019.

[50] D. Saxton, E. Grefenstette, F. Hill, and P. Kohli, "Analysing mathematical reasoning abilities of neural models," *arXiv preprint arXiv:1904.01557*, 2019.

[51] L. Sutawika, L. Gao, H. Schoelkopf, S. Biderman, J. Tow, B. Abbasi, ben fattori, C. Lovering, farzanehnakhaee70, J. Phang, A. Thite, Fazz, Aflah, N. Muennighoff, T. Wang, sdtblck, nopperl, gakada, tttyuntian, researcher2, Chris, J. Etxaniz, Z. Kasner, Khalid, J. Hsu, AndyZwei, P. S. Ammanamanchi, D. Groeneveld, E. Smith, and E. Tang, "Eleutherai/lm-evaluation-harness: Major refactor," Dec. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.10256836

A Proofs for Theoretical Results

A.1 Proof of Theorem 1

A.1.1 Preliminaries

Consider a single transformer layer ℓ with hidden state dimension d_ℓ. Let $h^\ell \in \mathbb{R}^{d_\ell}$ denote the hidden state of the last token at that layer.

ATV (Adaptive Task Vector) update ATV injects an additive low-rank vector

$$\tilde{h}^\ell = h^\ell + \lambda\, v_{\text{small}} A_\ell,$$

where

- $h^\ell \in \mathbb{R}^{d_\ell}$: last-token hidden state in layer ℓ.
- $v_{\text{small}} \in \mathbb{R}^{d_s}$ with $d_s \ll d_\ell$: query-specific vector from the $\mathcal{M}_{\text{small}}$.
- $A_\ell \in \mathbb{R}^{d_s \times d_\ell}$: the ℓ-th $d_s \times d_\ell$ block of the linear expansion $f_\theta(v_{\text{small}}) = v_{\text{ATV}} \in \mathbb{R}^{L \times d_\ell}$; thus, $v_{\text{small}} A_\ell$ is the ℓ-th row of v_{ATV}.
- $\lambda \in \mathbb{R}$: global scaling constant.

LoRA (low-rank adaptation) update LoRA applies a rank-r additive update to the projection. Its effect on the hidden state is:

$$\hat{h}^\ell = h^\ell + s\, x^\top W_{\text{down}} W_{\text{up}},$$

where

- $x^\top \in \mathbb{R}^{1 \times d_\ell}$: the current-token activation entering the augmented projection; for a query / value projection, this equals $(h^\ell)^\top$.
- $W_{\text{down}} \in \mathbb{R}^{d_\ell \times r}$ and $W_{\text{up}} \in \mathbb{R}^{r \times d_\ell}$: LoRA projection matrices.
- $s \in \mathbb{R}$: global scaling constant.

Assumption (A1) — matched rank budgets We henceforth set the LoRA bottleneck rank equal to the ATV bottleneck size:

$$r = d_s.$$

A.1.2 Theorem

Theorem (ATV–LoRA equivalence under equal rank) Under Assumption A1:

1. **ATV $\Rightarrow$ LoRA (simulation).**
   For every pair $(h^\ell, v_{\text{small}})$ there exist static LoRA factors $(W_{\text{down}}, W_{\text{up}})$ and a scale s, all independent of the runtime query, such that
   $$\tilde{h}^\ell = \hat{h}^\ell \quad \text{for all inputs.}$$
2. **LoRA $\Rightarrow$ ATV (simulation).**
   Conversely, any LoRA update with rank $r = d_s$ can be expressed in ATV form by an appropriate choice of $(\lambda, v_{\text{small}}, A_\ell)$.

Hence, when the rank budgets are matched, ATV and LoRA realize the same class of low-rank additive perturbations to the frozen model; they are expressively equivalent.

A.1.3 Proof

Throughout the proof we fix the layer index ℓ and omit it from superscripts whenever no ambiguity arises.

Step 1 Factorize the ATV increment The additive term introduced by ATV is

$$\Delta h_{\text{ATV}} = \lambda\, v_{\text{small}} A_\ell = \underbrace{(\lambda v_{\text{small}})}_{1\times d_s}\ \underbrace{A_\ell}_{d_s\times d_\ell}.$$

This is an outer product of a $1 \times d_s$ row vector and a $d_s \times d_\ell$ matrix, so its matrix rank satisfies

$$\operatorname{rank}(\Delta h_{\text{ATV}}) \le d_s.$$

Hence, ATV always adds a vector lying in a rank-d_s subspace of $\mathbb{R}^{d_\ell}$.

Step 2 Factorize the LoRA increment LoRA's contribution can be written in vector form as

$$\Delta h_{\text{LoRA}} = s\, x^\top W_{\text{down}} W_{\text{up}} = \underbrace{(s\, x^\top W_{\text{down}})}_{1\times r}\ \underbrace{W_{\text{up}}}_{r\times d_\ell},$$

which is likewise an outer product—now of shapes $1 \times r$ and $r \times d_\ell$. Consequently,

$$\operatorname{rank}(\Delta h_{\text{LoRA}}) \le r.$$

Under Assumption A1 ($r = d_s$), the rank upper bounds obtained in Steps 1 and 2 are identical, establishing that the two increments live in subspaces of equal maximal rank.

Step 3 Rewrite ATV as a LoRA update (ATV $\Rightarrow$ LoRA) Let

$$W_{\text{up}} := A_\ell \quad \text{and} \quad W_{\text{down}} := M \in \mathbb{R}^{d_\ell \times d_s}.$$

We choose M so that the following linear constraint holds for the last token:

$$x^\top M = \lambda\, v \in \mathbb{R}^{1\times d_s}. \tag{9}$$

Existence of a solution. The row vector $x^\top$ has d_ℓ free coordinates, whereas the right-hand side specifies only d_s values with $d_s \ll d_\ell$; consequently, the under-determined system (1) always admits a solution provided $x^\top \neq 0$. A canonical choice is

$$M := x^{+\top}(\lambda v),$$

where $x^{+\top}$ denotes the Moore–Penrose pseudoinverse of $x^\top$ and satisfies $x^\top x^{+\top} = 1$.

Resulting LoRA increment. With these matrices,

$$\Delta h_{\text{LoRA}} = s\, x^\top W_{\text{down}} W_{\text{up}} = s\,(x^\top M)\, A_\ell = s\,(\lambda v)\, A_\ell.$$

Selecting the scale $s = 1$ yields

$$\Delta h_{\text{LoRA}} = \lambda v A_\ell = \Delta h_{\text{ATV}},$$

and hence the LoRA-modified hidden state satisfies $\tilde{h}^\ell = \hat{h}^\ell$.

This completes the direction "ATV implies LoRA" under the rank-matching assumption $r = d_s$.

Step 4 Rewrite LoRA as an ATV update (LoRA $\Rightarrow$ ATV) Take fixed LoRA factors $W_{\text{down}} \in \mathbb{R}^{d_\ell \times r}$, $W_{\text{up}} \in \mathbb{R}^{r\times d_\ell}$ with $r = d_s$.

For the last token we observe the row vector

$$x^\top W_{\text{down}} \in \mathbb{R}^{1\times d_s}.$$

Row-vector SVD. Compute a thin singular-value decomposition

$$x^\top W_{\text{down}} = u \Sigma v^\top,$$

where

- $u \in \mathbb{R}^{1\times d_s}$ is orthonormal,
- $\Sigma \in \mathbb{R}^{d_s \times d_s}$ is diagonal (rank ≤ 1),

- $v^\top \in \mathbb{R}^{d_s \times d_s}$ is orthonormal.

Define ATV parameters

$$\lambda := \|u\Sigma\|_2, \quad v := \frac{u\Sigma}{\|u\Sigma\|_2} \in \mathbb{R}^{d_s}, \quad A_\ell := v^\top W_{\text{up}} \in \mathbb{R}^{d_s \times d_\ell}.$$

The vector v has unit norm by construction, and A_ℓ keeps the required shape.

ATV increment equals LoRA increment

$$\lambda v A_\ell = (\|u\Sigma\|_2) \frac{u\Sigma}{\|u\Sigma\|_2} v^\top W_{\text{up}} = (u\Sigma)\, v^\top W_{\text{up}} = x^\top W_{\text{down}} W_{\text{up}} = \Delta h_{\text{LoRA}}.$$

Hence $\tilde{h}^\ell = \hat{h}^\ell$; the LoRA update is reproduced exactly by ATV.

Conclusion of the proof Both ATV and LoRA ultimately realize the same operation class

$$h \longmapsto h + (\text{rank} \le r) \text{ outer product},$$

and, under the matched rank budget $r = d_s$, have identical functional capacity on a frozen backbone. This completes the proof of the theorem.

A.1.4 Discussion of Assumptions

- **Matched rank budgets ($r = d_s$).**
  The constructive proof fixes the LoRA rank to equal the ATV bottleneck size so that both methods have the same number of degrees of freedom.
  If $d_s < r$, LoRA has $r - d_s$ surplus channels; turning those channels off yields an exact simulation of ATV, so LoRA is (weakly) more expressive.
  If $d_s > r$, ATV can move in directions that LoRA cannot represent. Projecting the ATV increment onto a rank-r subspace gives the closest LoRA-matchable update, so equivalence holds only "up to rank-r projection."
- **Non-zero input row vector $x^\top$.**
  The linear system $x^\top M = \lambda v$ in Step 3 requires $x^\top \neq 0$ to admit a solution via the Moore–Penrose pseudoinverse.
  In a transformer, $x^\top$ is simply the hidden state of the token entering the query/value projection and is never identically zero after normal training; therefore the assumption is benign in practice.

A.1.5 Operational Differences Between ATV and LoRA

Table 6: Implementation-level differences between ATV and LoRA.

Aspect	ATV	LoRA
Insertion point	Adds a vector $\lambda v A_\ell$ *after* the hidden state is computed	Adds a low-rank matrix $W_{\text{down}} W_{\text{up}}$ *inside* the projection weight
Learned parameters	Auxiliary small model $\mathcal{M}_{\text{small}}$ + expansion blocks A_ℓ of the linear expansion $f_\theta()$	Two fixed factor matrices $W_{\text{down}}, W_{\text{up}}$

Thus, even under equal rank budgets, the two methods differ operationally: ATV perturbs activations directly in the hidden state space, whereas LoRA perturbs projection weights via a static low-rank matrix. Nevertheless, as shown in the theorem, these implementation choices realize the same class of rank-r additive perturbations on a frozen backbone, yielding identical expressive power when $r = d_s$.

A.2 Proof of Theorem 2

A.2.1 Preliminaries

Linear approximation: $\text{Attn}(Q, K, V) \approx QK^\top V$ [27]

Attention ouptputs from Prefix-Tuning

$$\text{Attn}_{\text{prefix}} = \text{Attn}(xW_q, [P_k; CW_k], [P_v; CW_v])$$

- $x = [x_1, x_2, ..., x_T] \in \mathbb{R}^{T \times d_l}$
- $C \in \mathbb{R}^{m \times d_l}$:
  context sequence of length m with d_l dimension(l-th layer's dimension)
- $P_k, P_v \in \mathbb{R}^{p \times d_l}$: p tunable prefix vectors to the keys and values

Attention ouptputs from Prefix-Tuning

$$\text{Attn}_{\text{ATV}} = \text{Attn}((x + e_T \cdot (v^l_{ATV})^\top)W_q, (C + e_m \cdot (v^l_{ATV})^\top)W_k, (C + e_m \cdot (v^l_{ATV})^\top)W_v)$$

- $v^l_{ATV} \in \mathbb{R}^{d_l}$
- $e_m = [0, ..., 0, 1] \in \mathbb{R}^{m \times 1}$
- $(e_m \cdot (v^l_{ATV})^\top)W_k = P'_k$

A.2.2 Theorem

Theorem (ATV is more expressive than Prefix-Tuning) The representational space $\mathcal{F}$ of Attn_{ATV} includes that of $\text{Attn}_{\text{prefix}}$:

$$\mathcal{F}(\text{Attn}_{\text{prefix}}) \subseteq \mathcal{F}(\text{Attn}_{\text{ATV}})$$

A.2.3 Process of Derivation

Linear approximation of Prefix-Tuning

$$\begin{aligned}
\text{Attn}_{\text{prefix}} &= \text{Attn}(xW_q, concat(P_k, CW_k), concat(P_v, CW_v)) \\
&= \text{softmax}(xW_q([P_k; CW_k])^\top)[P_v; CW_v] \\
&\approx xW_q(P_k; CW_k])^\top([P_v; CW_v])(\because \text{Attn}(Q, K, V) \approx QK^\top V) \\
&= xW_q(P_k)^\top P_v + xW_q(CW_k)^\top CW_v
\end{aligned}$$

Linear approximation of ATV

$$\begin{aligned}
\text{Attn}_{\text{ATV}} =& \text{Attn}\left((x + e_T \cdot (v^l_{ATV})^\top)W_q,\ (C + e_m \cdot (v^l_{ATV})^\top)W_k,\ (C + e_m \cdot (v^l_{ATV})^\top)W_v\right) \\
\approx& \Big[xW_q(CW_k)^\top + xW_q(e_m \cdot (v^l_{ATV})^\top)^\top \\
&+ e_T \cdot (v^l_{ATV})^\top W_q(CW_k)^\top + e_T \cdot (v^l_{ATV})^\top W_q(e_m \cdot (v^l_{ATV})^\top W_k)^\top \Big] \\
&\cdot \left(CW_v + e_m \cdot (v^l_{ATV})^\top W_v\right)
\end{aligned}$$

Let $(e_m \cdot (v^l_{ATV})^\top)W_k = P'_k, (e_m \cdot (v^l_{ATV})^\top)W_v = P'_v, (e_T \cdot (v^l_{ATV})^\top)W_q = P'_q$

$$\begin{aligned}
\Rightarrow \quad \text{Attn}_{\text{ATV}} \approx\ & xW_q(CW_k)^\top CW_v + xW_q(P'_k)^\top P'_v (\text{Similar to Attn}_{\text{Prefix}}) && (T_1 + T_2) \\
&+ xW_q(P'_k)^\top CW_v && (T_3) \\
&+ xW_q(CW_k)^\top P'_v && (T_4) \\
&+ P'_q(CW_k)^\top CW_v && (T_5) \\
&+ P'_q(P'_k)^\top CW_v && (T_6) \\
&+ P'_q(CW_k)^\top P'_v && (T_7) \\
&+ P'_q {P'_k}^\top P'_v && (T_8)
\end{aligned}$$

A.2.4 Analysis of Each Term in Attn_{ATV}

For each term we report (i) the intuition behind the interaction, and (ii) how it extends or subsumes the behavior attainable with classic Prefix-Tuning (PT), treating the ATV-generated vectors P'_k, P'_v, P'_q as soft-prefix counterparts to PT's fixed prefixes (see Table 7).

Table 7: Qualitative roles of ATV attention terms and their relation to Prefix-Tuning (PT).

Term	Qualitative role	Relation to Prefix-Tuning (PT) / Added expressivity
T_1	**Base attention** of the frozen model	**Representable in PT (identical).** No additional expressivity; both methods preserve this term unchanged.
T_2	**Prefix keys & values only.** Query attends only to prefix key/value	**Representable in PT (exact match).** This is the sole extra path PT can realize; ATV contains it and can therefore emulate PT exactly.
T_3	**Prefix key → content values.** A soft prefix key reshapes the attention weights, but the actual information still comes from the content values.	**Not representable in PT.** PT would need a separate learned key for every content token, whereas ATV achieves the same effect with a single soft key—thus widening the attention design space.
T_4	**Content keys → prefix value.** Normal keys set the weights, but an extra value generated by the adapter is injected at the output stage.	**Not representable in PT.** PT lacks a mechanism to inject new information exclusively at the value stage for existing keys; ATV can graft auxiliary content into any token's output.
T_5	**Prefix query.** The query itself is shifted in a new direction while still using ordinary keys and values.	**Not representable in PT.** Because PT keeps W_q frozen, it cannot alter queries. ATV adds a query-side degree of freedom, enabling new attention directions.
T_6	**Prefix query + key.** Both sides of the similarity come from the same learnable vector, but the output is still built from content values.	**Not representable in PT.** ATV can simultaneously steer queries and keys while still reading content values, providing a finer redistribution of attention mass that PT cannot mimic.
T_7	**Prefix query + value.** Ordinary keys choose the weights; the returned information comes from a prefix-generated value.	**Not representable in PT.** PT can supply prefix values but cannot adapt the query; ATV adds this missing query modulation, enhancing expressivity.
T_8	**Full prefix triad.** Query, key, and value are all produced by the same low-rank adapter, yielding a fully synthetic attention path.	**Not representable in PT.** PT has no mechanism for a fully synthetic attention channel without real tokens; ATV introduces an entirely new path, further enlarging the representational space.

Key points of the theorem

- **Containment:** PT spans only the subspace generated by $T_1 + T_2$. ATV keeps those terms and introduces $T_3 - T_8$, hence

$$\mathcal{F}(\text{Attn}_{\text{prefix}}) \subseteq \mathcal{F}(\text{Attn}_{\text{ATV}})$$

- **Query-side freedom (T_5, T_6, T_7, T_8):** Because PT never changes W_q, any behavior that requires altering the query vector is strictly outside its representational span. ATV realizes this through the additive query P'_q.
- **Mixed interactions (T_3, T_4):** Unlike PT, ATV can blend a single soft prefix key or value with the untouched content tokens. To even approximate T_3, PT would have to add one custom prefix key for every content token, which is an impractical workaround, and T_4 cannot be reproduced by PT at all.
- **Full prefix channel (T_8):** A complete synthetic path lets ATV add task-specific information even when the original context is irrelevant, while still using no extra tokens at runtime.

Taken together, the additional six terms explain why ATV is more expressive: it augments the attention operator along every axis (query, key, and value) without introducing heavy retraining or large prefix matrices, yet it can still emulate PT as a special case.

B Detailed Experiments Setting

Our experimental setup follows ELICIT [12], using the same datasets and evaluation protocols. Below we provide detailed specifications.

B.1 Dataset List

All experiments are conducted on the same 20 in-domain tasks and 5 unseen tasks as used in ELICIT. Tasks are categorized as follows:

- **Knowledge**: CommonsenseQA [37], OpenBookQA [38], HellaSwag [39], BoolQ [40]
- **Reasoning**: Four subsets from Big-Bench Hard (BBH) [41] (BBH Boolean Expressions, BBH Date Understanding, BBH Reasoning about Colored Objects, BBH Temporal Sequences), ARC-Challenge [42]
- **Mathematics**: MathQA [43], MMLU Pro-MATH [44]
- **Safety**: Crows-Pairs [45], BBQ-Age [46], Ethics-Commonsense, Ethics-Justice [47]
- **Natural Language Understanding (NLU)**: GLUE (SST-2, QNLI, MNLI) [48], SuperGLUE (WIC, RTE) [49]
- **Unseen**: GLUE COLA, BBQ-Religion, Deepmind [50], MMLU High School Psychology, BBH Logical Deduction Five objects

B.2 Implementation Details and Baseline Configurations

Common Setup. All experiments are conducted on NVIDIA A100 80GB GPUs. We use the same training data splits and evaluation protocols across all methods for fair comparison. Each experiment is repeated 3 times with different random seeds (42, 100, 10) to compute statistical significance. For training, we sample **90 examples per task** from the official training split of each in-domain task (excluding unseen tasks), and use the same sampled data across all baselines.

ATV. We train the small model $\mathcal{M}_{\text{small}}$ (GPT-2, 137M parameters) and the expansion module f_θ jointly with the following hyperparameters. A constant learning rate of **5e-4** is used without warmup or learning rate scheduling, along with **weight decay of 1e-5**. The model is optimized for **15 epochs** using the Adam optimizer.

We inject a task vector $v_{\text{ATV}} \in \mathbb{R}^{L \times d_l}$ into the last token's hidden state at each layer as $\tilde{h}^l = h^l + \lambda v^l_{\text{ATV}}$.

We use a scaling factor of $\boldsymbol{\lambda} = \mathbf{0.001}$ throughout all experiments. In our implementation, the hidden size of the small model is $d_s = 768$ (GPT-2), and the large models (LLaMA3-8B and Mistral-7B) use $d_l = 4096$ with $L = 32$ transformer layers.

ELICIT. We follow the official implementation and configuration of ELICIT. Task vectors are retrieved from a precomputed capability library, each paired with its optimal injection layer. At inference time, the selected vector is additively injected into the frozen LLM at the designated layer. All training and evaluation use the official codebase and default settings.

Each task vector is constructed from 10 exemplars per task, each with a 16-shot prompt. While the total number of unique samples may vary due to overlap, our analysis confirms a minimum of 91 unique samples per task. To ensure fair comparison, we use 90 training samples per task for all other baselines.

I2CL. We adopt the official I2CL implementation [11], modifying only the number of training epochs to 15 for consistency with other baselines. To ensure fair comparison, we deviate from the original setting, which calibrates context vectors and injection coefficients separately for each dataset using task identity. Instead, we train a shared set of coefficients across all datasets while keeping dataset-specific context vectors.

For evaluation on unseen tasks, we use a retrieval strategy that selects the most similar context vector among those obtained from in-domain datasets, based on cosine similarity between the input query and training prompts.

LoRA. We adopt the LoRA configuration described in the I2CL paper, which applies low-rank adaptation to the query and value projection matrices in all attention layers. The setup uses rank $r = 8$, scaling factor $\alpha = 32$, and a dropout rate of 0.05. All other settings, including the optimizer, follow the official implementation. However, as the original learning rate of $1e{-}3$ resulted in poor performance in our setting, we adjust it to $4e{-}4$.

B.3 Task-Specific Prompt List

We follow the prompt template settings from ELICIT, which adopts task-specific templates manually crafted based on guidelines from `lm-harness` [51] and the `chain-of-thought-hub`[1]. For each task, we use the same three distinct question templates as provided in ELICIT. The full set of question templates used for each task is listed in Table 8.

The answer-side format is consistent across all tasks and composed of the following structure:

- A line break (`\n`) after the question template,
- A list of options in the form: `Options: (A) ..., (B) ..., (C) ..., ...`,
- One of the following three answer prefixes:
  - `A:`
  - `Answer:`
  - `The answer is`

By combining the 3 question templates with the 3 answer prefixes, we construct 9 distinct prompt variants per task. Following the ELICIT setup, only the `A:` answer prefix is used during training, while all 3 answer formats are used during evaluation to assess generalization to unseen answer styles. This setting is consistently applied across all baseline methods.

Table 8: **Question-side templates used for each task.** Each task uses three distinct prompt formats as provided in the original ELICIT setting.

Task (Dataset)	Template
CommonsenseQA	• The following are multiple choice questions (with answers) about commonsense knowledge reasoning. Finish your answer with 'X' where X is the correct letter choice.\n\nQuestion: {input} • Below are multiple-choice questions about commonsense reasoning. Answer with 'X', X being the correct option.\n\nQuestion: {input} • Respond to these multiple-choice questions on commonsense knowledge. Conclude with 'X', where X is the right letter choice.\n\nQuestion: {input}
OpenBookQA	• The following are multiple choice questions (with answers) about multi-step reasoning. Finish your answer with 'X' where X is the correct letter choice.\n\nQuestion: {input} • The following are multiple-choice questions testing multi-step reasoning. Answer with 'X', X being the correct option.\n\nQuestion: {input} • Answer these multiple-choice questions involving multi-step logical thinking. Conclude with 'X', where X is the right letter choice.\n\nQuestion: {input}

Continued on next page

[1] `https://github.com/FranxYao/chain-of-thought-hub`

Table 8 – continued from previous page

Task (Dataset)	Template
HellaSwag	• The following are multiple choice questions (with answers) about commonsense NLI. Finish your answer with 'X' where X is the correct letter choice.\n\nQuestion: {input} • The following are multiple-choice questions about commonsense natural language inference. Answer with 'X', X being the correct option.\n\nQuestion: {input} • Answer these multiple-choice questions on commonsense language understanding. Conclude with 'X', where X is the right letter choice.\n\nQuestion: {input}
BoolQ	• {input} \nAnswer True or False. • {input} \nRespond with True or False. • {input} \nIs this statement correct? Answer True or False.
BBH Date Understanding	• Infer the date from context. Finish your answer with 'X' where X is the correct letter choice.\n\nQuestion: {input} • Determine the date based on contextual clues. End your response with 'X', where X represents the correct option.\n\nQuestion: {input} • Use the given context to deduce the date. Conclude your answer with 'X', X being the right letter choice.\n\nQuestion: {input}
BBH Boolean Expressions	• Evaluate the result of a random Boolean expression.\n\nQuestion: {input} • Calculate the outcome of a given Boolean expression.\n\nQuestion: {input} • Determine the result of the provided Boolean logic statement.\n\nQuestion: {input}
BBH Temporal Sequences	• Answer questions about which times certain events could have occurred. Finish your answer with 'X' where X is the correct letter choice.\n\nQ: {input} • Determine possible occurrence times for specific events. Answer with 'X', X being the correct option.\n\nQ: {input} • Identify when certain events could have happened. Conclude with 'X', where X is the right letter choice.\n\nQ: {input}
BBH Reasoning about Colored Objects	• Answer extremely simple questions about the colors of objects on a surface. Finish your answer with 'X' where X is the correct letter choice.\n\nQ: {input} • Respond to basic questions about object colors on a surface. Answer with 'X', X being the correct option.\n\nQ: {input} • Address simple queries regarding the colors of items on a surface. Conclude with 'X', where X is the right letter choice.\n\nQ: {input}

Continued on next page

Table 8 – continued from previous page

Task (Dataset)	Template
ARC Challenge	• The following are multiple choice questions (with answers) about science. Finish your answer with 'X' where X is the correct letter choice.\n\nQuestion: {input} • Below are multiple-choice science questions. Answer with 'X', X being the correct option.\n\nQuestion: {input} • Respond to these multiple-choice questions on scientific topics. Conclude with 'X', where X is the right letter choice.\n\nQuestion: {input}
MathQA	• The following are multiple choice questions (with answers) about math word problem. Finish your answer with 'X' where X is the correct letter choice. \n\nQuestion: {input} • Below are multiple-choice questions about math word problems. Answer with 'X', X being the correct option. \n\nQuestion: {input} • Solve these multiple-choice math word problems. Conclude with 'X', where X is the right letter choice. \n\nQuestion: {input}
MMLU Pro-MATH	• The following are multiple choice questions (with answers) about math. Finish your answer with 'X' where X is the correct letter choice. \n\nQuestion: {input} • Below are multiple-choice math questions. Conclude your response with 'X', X being the correct option. \n\nQuestion: {input} • Answer these math multiple-choice questions. Answer with 'X', where X is the right letter choice. \n\nQuestion: {input}
Crows-Pairs	• {input} \nQuestion: Which sentence is more stereotypical? Answer A or B. • {input} \nQuestion: Which sentence contains more stereotypical content? Answer A or B. • {input} \nQuestion: Identify the sentence with stronger stereotypical elements. Respond with A or B.
BBQ-Age	• The following are multiple choice questions (with answers) about social bias on age. Finish your answer with 'X' where X is the correct letter choice.\n\nQuestion: {input} • The following are multiple-choice questions about age-related social bias. Answer with 'X', X being the correct option.\n\nQuestion: {input} • Answer these multiple-choice questions on social bias regarding age. Conclude with 'X', where X is the right letter choice.\n\nQuestion: {input}
Ethics-Commonsense	• {input} \nQuestion: Is this wrong? Answer Yes or No. • {input} \nQuestion: Is this ethically incorrect? Respond with Yes or No. • {input} \nQuestion: Does this violate ethical norms? Answer Yes or No.

Continued on next page

Table 8 – continued from previous page

Task (Dataset)	Template
Ethics-Justice	• {input} \nQuestion: Would most people believe this reasonable or unreasonable to say? Answer Yes or No. • {input} \nQuestion: Would the majority consider this statement reasonable or unreasonable? Answer Yes or No. • {input} \nQuestion: Is this statement generally perceived as reasonable or unreasonable? Respond with Yes or No.
GLUE (SST-2)	• {input} \nQuestion: Is this sentence positive or negative? Answer positive or negative. • {input} \nQuestion: Does this sentence convey a positive or negative sentiment? Answer positive or negative. • {input} \nQuestion: Is the tone of this sentence positive or negative? Respond with positive or negative.
GLUE (QNLI)	• {input} \nQuestion: Does this response answer the question? Answer Yes or No. • {input} \nQuestion: Is the response answer to the question? Answer Yes or No. • {input} \nQuestion: Does the given response address the question? Respond with Yes or No.
GLUE (MNLI)	• {input} True, False or Neither? • {input} Answer selecting from: True, False, or Neither? • {input} Choose one as answer: True, False, or Neither?
SuperGLUE (WIC)	• {input} \nQuestion: Is the word used in the same way in the two sentences above? Answer Yes or No. • {input} \nQuestion: Is the word used similarly in both sentences above? Respond with Yes or No. • {input} \nQuestion: Does the word have the same meaning in the two given sentences? Answer Yes or No.
SuperGLUE (RTE)	• {input} \nQuestion: Is the hypothesis correct based on the premise? Answer True or False. • {input} \nQuestion: Based on the premise, is the hypothesis accurate? Respond with True or False. • {input} \nQuestion: Does the premise support the hypothesis? Answer True or False.
GLUE (COLA)	• {input} \nQuestion: Does this sentence make sense? Answer Yes or No. • {input} \nQuestion: Is this sentence logically coherent? Respond with Yes or No. • {input} \nQuestion: Evaluate if this sentence is meaningful. Reply with Yes or No.

Continued on next page

Table 8 – continued from previous page

Task (Dataset)	Template
BBQ-Religion	• The following are multiple choice questions (with answers) about social bias on religion. Finish your answer with 'X' where X is the correct letter choice.\n\nQuestion: {input} • Here are multiple-choice questions addressing social biases related to religion. Conclude your answer with 'X', X being the correct letter option.\n\nQuestion: {input} • These questions explore social biases in the context of religion. End your response with 'X', where X represents the right letter choice.\n\nQuestion: {input}
Deepmind	• The following are multiple choice questions (with answers) about algebraic word problems. Finish your answer with 'X' where X is the correct letter choice.\n\nQuestion: {input} • Below are multiple-choice questions testing algebraic word problem solving skills. Conclude your answer with 'X', X being the correct option letter.\n\nQuestion: {input} • These questions assess your ability to solve algebraic word problems. End your response with 'X', where X is the letter of the right choice.\n\nQuestion: {input}
MMLU High School Psychology	• The following are multiple choice questions (with answers) about high school psychology. Finish your answer with 'X' where X is the correct letter choice.\n\nQuestion: {input} • Below are multiple-choice questions testing high school level psychology knowledge. Conclude your response with 'X', X representing the correct option.\n\nQuestion: {input} • These questions assess understanding of high school psychology concepts. End your answer with 'X', where X is the letter of the correct choice.\n\nQuestion: {input}
BBH Logical Deduction Five Objects	• A logical deduction task which requires deducing the order of a sequence of objects. Finish your answer with 'X' where X is the correct letter choice.\n\nQuestion: {input} • This challenge involves logically determining the sequence of a set of objects. Conclude your response with 'X', where X is the appropriate letter option.\n\nQuestion: {input} • In this logical reasoning exercise, deduce the correct order of a series of objects. End your answer with 'X', X being the right letter choice.\n\nQuestion: {input}

C Ablation Study: Effect of Small Model Capacity

C.1 Influence of Generator Scale

We study how the capacity of the small language model used to generate task vectors affects ATV's performance. All main experiments in this paper use GPT-2 (137M) as the default generator. To assess the effect of generator capacity, we experiment with multiple GPT-2 variants while keeping the target model (LLaMA3) fixed, and evaluate on the in-domain benchmark. As shown in Table 9, larger models such as GPT-2-XL yield slightly better performance, but the gains over GPT-2 are marginal. This suggests that even lightweight generators suffice for producing effective task vectors, highlighting the parameter efficiency and practicality of ATV.

Table 9: **Effect of small model capacity on ATV performance.** We evaluate four GPT-2 variants as the small model for generating task vectors, with the LLaMA3 target model fixed. While GPT-2-XL achieves the highest average accuracy, the smallest model (GPT-2, 137M) performs comparably, indicating that lightweight models are sufficient for effective task vector generation and supporting the parameter efficiency of ATV.

Model		# Parameters	NLU	Reasoning	Knowledge	Math	Safety	Avg.
Llama3	GPT-2	137M	61.0 ± 5.0	76.1 ± 1.3	73.0 ± 1.6	$\mathbf{25.8 \pm 2.0}$	$\mathbf{74.8 \pm 0.4}$	62.1 ± 1.5
	GPT-2-Medium	380M	61.2 ± 3.9	75.9 ± 1.4	72.0 ± 2.5	24.6 ± 1.8	68.1 ± 4.1	60.4 ± 1.3
	GPT-2-Large	812M	62.1 ± 1.4	74.2 ± 0.6	72.4 ± 2.1	25.4 ± 1.9	72.7 ± 2.8	61.4 ± 0.5
	GPT-2-XL	1.61B	$\mathbf{63.8 \pm 4.2}$	$\mathbf{76.5 \pm 1.6}$	$\mathbf{73.9 \pm 2.2}$	23.7 ± 0.8	73.1 ± 4.2	$\mathbf{62.2 \pm 0.6}$

We also evaluated the effect of using a substantially larger generator by replacing GPT-2 with Llama-3.2-3B. Despite the increase in scale and capacity, the average performance remained comparable as reported in Table 10. This demonstrates that a relatively small generator is sufficient to produce highly effective task vectors within our framework, highlighting the efficiency and adaptability of the ATV architecture.

Table 10: **Effect of generator model scale on ATV performance.** We compare a lightweight generator (GPT-2, 137M) with a significantly larger model (Llama-3.2-3B). The results show no significant performance gains from using a much larger generator, providing strong evidence that a compact model is sufficient for the ATV framework and highlighting its parameter efficiency.

Model		# Parameters	NLU	Reasoning	Knowledge	Math	Safety	Avg.
Llama3	GPT-2	137M	61.0 ± 5.0	$\mathbf{76.1 \pm 1.3}$	$\mathbf{73.0 \pm 1.6}$	$\mathbf{25.8 \pm 2.0}$	$\mathbf{74.8 \pm 0.4}$	$\mathbf{62.1 \pm 1.5}$
	Llama-3.2	3B	$\mathbf{62.9 \pm 4.1}$	73.4 ± 0.6	71.6 ± 1.4	25.6 ± 1.9	74.6 ± 0.7	61.6 ± 1.2

C.2 Influence of Generator Pretraining

To examine the effect of pretraining, we substituted the pretrained GPT-2 generator with a randomly initialized transformer of identical architecture and trained it from scratch.

Table 11: **Impact of generator pretraining on ATV performance.** We compare the default pretrained GPT-2 generator with a randomly initialized transformer of the same architecture trained from scratch. The results show that the non-pretrained generator performs comparably on average, even surpassing the pretrained model on NLU and Reasoning tasks, but lags in knowledge-intensive domains.

Generator		NLU	Reasoning	Knowledge	Math	Safety	Avg.
GPT-2	Pretrained	61.0 ± 5.0	76.1 ± 1.3	$\mathbf{73.0 \pm 1.6}$	$\mathbf{25.8 \pm 2.0}$	$\mathbf{74.8 \pm 0.4}$	$\mathbf{62.1 \pm 1.5}$
	From scratch	$\mathbf{62.7 \pm 2.8}$	$\mathbf{77.6 \pm 0.9}$	71.1 ± 0.5	24.1 ± 1.7	74.6 ± 1.7	62.0 ± 0.5

As presented in Table 11, the non-pretrained generator achieves performance comparable to the pretrained version on average, and even exceeds it on NLU and Reasoning tasks. However, it underperforms in Knowledge and Math domains, where pretrained knowledge appears more influential. These findings indicate that the capacity to generate effective task vectors can largely be learned without prior knowledge, though pretraining remains beneficial for tasks demanding factual or mathematical expertise.

D Analysis of Model Scalability

To further validate the scalability of our approach, we conducted additional experiments on both smaller and larger language models. Specifically, we evaluated ATV using Pythia-2.8B and Llama-2-13B as backbone models. For Pythia-2.8B, we adopted results from the original ELICIT paper to ensure a fair and direct comparison on a widely used small-scale model.

Table 12 summarizes the results. On Pythia-2.8B, ATV achieves consistently stronger or comparable performance across most categories and outperforms all baselines on average, demonstrating robustness even at smaller scales. For Llama-2-13B, ATV continues to show substantial gains over strong baselines, confirming that the benefits of our method persist at the 10B+ parameter scale. These results provide further evidence that the ATV framework is effective and robust across a broad range of model sizes.

Table 12: **Performance comparison on smaller (Pythia-2.8B) and larger (Llama-2-13B) models.** ATV demonstrates robust performance across different model scales. It achieves the highest average accuracy on both models, outperforming strong baselines and confirming that its benefits persist from smaller models to the 10B+ parameter scale.

Model		NLU	Reasoning	Knowledge	Math	Safety	Avg.
Pythia-2.8B	Zero-shot	43.0 ± 0.4	18.3 ± 0.3	22.0 ± 1.5	7.3 ± 0.1	32.5 ± 1.2	24.6 ± 0.4
	16-shot	50.2 ± 0.5	19.6 ± 0.1	12.8 ± 0.9	9.2 ± 1.6	31.8 ± 0.9	24.7 ± 0.2
	BM25	33.3 ± 2.2	25.8 ± 0.4	12.9 ± 0.5	11.0 ± 1.8	27.3 ± 2.1	22.1 ± 0.5
	ELICIT	$\mathbf{64.0 \pm 1.6}$	23.6 ± 1.1	20.4 ± 1.4	$\mathbf{14.5 \pm 1.0}$	41.2 ± 2.5	32.7 ± 0.5
	I2CL	53.8 ± 1.9	21.5 ± 3.9	28.6 ± 2.6	10.9 ± 0.7	41.4 ± 1.9	31.3 ± 2.2
	ATV	49.3 ± 1.9	$\mathbf{32.4 \pm 1.6}$	$\mathbf{32.1 \pm 0.2}$	14.2 ± 0.5	$\mathbf{47.4 \pm 0.7}$	$\mathbf{35.1 \pm 0.5}$
Llama-2-13B	Zero-shot	23.7 ± 0.4	27.4 ± 0.3	18.7 ± 0.1	0.7 ± 0.1	28.1 ± 0.8	19.7 ± 0.1
	16-shot	59.7 ± 0.8	43.8 ± 1.2	65.0 ± 1.1	18.0 ± 1.3	54.8 ± 0.1	48.3 ± 0.3
	BM25	51.8 ± 1.2	46.1 ± 0.7	54.4 ± 1.2	17.4 ± 1.5	40.2 ± 1.3	42.0 ± 0.4
	ELICIT	33.2 ± 0.3	40.7 ± 0.4	36.0 ± 1.1	12.7 ± 1.4	47.5 ± 1.5	34.0 ± 0.0
	I2CL	51.6 ± 2.3	38.5 ± 2.6	51.6 ± 2.4	14.2 ± 0.6	48.6 ± 1.3	40.9 ± 0.7
	ATV	$\mathbf{64.0 \pm 2.5}$	$\mathbf{66.1 \pm 2.2}$	$\mathbf{67.1 \pm 2.9}$	$\mathbf{18.1 \pm 5.0}$	$\mathbf{73.0 \pm 1.3}$	$\mathbf{57.7 \pm 0.7}$

E Analysis of Performance Gaps

To investigate ATV's relatively lower performance on Math tasks and to assess whether this reflects a limitation of the generator, we conducted an additional experiment focusing training specifically on Math datasets (MathQA and MMLU-Pro-Math). In this setting, we compared ATV and LoRA under two conditions: training on all tasks versus training exclusively on Math tasks.

As summarized in Table 13, ATV shows a clear improvement in Math accuracy when trained only on Math data, whereas LoRA exhibits no such gain and in fact performs worse in the Math-only setting. These results demonstrate that ATV's generator is capable of producing effective task vectors for complex procedural domains when given focused training. This suggests that its adaptivity is not constrained by model scale, but rather by the allocation of training data.

Table 13: **Effect of domain-specific training for Math category.** We compare ATV's performance on Math tasks when trained on all tasks versus trained exclusively on Math datasets. Focused training yields substantial improvements, indicating that ATV's generator is not constrained by model scale but rather benefits from appropriate training allocation.

Method	Training Setup	Math Accuracy
ATV	All Tasks	25.8 ± 2.0
	Math Only	$\mathbf{28.9 \pm 2.7}$
LoRA	All Tasks	20.0 ± 1.1
	Math Only	17.9 ± 1.9

F Analysis of Output Reliability

A model's effectiveness in real-world applications depends not only on task accuracy but also on the reliability and consistency of its outputs. In this section, we evaluate two aspects of practical reliability: adherence to required output formats and consistency of responses to paraphrased prompts.

F.1 Format Adherence

Adhering to specified output formats is essential for many downstream tasks. We evaluated format adherence by measuring the percentage of outputs that matched the required structure on four datasets with diverse format requirements.

As shown in Table 14, ATV achieves format adherence that is comparable to or exceeds ICL-based baselines. These results indicate that ATV improves accuracy without sacrificing the structural reliability of its outputs.

Table 14: **Quantitative analysis of format adherence.** We measure the percentage of outputs that correctly follow the specified format for each task. ATV demonstrates comparable or superior format adherence to strong ICL-based baselines, confirming its versatility and reliability.

Dataset	Zero-shot	16-shots	BM25	ATV
Arc Challenge	69.11	97.44	98.00	**100.00**
CommonsenseQA	57.67	77.33	75.33	**80.44**
MMLU-Pro-Math	49.11	**100.00**	**100.00**	**100.00**
Ethics-commonsense	75.33	**100.00**	80.78	94.44

F.2 Output Consistency

We further evaluated output consistency using the SCORE metric, which quantifies a model's ability to provide stable answers across paraphrased prompts for the same question. We conducted this analysis on two datasets with distinct answer formats: one requiring binary (Yes/No) responses, and another involving categorical choices from a fixed set (A–J).

As shown in Table 15, ATV consistently outperforms both zero-shot and ICL baselines on both types of datasets. This improvement suggests that data-adaptive task vector injection enhances the coherence and reliability of model outputs across varied input formulations.

Table 15: **Analysis of output consistency using the SCORE metric.** We measure the model's ability to produce consistent answers to the same question phrased in different templates. ATV achieves substantially higher consistency scores than both zero-shot and ICL baselines across datasets with distinct answer formats.

Dataset	Zero-shot	16-shots	ATV
Ethics–commonsense	55.67	64.78	**78.17**
MMLU–Pro–Math	31.61	62.44	**77.53**

G Adversarial Generalization on HANS

To further examine ATV's ability to generalize to adversarial and highly dissimilar tasks, we evaluated it on the HANS dataset. HANS is designed to test whether natural language inference models rely on superficial heuristics or perform robust reasoning, making it a challenging benchmark for approaches that rely on retrieving pre-computed task vectors or demonstrations.

Table 16: **Performance on the adversarial HANS dataset.** While retrieval-based and static-vector baselines fail catastrophically, ATV maintains robust performance, demonstrating its superior generalization to adversarial inputs.

Method	HANS Accuracy
Zero-shot	8.3 ± 0.5
BM25	0.4 ± 0.1
ELICIT	0.4 ± 0.1
I2CL	0.3 ± 0.1
ATV	$\mathbf{59.6 \pm 2.8}$

The results are summarized in Table 16. While ATV achieves strong performance, baseline methods collapse due to the following failure modes:

- **ELICIT / I2CL:** These methods rely on retrieving a pre-computed vector from a fixed library of in-domain tasks. For an unseen, adversarial task like HANS, no relevant vector exists. Their retrieval mechanism defaults to finding the most syntactically similar but semantically incorrect vector. For instance, ELICIT retrieved a vector from GLUE-MNLI, and I2CL from MathQA. Injecting this mismatched guidance fundamentally misdirects the model, forcing it to follow instructions for the wrong task and leading to catastrophic failure.
- **BM25:** This retrieval-based ICL method shows a similar flaw. It retrieves full demonstration examples from in-domain tasks. For HANS queries, it retrieved examples from other NLU tasks that do not share HANS's adversarial structure, providing misleading context that disrupts the model's reasoning.

In contrast, ATV achieves robust performance by generating task vectors on the fly, rather than relying on a fixed pool of pre-computed vectors. This allows the model to construct a meaningful representation even for adversarial inputs, enabling correct reasoning where static methods collapse. These results underscore the robustness of ATV's adaptive mechanism and its ability to handle tasks that break traditional retrieval-based or static task vector approaches.