ADAPTIVE TASK VECTORS FOR LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

In-Context Learning (ICL) enables Large Language Models (LLMs) to perform tasks without parameter updates by conditioning on demonstrations in the prompt. Despite its success, ICL suffers from sensitivity to demonstration order, context length constraints, and computational inefficiency. Task vector-based approaches address these issues by compressing task information into a single vector. However, existing methods typically construct task vectors from fixed demonstration sets and reuse them across queries, limiting their ability to adapt to individual inputs and generalize to unseen tasks. We propose **Adaptive Task Vectors (ATV)**, a framework that dynamically generates task vectors conditioned on each input query. ATV employs a small language model whose last-token and mean-pooled representations are jointly projected into task vectors and injected into the frozen target LLM's hidden states. This enables adaptive model steering without explicit demonstrations during inference. Theoretically, ATV extends the static LoRA-style residual perspective via query conditioning and introduces interactions beyond Prefix-Tuning via direct hidden-state injection. Empirically, ATV achieves strong in-domain and unseen-task performance, including an adversarial shift, demonstrating effective adaptive model steering.

1 INTRODUCTION

Large Language Models (LLMs) have made remarkable progress in natural language processing, demonstrating impressive performance across various tasks. In-Context Learning (ICL) (Brown et al., 2020) has become a pivotal method for enhancing LLM performance, enabling models to effectively perform specific tasks by including demonstration samples in prompts without requiring additional training (Dong et al., 2022). However, ICL faces several limitations. Its performance varies considerably depending on the order and selection of demonstrations (Liu et al., 2021; Peng et al., 2024; Wang et al., 2023). In addition, context length constraints make it challenging to handle tasks requiring long-context reasoning or diverse demonstration sets (Li et al., 2024), and processing numerous demonstrations reduces computational efficiency (Kuratov et al., 2024).

To mitigate these issues, task vector-based approaches (Hendel et al., 2023; Liu et al., 2024; Li et al., 2025; Wang et al., 2025a) have attracted growing interest for improving the efficiency and robustness of ICL. Task vectors (Hendel et al., 2023) are vector representations that compress task-specific information, typically obtained from the hidden state of the last token in the prompt, or its processed variant. These vectors are integrated with the input query to modulate the model's output in a task-specific manner. Recent studies have utilized task vectors to effectively mitigate the limitations of conventional ICL (Yang et al., 2025; Huang et al., 2024). By compressing demonstrations into task vectors, these methods reduce prompt length and can mitigate sensitivity to demonstration ordering (Zhang et al., 2024; Li et al., 2025). As a result, they preserve the effectiveness of ICL while improving computational efficiency and consistency (Li et al., 2025).

Despite these advances, many existing task vector-based approaches derive task representations from fixed demonstrations and apply the resulting vectors across input queries (Hendel et al., 2023; Yang et al., 2025; Todd et al., 2023; Saglam et al., 2025). While recent retrieval-based methods provide a more flexible alternative by selecting task vectors according to query similarity (Wang et al., 2025a), the candidate vectors are still precomputed and remain fixed during inference. This motivates

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

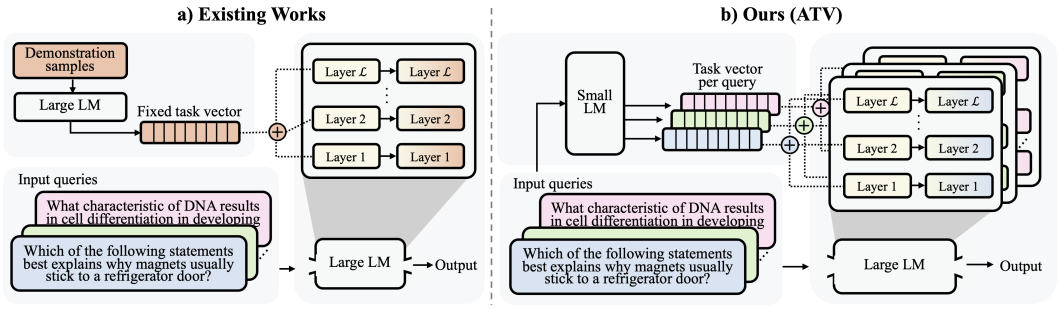

Figure 1: **Comparison between task vector methods.** a) Existing methods based on precomputed task vectors reuse demonstration-derived vectors across input queries. b) ATV generates a query-specific task vector from each input query using a small language model and injects it into the frozen target LLM, enabling the injected vector to adapt to individual inputs.

generating task vectors directly from each input query to provide adaptive guidance without relying on a fixed pool of precomputed vectors, particularly for unseen or shifted inputs.

Accordingly, we propose **Adaptive Task Vectors (ATV)**, a new framework for dynamically generating task vectors conditioned on each input query. ATV jointly projects the last-token and mean-pooled states of a small language model into task vectors for steering the frozen target LLM. Figure 1 illustrates the key difference between ATV and methods based on precomputed task vectors (Liu et al., 2024; Li et al., 2025; Wang et al., 2025a). These methods reuse or retrieve precomputed vectors across input queries, whereas ATV generates a query-specific vector to guide the frozen target model, enabling adaptation when fixed vectors may not align with unseen or shifted test queries. Moreover, our representation analysis in Section 4.5 shows that the vectors generated by ATV vary substantially across input queries, even within the same task. This diversity suggests that task-relevant information is not fully captured by a single fixed vector and motivates representations that adapt to each query.

In this paper, we establish the effectiveness of ATV through both theoretical and empirical evaluation. Theoretically, we connect ATV to LoRA-style residual adaptation and Prefix-Tuning (Hu et al., 2022; Li & Liang, 2021). ATV generates query-conditioned task vectors in activation space instead of relying on update parameters fixed after training, and steers the model through direct hidden-state injection rather than prefix-based conditioning. Section 3.3 formalizes these connections under matched assumptions. Empirically, we evaluate ATV on in-domain performance, generalization to unseen tasks, and ablations over model capacity, architecture, and injection configuration, showing that it outperforms baselines and yields interpretable insights into model design and injection behavior.

Our main contributions are as follows: **(1)** We introduce **ATV**, a framework that generates task vectors dynamically for each input query rather than relying on fixed or retrieved vectors computed from demonstration sets. This enables query-adaptive model steering through direct hidden-state injection into a frozen target LLM. **(2)** We provide a theoretical characterization of ATV in relation to parameter-efficient adaptation methods. Under the same rank budget and insertion layers, we show that a static rank-r affine generalization of ATV is equivalent to bias-augmented LoRA residual updates, while query-conditioned ATV extends this view by allowing the effective residual intervention to vary across queries. We also show that, under a relaxed linear attention approximation, ATV introduces cross-interaction terms beyond the prefix key-value pathway of Prefix-Tuning. **(3)** We empirically demonstrate that ATV achieves strong performance across diverse evaluation settings. ATV improves over strong baselines on in-domain tasks, generalizes effectively to unseen tasks, and remains robust under an adversarial task shift.

2 Related Work

In-Context Learning. ICL enables LLMs to perform new tasks by conditioning on a few input-output pairs without parameter updates (Brown et al., 2020), and has proven effective across diverse tasks through prompt engineering (Wei et al., 2022; Kojima et al., 2022; Zhou et al., 2022). However, it remains sensitive to demonstration selection and order (Zhao et al., 2021; Min et al., 2022), constrained by context length (Li et al., 2024), and costly at inference time (Kuratov et al., 2024).

Retrieval-based ICL methods mitigate some of these issues by dynamically selecting examples (Rubin et al., 2022), but still rely on explicit, token-based prompts. This motivates approaches that convey task information without explicit demonstrations while preserving the adaptability of ICL.

Vector-Based Approaches for Model Steering. Task-vector methods compress demonstration information into vectors for model steering (Hendel et al., 2023; Todd et al., 2023; Saglam et al., 2025; Li & Wang, 2026). I2CL (Li et al., 2025) constructs a context vector, while ELICIT (Wang et al., 2025a) retrieves query-relevant capability vectors from a precomputed library. Adaptive steering methods generate query-specific vectors for behaviors such as truthfulness or safety (Wang et al., 2025b; Weng et al., 2026), or rely on episodic retrieval (Do et al., 2025). ICR (Li et al., 2026) trains a query-conditioned router once and reuses it on unseen tasks, but uses fixed attention directions extracted from multi-domain ICL prompts. In contrast, ATV directly generates layer-wise task vectors from each input query, without retrieval or precomputed demonstration-derived vectors or directions.

Parameter-Efficient Adaptation and Hypernetworks. Parameter-efficient methods adapt frozen LLMs through learned prefixes (Li & Liang, 2021), low-rank updates (Hu et al., 2022), or hidden-state interventions (Wu et al., 2024). Related approaches incorporate input conditioning through instance-dependent prompts (Wu et al., 2022) and soft thought tokens generated by an assistant model and projected into the target LLM (Xu et al., 2025). Hypernetworks (Ha et al., 2017; De Brabandere et al., 2016; Karimi Mahabadi et al., 2021) provide a generative approach to adaptation, producing adaptation parameters (Phang et al., 2023) or steering vectors. HyperSteer (Sun et al., 2025) generates a single steering vector that is injected at one layer, conditioned on an explicit natural-language steering prompt and optionally on the base prompt and the internal activations of the target model. In contrast, ATV uses a small language model to generate layer-wise task vectors from the same input query fed to the target LLM, without a separate steering prompt. These vectors provide query-specific guidance through direct hidden-state injection without extending the target LLM's input sequence.

3 METHODOLOGY

3.1 PRELIMINARIES

Task Vector. In autoregressive models, the hidden state of the last token summarizes the input context and is used for next-token prediction (Radford et al., 2019; Muennighoff, 2022). Given a Transformer model with L layers, let $h^l(s) \in \mathbb{R}^{d_l}$ denote the hidden state of the last token at layer l for an input sequence s, where d_l is the hidden dimensionality at that layer.

Following prior work (Li et al., 2025; Wang et al., 2025a), we represent the demonstration context as a prompt P that concatenates N demonstrations. Specifically, let $\{(x_j^{\text{demo}}, y_j^{\text{demo}})\}_{j=1}^N$ denote N demonstration input-output pairs, where x_j^{demo} is a demonstration input and y_j^{demo} is its target output, and let x denote the input query. We write the prompt as $P = [(x_1^{\text{demo}}, y_1^{\text{demo}}), \ldots, (x_N^{\text{demo}}, y_N^{\text{demo}})]$.

The task vector $v_{\text{task}}^l \in \mathbb{R}^{d_l}$ at layer l is defined as the hidden state of the last token extracted from the demonstration prompt P, i.e., $v_{\text{task}}^l = h^l(P)$. This vector encodes task-specific information derived from the demonstrations in the prompt. To steer the model output in a task-specific direction, we inject this task vector into the hidden state of the last token computed from the input query x without demonstrations. Specifically, for each layer l, the modified hidden state $\tilde{h}^l(x)$ is computed as

$$\tilde{h}^l(x) = h^l(x) + \lambda v_{\text{task}}^l, \tag{1}$$

where λ is a scaling factor controlling the strength of the intervention. This formulation allows the task vector to modulate the model's behavior in a lightweight and interpretable manner. This injection mechanism underlies prior task-vector methods. We introduce **Adaptive Task Vectors (ATV)** to generate input-conditioned vectors by combining the last-token and mean-pooled representations of the small language model.

3.2 ATV: ADAPTIVE TASK VECTORS FOR LARGE LANGUAGE MODELS

ATV Framework. Let $x = [x_1, \ldots, x_T]$ be a tokenized input query with T tokens, and y the corresponding target output. Let $\mathcal{M}_{\text{large}}$ denote the frozen target model with L layers and hidden size d_l. ATV consists of a trainable generator $\mathcal{M}_{\text{small}}$ and a trainable expansion module f_θ.

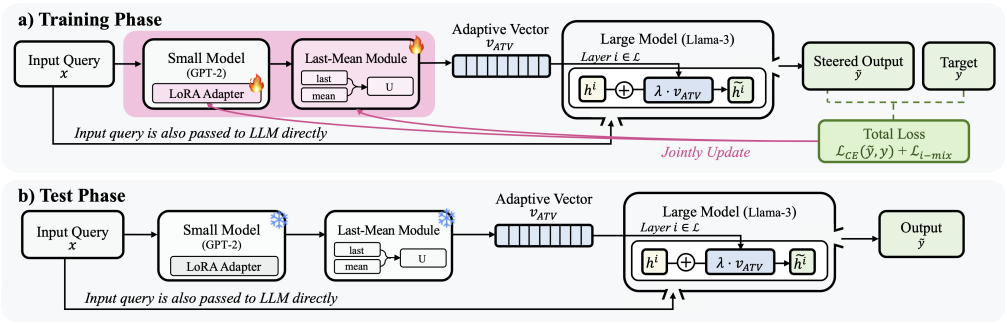

Figure 2: **Overview of the ATV framework.** a) During training, the small model's LoRA adapters and Last-Mean expansion module are optimized using target-model cross-entropy and auxiliary i-Mix regularization. b) During inference, both modules are frozen and generate an ATV from the input query, which is injected into the frozen target LLM.

Given x, let $h_t(x) \in \mathbb{R}^{d_s}$ denote the final-layer state of its t-th non-padding token in $\mathcal{M}_{\text{small}}$. While the last-token state is used for next-token prediction, prior work on decoder-based representations reports benefits from pooling across token states rather than relying on the last token alone (Wu & Xiong, 2020; Lee et al., 2025; Muennighoff, 2022). We therefore introduce Last-Mean, an expansion module f_θ that combines $z_{\text{last}} = h_T(x)$ and $z_{\text{mean}} = T^{-1}\sum_{t=1}^{T} h_t(x)$ through separate affine down-projections $D_{\text{last}}, D_{\text{mean}}$ and a shared up-projection U to produce layer-wise task vectors:

$$v_{\text{ATV}} = U((1-\rho)D_{\text{last}}(z_{\text{last}}) + \rho D_{\text{mean}}(z_{\text{mean}})) \in \mathbb{R}^{L \times d_l}. \tag{2}$$

Here, $\rho \in [0,1]$ is the fixed weight of the mean-pooled representation. Implementation details are provided in Appendix C.3. Let $v^l_{\text{ATV}} \in \mathbb{R}^{d_l}$ denote the task vector for layer l of the target model. The same query x is fed directly to $\mathcal{M}_{\text{large}}$, and the generated ATV modulates its last-token hidden state at each layer through the additive injection rule

$$\tilde{h}^l(x) = h^l(x) + \lambda v^l_{\text{ATV}} \tag{3}$$

Unless otherwise stated, injection is applied to all layers. We use a decoder-only GPT-2 model as the default generator (Radford et al., 2019) and evaluate encoder-based alternatives in Section 4.6.

Training and Inference. As illustrated in Figure 2, we train LoRA parameters ϕ in $\mathcal{M}_{\text{small}}$ and f_θ while keeping both pretrained backbones frozen. We optimize these parameters using target-model cross-entropy L_{CE} and an auxiliary i-Mix loss L_{iMix} that regularizes the generator's last-token representations based on dataset identity (Lee et al., 2020; Khosla et al., 2020). For a minibatch of B input-output pairs $\{(x_i, y_i)\}_{i=1}^{B}$ sampled from the training data distribution $\mathcal{D}$, we minimize

$$\min_{\phi,\theta} \left[\frac{1}{B}\sum_{i=1}^{B} \mathcal{L}_{\text{CE}}(\tilde{y}_i, y_i)\right] + \lambda_{\text{con}}\mathcal{L}_{\text{iMix}}, \qquad \tilde{y}_i = \mathcal{M}_{\text{large}}(x_i; v_{\text{ATV},i}), \tag{4}$$

where $v_{\text{ATV},i}$ denotes the layer-wise task vectors generated for x_i using Eq. 2, and λ_{con} weights the minibatch-level auxiliary loss.

For this auxiliary loss, let $z_i = z_{\text{last}}(x_i)$ and let $q_i \in \{0,1\}^{N_{\text{dataset}}}$ be a one-hot label identifying the source dataset of x_i among the N_{dataset} training datasets, rather than its target output y_i. Using a random minibatch permutation π and sampled mixing coefficients $\mu_i \in [0,1]$, we interpolate the representations and their dataset labels:

$$\tilde{z}_i = \mu_i z_i + (1-\mu_i) z_{\pi(i)}, \qquad \tilde{q}_i = \mu_i q_i + (1-\mu_i) q_{\pi(i)}. \tag{5}$$

The i-Mix loss matches a softmax over representation similarities to soft targets derived from dataset-label overlap. Full sampling, normalization, and loss details are provided in Appendix C.3.1. The interpolation is used only to compute $\mathcal{L}_{\text{iMix}}$, while task-vector generation in Eq. 2 and target-model cross-entropy use the original representations and input-output pairs. At inference time, ATV generates layer-wise task vectors for each new query and injects them into the frozen target model without using explicit demonstrations or dataset labels.

3.3 THEORETICAL ANALYSIS

Viewed as a query-conditioned residual intervention in activation space, ATV can be related to LoRA-style residual adaptation (Hu et al., 2022) and Prefix-Tuning (Li & Liang, 2021). In its static affine form, this perspective connects ATV to LoRA-style residual updates, while input conditioning allows ATV to generate query-specific injected vectors. Relative to Prefix-Tuning, ATV injects task information directly into hidden states rather than through prefix key-value representations, giving rise to additional cross-interactions.

We make these comparisons under the following analytical settings: a bias-augmented LoRA-style hidden-state residual update, consisting of a low-rank residual and an additive bias term, and a relaxed linear attention approximation for Prefix-Tuning. Specifically, we focus on addressing the following two questions: (1) How does ATV compare to LoRA-style residual adaptation in expressivity under the same rank budget and insertion layers, and how does query conditioning extend this comparison? (2) Under the relaxed linear attention approximation, what additional cross-interaction terms does ATV introduce beyond Prefix-Tuning?

LoRA-style comparison. To analyze our first question regarding comparative expressivity, we begin by defining the scope of our analysis. Our claim concerns the next-token prediction distribution, $F(x) = \mathrm{softmax}(W_{\mathrm{LM}} h_T^L)$, where $h_T^L \in \mathbb{R}^{d_L}$ denotes the hidden state at the last-token position T in the top layer L, and W_{LM} is the output projection matrix of the language model.

Since our analysis concerns the single-step next-token map $F(x)$, restricting attention to $F(x)$ suffices for the expressivity comparison in this section. Within this scope, and assuming the same insertion layers and per-layer rank budget r, we consider a static rank-r affine generalization of ATV, where a fixed latent vector v_0 determines a rank-r linear residual term and a bias at each insertion layer. The additive ATV intervention used in Section 3.2 corresponds to the $r = 0$, or bias-only, special case of this formulation. Proposition 3.1 characterizes the relationship between this generalization and bias-augmented LoRA residual updates under matched rank and placement assumptions.

Proposition 3.1 (Static Rank-r Affine ATV–LoRA Residual Equivalence)**.** *Let $h^l \in \mathbb{R}^{d_l}$ be the pre-intervention hidden state of the last token at layer l, and let $\tilde{h}^l$ and $\hat{h}^l$ denote the updated hidden states produced by static ATV and LoRA, respectively. For a rank budget r and a fixed nonzero vector v_0, let $\mathcal{F}_{\text{ATV-static}}(r)$ and $\mathcal{F}_{\text{LoRA}}(r)$ denote the classes of next-token maps $F : x \mapsto p(\cdot \mid x)$. The static rank-$r$ affine ATV generalization and bias-augmented LoRA residual updates induce the same class:*

$$\mathcal{F}_{\text{ATV-static}}(r) = \mathcal{F}_{\text{LoRA}}(r). \tag{6}$$

That is, for any static ATV update $\tilde{h}^l$, there exists a LoRA configuration yielding $\hat{h}^l = \tilde{h}^l$, and vice versa. The full proof is provided in Appendix A.

Remark 3.2 (Bias-only case)**.** *In the static setting above, the additive form of Eq. 3 corresponds to the $r = 0$ case of the affine generalization. Under the bias-augmented LoRA formulation considered in Appendix A, this matches the bias-only update case with $r = 0$ at the same insertion layers.*

Dynamic advantage of ATV. Proposition 3.1 establishes equivalence between this static affine generalization and bias-augmented LoRA residual updates under the same rank budget and insertion layers. Beyond this static setting, input-conditioned ATV replaces the fixed latent vector v_0 with a query-dependent vector $v(x)$, allowing the injected layer-wise vector $v_{\mathrm{ATV}}^l(x)$ to vary across queries, whereas LoRA applies the same learned low-rank transformation to the target hidden state across queries.

Prefix-Tuning comparison. To address our second question, we compare ATV to Prefix-Tuning under the relaxed linear attention approximation. We first write the standard attention operator in terms of its query, key, and value inputs as $\mathrm{Attn}(H^l W_q, C W_k, C W_v)$, where $H^l \in \mathbb{R}^{T \times d_l}$ denotes the layer-l hidden-state sequence on the query side, $C \in \mathbb{R}^{m \times d_l}$ denotes the key/value-side context representation of length m at layer l, 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 (He et al., 2022): $\mathrm{Attn}_{\text{prefix}} = \mathrm{Attn}(H^l W_q, [P_k; C W_k], [P_v; C W_v])$.

ATV injects v_{ATV}^l into the last-token residual state, which affects the subsequent attention block. For analytical tractability, we omit the intervening normalization and absorb λ into v_{ATV}^l, and write

$$\mathrm{Attn}_{\mathrm{ATV}} = \mathrm{Attn}\left((H^l + e_T (v_{\mathrm{ATV}}^l)^\top) W_q,\ (C + e_m (v_{\mathrm{ATV}}^l)^\top) W_k,\ (C + e_m (v_{\mathrm{ATV}}^l)^\top) W_v\right) \tag{7}$$

where $e_m = [0, \ldots, 0, 1] \in \mathbb{R}^{m \times 1}$ and $e_T = [0, \ldots, 0, 1] \in \mathbb{R}^{T \times 1}$ are one-hot vectors selecting the last-token positions. Thus, $e_T(v^l_{\text{ATV}})^\top$ and $e_m(v^l_{\text{ATV}})^\top$ add v^l_{ATV} only to the last rows of H^l and C, respectively.

Proposition 3.3 (Cross-Interaction Expansion of ATV)**.** *Let Attn_{ATV} and $\text{Attn}_{\text{prefix}}$ denote the attention outputs from ATV and Prefix-Tuning, respectively. Under the above abstraction and relaxed linear attention approximation, for a query hidden-state sequence $H^l \in \mathbb{R}^{T \times d_l}$, where T is the sequence length, ATV attention decomposes into a content-attention term, a prefix key-value interaction term analogous to the Prefix-Tuning pathway, and additional cross-interaction terms induced by additive hidden-state injection:*

$$\text{Attn}_{\text{ATV}} \approx T_{\text{content}} + T_{\text{prefix}} + \Delta_{\text{cross}}. \tag{8}$$

Comparison of Attn_{ATV} and $\text{Attn}_{\text{prefix}}$. Proposition 3.3 shows that T_{content} is the standard content-attention term, while T_{prefix} captures an ATV-induced key-value interaction analogous to the Prefix-Tuning pathway. Δ_{cross} collects additional mixed and query-side interactions induced by ATV. These terms arise from ATV-induced shifts in the query, key, and value inputs of the subsequent attention block. The full derivation is provided in Appendix B.

4 Experiments

We evaluate ATV on in-domain and unseen tasks and analyze task vector representations, injection strategies, and generator ablations. We also compare ATV with parameter-efficient tuning methods to complement our theoretical analysis. We closely follow ELICIT's (Wang et al., 2025a) experimental design. We evaluate on Llama-3-8B (Grattafiori et al., 2024) and Mistral-7B (Jiang et al., 2023), with I2CL (Li et al., 2025) as an additional baseline and a separate comparison to LoRA (Hu et al., 2022). Our code is available at `https://anonymous.4open.science/r/ATV-8B5A`.

4.1 Experiment Setup

Models and Baselines. Our primary models are Llama-3-8B and Mistral-7B. We compare ATV against (i) Zero-shot, (ii) 16-shot ICL, (iii) 16-shot BM25 retrieval-based ICL (Robertson & Zaragoza, 2009), (iv) ELICIT (Wang et al., 2025a), and (v) I2CL (Li et al., 2025). The demonstrations for (ii) are randomly sampled from the task training set, while those for (iii) are retrieved using BM25.

Datasets. We evaluate ATV on a diverse collection of 20 tasks spanning five categories, including Natural Language Understanding (Wang et al., 2018; 2019), Reasoning (Suzgun et al., 2022; Clark et al., 2018), Knowledge (Talmor et al., 2018; Mihaylov et al., 2018; Zellers et al., 2019; Clark et al., 2019), Mathematics (Amini et al., 2019; Wang et al., 2024), and Safety (Nangia et al., 2020; Parrish et al., 2021; Hendrycks et al., 2021a). We additionally evaluate ATV on a separate set of unseen tasks (Wang et al., 2018; Parrish et al., 2021; Ling et al., 2017; Hendrycks et al., 2021b; Suzgun et al., 2022; Lai et al., 2017; Sprague et al., 2024; Zhong et al., 2024) to assess its generalization ability.

Evaluation. We adopt the same evaluation strategy as ELICIT. Training uses the three ELICIT question templates with only the `A:` answer prefix, while evaluation combines these templates with three answer prefixes, yielding nine prompt variants per task. Task-specific instructions are prepended for all methods. Baseline descriptions, datasets, and experimental details are provided in Appendix C. We tune the scaling coefficient λ on Llama-3 validation data and use the selected value ($\lambda = 0.001$) across all backbones without further searching. A full sensitivity analysis is provided in Appendix D.

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 demonstration tokens in the prompt. ATV achieves the highest overall accuracy on both Llama-3 and Mistral, demonstrating strong performance across diverse task categories. Beyond task accuracy, we evaluate output reliability in terms of format adherence and consistency across prompt variations. Appendix E shows that ATV achieves strong performance on both metrics without compromising structural reliability, and maintains harmful compliance

Table 1: **In-domain performance comparison across five categories under Llama-3 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 all five categories and maintaining superior token efficiency over prompt-based methods. Error bars denote the standard deviation across three random seeds, and # Tokens denotes the number of prompt tokens provided to the target LLM.

Model		# Tokens	NLU	Reasoning	Knowledge	Math	Safety	Avg.
Llama-3	Zero-shot	108.3 ± 1.4	37.4 ± 1.3	31.6 ± 0.2	42.5 ± 1.5	14.0 ± 1.2	35.5 ± 1.5	32.2 ± 0.9
	16-shot	1883.8 ± 0.9	69.9 ± 2.0	56.0 ± 0.5	70.6 ± 1.2	26.7 ± 2.4	62.1 ± 0.5	57.1 ± 0.9
	BM25	2350.7 ± 24.9	68.6 ± 1.2	68.8 ± 0.2	69.5 ± 1.1	28.0 ± 2.9	56.7 ± 2.4	58.3 ± 1.1
	ELICIT	108.3 ± 1.4	47.4 ± 1.6	46.7 ± 0.2	60.6 ± 1.7	19.1 ± 1.7	49.9 ± 2.6	44.7 ± 1.4
	I2CL	108.3 ± 1.4	57.7 ± 1.5	48.5 ± 0.9	52.2 ± 3.0	17.9 ± 2.2	45.6 ± 2.4	44.4 ± 0.8
	ATV	108.3 ± 1.4	**77.7 ± 2.2**	**77.1 ± 2.3**	**75.8 ± 1.9**	**28.6 ± 0.3**	**73.8 ± 1.1**	**66.6 ± 0.4**
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	**59.6 ± 1.9**	**73.1 ± 1.0**	**76.7 ± 1.9**	**24.5 ± 0.4**	**72.1 ± 2.2**	**61.2 ± 1.3**

Table 2: **Performance on unseen tasks not included in the ATV training set, evaluated under Llama-3 and Mistral.** ATV achieves the highest average accuracy across all methods while using the same number of tokens as ELICIT and I2CL and substantially fewer tokens than prompt-based retrieval methods. Since DeepMind has unbalanced answer positions, we further evaluate it under Llama-3 with balanced positions, where BM25 and ATV achieve 25.8 and 26.0, respectively.

Model		# Tokens	GLUE CoLA	BBQ Religion	DeepMind	MMLU Psychology	BBH Five Objects	Avg.
Llama-3	Zero-shot	103.6 ± 47.7	53.8 ± 1.5	38.6 ± 1.1	17.5 ± 2.6	54.2 ± 0.3	17.1 ± 0.0	36.3 ± 0.6
	BM25	2502.8 ± 26.0	52.9 ± 6.3	64.6 ± 1.3	**30.7 ± 1.7**	83.0 ± 0.1	48.3 ± 0.0	55.9 ± 1.0
	ELICIT	103.6 ± 47.7	55.6 ± 0.2	45.0 ± 0.7	23.7 ± 3.4	70.0 ± 0.6	25.7 ± 0.0	44.0 ± 0.6
	I2CL	103.6 ± 47.7	30.0 ± 1.9	39.4 ± 3.1	23.5 ± 3.7	75.0 ± 1.0	27.3 ± 2.5	39.0 ± 0.4
	ATV	103.6 ± 47.7	**58.2 ± 5.6**	**83.5 ± 2.6**	28.4 ± 5.2	**83.2 ± 1.7**	**48.8 ± 0.7**	**60.4 ± 1.9**
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	**65.6 ± 4.8**	**81.2 ± 2.2**	**27.4 ± 5.0**	74.8 ± 1.5	**41.9 ± 3.6**	**58.2 ± 1.1**

comparable to that of the target model without ATV. We conduct one-sided paired t-tests to assess whether ATV's improvements are statistically reliable. Across the evaluated settings, the paired tests corroborate ATV's consistent gains over the strongest baseline. Results are reported in Appendix G.

4.3 Generalization to Unseen Tasks

To assess generalization, we evaluate ATV on unseen tasks held out from training. As shown in Table 2, ATV achieves the highest average accuracy under both Llama-3 and Mistral, indicating that its generated task vectors transfer beyond the training tasks without demonstration tokens. Under Llama-3, BM25 outperforms ATV on DeepMind. However, DeepMind has unbalanced answer positions. When they are balanced, BM25 drops from 30.7 to 25.8 and ATV from 28.4 to 26.0. This suggests that BM25's advantage partly reflects sensitivity to answer-option ordering, as detailed in Appendix F. Table 3 shows that ATV also attains the highest average accuracy under Llama-3 on four datasets from benchmarks not used in training, which involve a larger distributional gap.

We further evaluate adversarial generalization on HANS (McCoy et al., 2019), which is designed to reveal heuristic-driven failures in NLI models. Since most baselines rarely produce a valid HANS label, we report accuracy under both the standard protocol and constrained decoding to the two valid labels. As shown in Table 4, ATV achieves the highest accuracy in both settings, highlighting its robustness under an adversarial task shift. A detailed analysis is provided in Appendix H.

4.4 Layer-wise Analysis of Injection Strategies

We examine how injection depth influences performance for ATV and ELICIT. While ELICIT requires selecting a task-specific injection layer, we evaluate both methods under the same layer-restricted setting, restricting injection to the bottom, middle, or top third of the Transformer. ATV performs best with all-layer injection and remains robust under bottom-layer injection, whereas ELICIT performs best when injected into the top layers. Detailed results are provided in Appendix I.1. Figure 3

Table 3: **Performance on tasks with unseen benchmarks under Llama-3.** These datasets involve long-passage comprehension, narrative reasoning, and standardized-test logic problems, none of which appear in the training tasks. ATV achieves the highest average accuracy.

Method	RACE-High	MuSR	LSAT-AR	LSAT-LR	Avg.
Zero-shot	44.2 ± 3.0	23.1 ± 2.2	13.6 ± 1.8	25.5 ± 3.3	26.6 ± 2.5
BM25	75.4 ± 3.6	54.3 ± 3.0	21.0 ± 1.7	43.4 ± 4.2	48.5 ± 2.8
ELICIT	31.8 ± 2.8	19.3 ± 1.6	13.8 ± 2.0	21.4 ± 1.8	21.6 ± 1.7
I2CL	63.4 ± 6.1	51.0 ± 3.8	24.6 ± 0.2	32.2 ± 4.1	42.8 ± 3.3
ATV	$\mathbf{78.5 \pm 5.8}$	$\mathbf{56.9 \pm 3.1}$	$\mathbf{27.7 \pm 4.9}$	$\mathbf{44.7 \pm 2.0}$	$\mathbf{52.0 \pm 2.5}$

Table 4: **Performance on the adversarial HANS dataset under Llama-3.** ATV achieves the highest accuracy under both the standard protocol and constrained decoding to the two valid labels, demonstrating strong robustness under an adversarial task shift.

Method	Standard	Constrained
Zero-shot	8.3 ± 0.6	51.7 ± 1.8
BM25	0.4 ± 0.1	62.1 ± 2.5
ELICIT	0.4 ± 0.1	52.4 ± 2.6
I2CL	0.3 ± 0.1	52.7 ± 2.3
ATV	$\mathbf{51.6 \pm 4.6}$	$\mathbf{69.7 \pm 2.0}$

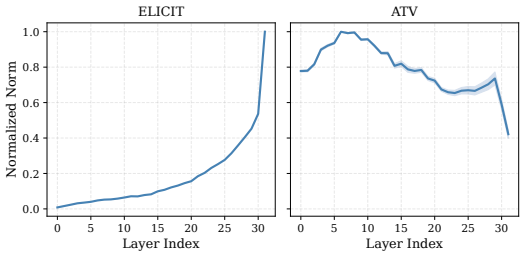

Figure 3: **Layer-wise normalized ℓ_2-norm profiles of ATV vectors and retrieved ELICIT capability vectors.** ELICIT's norms peak in the top layers, whereas ATV's peak in the lower layers, consistent with their performance trends.

visualizes the layer-wise normalized ℓ_2 norm of the injected vectors. ATV's norms peak in the lower layers, whereas ELICIT's peak in the top layers.

Together, these results suggest that ATV benefits from early-layer modifications that propagate through subsequent transformations, while ELICIT relies more on later-stage processing. Beyond layer-wise placement, we evaluate whether applying the injection across decoding steps benefits tasks that require multi-step reasoning. While the standard ATV injects the task vector only into the last token, we train both configurations on GSM8K (Cobbe et al., 2021) and find that extending injection throughout generation yields substantially higher accuracy, as detailed in Appendix I.2. These results suggest that maintaining task information during decoding can further support multi-step reasoning.

4.5 Visualizing Task Vector Distributions

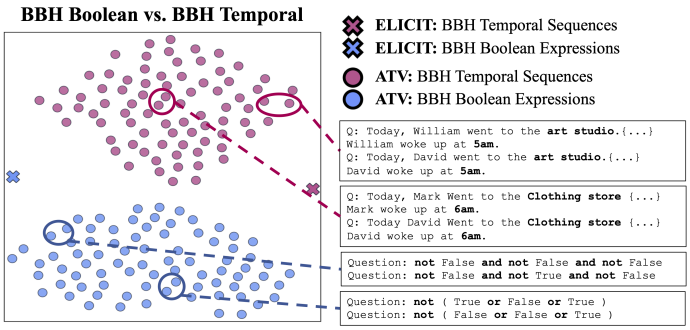

Figure 4: **t-SNE visualization of task vector distributions for two BBH tasks.** Dots represent query-specific ATV vectors concatenated across all layers, while crosses denote the corresponding retrieved ELICIT capability vectors. ATV vectors form structured clusters across queries, whereas the retrieved ELICIT vectors overlap within each displayed task.

We further analyze learned representations using t-SNE (van der Maaten & Hinton, 2008) to visualize query-specific task vectors and examine how ATV captures input variation. Figure 4 shows vectors for two BBH tasks alongside their ELICIT counterparts. ATV vectors exhibit structured within-task variation across queries. In contrast, ELICIT retrieves the same capability vector across queries, providing less query-level variation than ATV.

Table 5: **Generator ablations on capacity and architecture under Llama-3.** *Params* count only the generator backbone. Varying capacity shows that GPT-2 performs comparably to GPT-2-XL despite its smaller size, and comparing architectures shows that the decoder-based GPT-2 achieves the highest average accuracy. RoBERTa and Sentence-Transformer denote RoBERTa-base (Liu et al., 2019) and all-MiniLM-L6-v2 (Reimers & Gurevych, 2019; Wang et al., 2020), respectively.

Model	Params	NLU	Reasoning	Knowledge	Math	Safety	Avg.
(1) Capacity							
GPT-2	124M	77.7 ± 2.2	77.1 ± 2.3	75.8 ± 1.9	28.6 ± 0.3	73.8 ± 1.1	66.6 ± 0.4
GPT-2-XL	1.5B	77.4 ± 1.7	79.5 ± 1.5	74.5 ± 1.3	24.7 ± 3.1	73.8 ± 0.9	66.0 ± 1.3
(2) Architecture							
RoBERTa	125M	78.8 ± 2.9	77.1 ± 1.0	74.3 ± 0.8	27.3 ± 0.1	74.1 ± 1.8	66.3 ± 1.0
Sentence-Transformer	22.7M	76.4 ± 1.7	76.8 ± 1.5	75.7 ± 1.9	25.6 ± 1.3	75.3 ± 2.7	66.0 ± 0.4

4.6 Ablation Study

We study how the capacity of the small language model used to generate task vectors affects ATV's performance. As shown in Table 5, GPT-2 achieves performance comparable to GPT-2-XL despite its smaller size, indicating that lightweight generators suffice for producing effective task vectors. We also assess the effect of generator architecture by comparing decoder-based GPT-2 with encoder-based generators. Table 5 shows that encoder-based generators produce functional task vectors, whereas GPT-2 achieves higher average accuracy, consistent with prior findings that transfer is more stable when source and target architectures are similar (Su et al., 2022; Wu et al., 2023).

We further examine mean pooling and i-Mix regularization in Appendix J. Mean pooling improves unseen-task accuracy with similar in-domain performance, while i-Mix further improves ATV's overall average but reduces accuracy for I2CL and ELICIT. Finally, we compare ATV with static variants that replace input-conditioned generation with static task vectors in Appendix K. These variants underperform overall, indicating the importance of input-conditioned generation.

4.7 Efficiency Comparison with Baselines

We evaluate efficiency by measuring inference latency alongside predictive accuracy for ELICIT, I2CL, LoRA, and ATV, as shown in Figure 5. ATV remains faster than retrieval-based approaches like ELICIT. Moreover, ATV achieves markedly higher accuracy on both in-domain and unseen tasks compared to all baselines. These findings demonstrate that ATV delivers a favorable trade-off between practical inference efficiency and predictive accuracy among the compared methods.

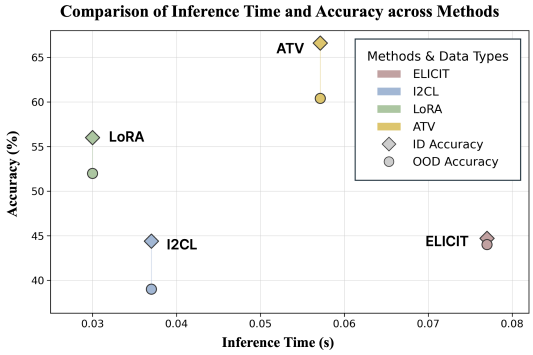

Figure 5: **Efficiency of ATV compared to existing methods in terms of inference time and accuracy.** ATV improves in-domain and unseen-task accuracy while requiring less inference time than ELICIT. See Appendix C.3 for measurement details.

5 Conclusion

We introduced Adaptive Task Vectors (ATV), a framework that generates task vectors conditioned on each input query and injects them into the hidden states of a frozen target LLM without updating its parameters. Our theoretical analysis relates ATV to LoRA-style residual adaptation and Prefix-Tuning. It shows that ATV extends static residual interventions with query-conditioned injected vectors, and that direct hidden-state injection induces interactions beyond prefix-based conditioning. Empirically, ATV achieves strong in-domain and unseen-task performance and remains robust under an adversarial shift, while maintaining token efficiency. These results support ATV as an effective approach to adaptive model steering.

AI USE STATEMENT

A large language model (LLM) was utilized solely as a writing assistant for limited tasks such as grammatical correction and enhancing textual clarity. All core scientific components of this work, including the conception of ideas, methodological development, theoretical and experimental work, were conducted entirely by the authors without any contribution from the LLM.

ETHICS STATEMENT

We do not anticipate ethical concerns specific to the experimental setting beyond those associated with the use of large language models. However, because ATV depends on its training data and input queries, poorly curated data may lead to unintended model behavior. Careful assessment of data quality and suitability for the intended application is therefore essential when applying this method.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide our code in the repository linked in Section 4. The datasets are described in Appendix C.1. Appendix C.3 details the experimental environment, training hyperparameters, random seeds, baseline configurations, and evaluation procedures, while Appendix C.4 provides the task-specific prompt templates.

REFERENCES

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2357–2367, 2019.

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.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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. *arXiv preprint arXiv:2110.14168*, 2021.

Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. Why can gpt learn in-context? language models implicitly perform gradient descent as meta-optimizers. *arXiv preprint arXiv:2212.10559*, 2022.

Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic filter networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 667–675, 2016.

Van Dai Do, Quan Hung Tran, Svetha Venkatesh, and Hung Le. Dynamic steering with episodic memory for large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 13731–13749, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.706. URL https://aclanthology.org/2025.findings-acl.706/.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, et al. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

David Ha, Andrew M Dai, and Quoc V Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2022.

Roee Hendel, Mor Geva, and Amir Globerson. In-context learning creates task vectors. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 9318–9333, December 2023. doi: 10.18653/v1/2023.findings-emnlp.624.

Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. Aligning ai with shared human values. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=dNy_RKzJacY.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021b. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

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.

Brandon Huang, Chancharik Mitra, Assaf Arbelle, Leonid Karlinsky, Trevor Darrell, and Roei Herzig. Multimodal task vectors enable many-shot multimodal in-context learning. *Advances in Neural Information Processing Systems*, 37:22124–22153, 2024.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 565–576, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.47. URL https://aclanthology.org/2021.acl-long.47/.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in neural information processing systems*, 33:18661–18673, 2020.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. *Advances in Neural Information Processing Systems*, 37:106519–106554, 2024.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pp. 785–794, 2017.

Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Nv-embed: Improved techniques for training llms as generalist embedding models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 79310–79333, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/c4bf73386022473a652a18941e9ea6f8-Paper-Conference.pdf.

Kibok Lee, Yian Zhu, Kihyuk Sohn, Chun-Liang Li, Jinwoo Shin, and Honglak Lee. i-mix: A domain-agnostic strategy for contrastive representation learning. *arXiv preprint arXiv:2010.08887*, 2020.

Feiyang Li and Yile Wang. DeCoVec: Building decoding space based task vector for large language models via in-context learning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 19096–19111, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.953. URL https://aclanthology.org/2026.findings-acl.953/.

Jiaqian Li, Yanshu Li, Ligong Han, Ruixiang Tang, and Wenya Wang. Train once, reuse everywhere: Generalizable implicit in-context learning by routing attention. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Gb2mqvRRa0.

Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. Long-context llms struggle with long in-context learning. *arXiv preprint arXiv:2404.02060*, 2024.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

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.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 158–167, 2017.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

Sheng Liu, Haotian Ye, Lei Xing, and James 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*, pp. 32287–32307, 2024.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

R. Thomas McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3428–3448, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1334. URL https://aclanthology.org/P19-1334/.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke 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*, pp. 11048–11064, 2022.

Niklas Muennighoff. Sgpt: Gpt sentence embeddings for semantic search, 2022. URL `https://arxiv.org/abs/2202.08904`.

Grigor Nalbandyan, Rima Shahbazyan, and Evelina Bakhturina. Score: Systematic consistency and robustness evaluation for large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pp. 470–484, 2025.

Nikita Nangia, Clara Vania, Rasika Bhalerao, and Samuel R Bowman. Crows-pairs: A challenge dataset for measuring social biases in masked language models. *arXiv preprint arXiv:2010.00133*, 2020.

Alicia Parrish, Angelica Chen, Nikita Nangia, Vishakh Padmakumar, Jason Phang, Jana Thompson, Phu Mon Htut, and Samuel R Bowman. Bbq: A hand-built bias benchmark for question answering. *arXiv preprint arXiv:2110.08193*, 2021.

Keqin Peng, Liang Ding, Yancheng Yuan, Xuebo Liu, Min Zhang, Yuanxin Ouyang, and Dacheng Tao. Revisiting demonstration selection strategies in in-context learning. *arXiv preprint arXiv:2401.12087*, 2024.

Jason Phang, Yi Mao, Pengcheng He, and Weizhu Chen. Hypertuning: Toward adapting large language models without back-propagation. In *International Conference on Machine Learning*, pp. 27854–27875. pmlr, 2023.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and trends® in information retrieval*, 4(1-2):1–174, 2009.

Ohad Rubin, Jonathan Herzig, and Jonathan 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*, pp. 2655–2671, 2022.

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*, pp. 6634–6663, 2025.

Alexandra Souly, Qingyuan Lu, Dillon Bowen, Tu Trinh, Elvis Hsieh, Sana Pandey, Pieter Abbeel, Justin Svegliato, Scott Emmons, Olivia Watkins, et al. A strongreject for empty jailbreaks. *Advances in Neural Information Processing Systems*, 37:125416–125440, 2024.

Zayne Sprague, Xi Ye, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. Musr: Testing the limits of chain-of-thought with multistep soft reasoning. In *International Conference on Learning Representations*, volume 2024, pp. 14670–14728, 2024.

Yusheng Su, Xiaozhi Wang, Yujia Qin, Chi-Min Chan, Yankai Lin, Huadong Wang, Kaiyue Wen, Zhiyuan Liu, Peng Li, Juanzi Li, et al. On transferability of prompt tuning for natural language processing. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3949–3969, 2022.

Jiuding Sun, Sidharth Baskaran, Zhengxuan Wu, Michael Sklar, Christopher Potts, and Atticus Geiger. Hypersteer: Activation steering at scale with hypernetworks. *arXiv preprint arXiv:2506.03292*, 2025.

Lintang Sutawika, Leo Gao, Hailey Schoelkopf, Stella Biderman, Jonathan Tow, Baber Abbasi, ben fattori, Charles Lovering, farzanehnakhaee70, Jason Phang, Anish Thite, Fazz, Aflah, Niklas Muennighoff, Thomas Wang, sdtblck, nopperl, gakada, tttyuntian, researcher2, Chris, Julen Etxaniz, Zdeněk Kasner, Khalid, Jeffrey Hsu, AndyZwei, Pawan Sasanka Ammanamanchi, Dirk Groeneveld, Ethan Smith, and Eric Tang. Eleutherai/lm-evaluation-harness: Major refactor, December 2023. URL https://doi.org/10.5281/zenodo.10256836.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*, 2018.

Eric Todd, Millicent L Li, Arnab Sen Sharma, Aaron Mueller, Byron C Wallace, and David Bau. Function vectors in large language models. *arXiv preprint arXiv:2310.15213*, 2023.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *International conference on machine learning*, pp. 6438–6447. PMLR, 2019.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

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*, 2025a.

Hanyu Wang, Bochuan Cao, Yuanpu Cao, and Jinghui Chen. Truthflow: Truthful llm generation via representation flow correction. *arXiv preprint arXiv:2502.04556*, 2025b.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788, 2020.

Xinyi Wang, Wanrong Zhu, Michael Saxon, Mark Steyvers, and William Yang Wang. Large language models are implicitly topic models: Explaining and finding good demonstrations for in-context learning. In *Workshop on Efficient Systems for Foundation Models @ ICML2023*, 2023. URL https://openreview.net/forum?id=HCkI1b6ksc.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37: 95266–95290, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

Zixuan Weng, Jinghuai Zhang, Kunlin Cai, Ying Li, Peiran Wang, and Yuan Tian. Finesteer: A unified framework for fine-grained inference-time steering in large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 18736–18756, 2026.

Chien-Sheng Wu and Caiming Xiong. Probing task-oriented dialogue representation from language models. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5036–5051, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.409. URL https://aclanthology.org/2020.emnlp-main.409/.

Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D Manning, and Christopher Potts. Reft: Representation finetuning for language models. *Advances in Neural Information Processing Systems*, 37:63908–63962, 2024.

Zhuofeng Wu, Sinong Wang, Jiatao Gu, Rui Hou, Yuxiao Dong, VG Vinod Vydiswaran, and Hao Ma. Idpg: An instance-dependent prompt generation method. In *Proceedings of the 2022 conference of the North American chapter of the association for computational linguistics: Human language technologies*, pp. 5507–5521, 2022.

Zijun Wu, Yongkang Wu, and Lili Mou. Zero-shot continuous prompt transfer: Generalizing task semantics across language models. *arXiv preprint arXiv:2310.01691*, 2023.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. Softcot: Soft chain-of-thought for efficient reasoning with llms. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23336–23351, 2025.

Liu Yang, Ziqian Lin, Kangwook Lee, Dimitris Papailiopoulos, and Robert Nowak. Task vectors in in-context learning: Emergence, formation, and benefit. *arXiv preprint arXiv:2501.09240*, 2025.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

Kaiyi Zhang, Ang Lv, Yuhan Chen, Hansen Ha, Tao Xu, and Rui Yan. Batch-icl: Effective, efficient, and order-agnostic in-context learning. *arXiv preprint arXiv:2401.06469*, 2024.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. In *International conference on machine learning*, pp. 12697–12706. PMLR, 2021.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. Agieval: A human-centric benchmark for evaluating foundation models. In *Findings of the association for computational linguistics: NAACL 2024*, pp. 2299–2314, 2024.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

A Proof of Proposition 3.1

A.1 Preliminaries

We analyze a generalized static rank-r affine variant of ATV. The additive intervention used in Section 3.2 corresponds to the $r = 0$ bias-only case of this formulation, while the proof below fixes v_0 to study the static setting.

Notation and scope. We analyze a Transformer with L layers. For an input sequence $x = (x_1, \ldots, x_T)$ let $h_t^l \in \mathbb{R}^{d_l}$ be the hidden state of token t *after* layer l ($l = 0$ denotes embeddings; $1 \le l \le L$). The model's output is the next-token distribution

$$F(x) \;=\; \mathrm{softmax}\big(W_{\mathrm{LM}}\, h_T^L\big),$$

where $W_{\mathrm{LM}} \in \mathbb{R}^{|\mathcal{V}| \times d_L}$ is the frozen LM head. Equality of *full sequence maps* is *not* required; only h_T^L matters for $F(x)$. For notational simplicity, we write $h^l := h_T^l$ for the last-token hidden state at layer l.

Insertion layers and rank budget. Let $S \subseteq \{1, \ldots, L\}$ be the set of layers at which updates are inserted, shared by both methods. Fix a rank budget $r \in \mathbb{Z}_{\ge 0}$. When $r = 0$, the low-rank term is absent, and only the bias term remains.

Static rank-r affine ATV for analysis. For each $l \in \mathcal{S}$, we choose *linear* expansion maps $A_l : \mathbb{R}^{d_s} \to \mathbb{R}^{d_l \times r}$, $B_l : \mathbb{R}^{d_s} \to \mathbb{R}^{d_l \times r}$, and $\beta_l : \mathbb{R}^{d_s} \to \mathbb{R}^{d_l}$. With a *fixed* nonzero vector $v_0 \in \mathbb{R}^{d_s}$ at inference, we write

$$\tilde{h}^l \;=\; h^l \;+\; \underbrace{B_l(v_0)\, A_l(v_0)^\top}_{\Delta W_l(v_0),\ \mathrm{rank} \le r} h^l \;+\; \beta_l(v_0), \quad l \in \mathcal{S},$$

and $\tilde{h}^l = h^l$ otherwise.

LoRA (same insertion layers). At the same layers, we fix matrices $W_{\downarrow,l} \in \mathbb{R}^{r \times d_l}$, $W_{\uparrow,l} \in \mathbb{R}^{d_l \times r}$, and an (optional) bias $b_l \in \mathbb{R}^{d_l}$, and set

$$\hat{h}^l = h^l + W_{\uparrow,l} W_{\downarrow,l} h^l + b_l, \qquad l \in S,$$

with $\hat{h}^l = h^l$ otherwise.

Throughout this proof v_0, A_l, B_l, β_l, $W_{\downarrow,l}$, $W_{\uparrow,l}$, and b_l are *fixed* ("static" setting).

A.2 Proposition

With the same insertion layer set S and rank budget r, let $\mathcal{F}_{\text{ATV-static}}(r)$ and $\mathcal{F}_{\text{LoRA}}(r)$ denote the corresponding classes of next-token maps $F : x \mapsto p(\cdot \mid x)$. Then,

$$\mathcal{F}_{\text{ATV-static}}(r) \;=\; \mathcal{F}_{\text{LoRA}}(r).$$

A.3 Proof

Step 1 LoRA ⇒ ATV (simulation). Fix the nonzero inference-time vector $v_0 \in \mathbb{R}^{d_s}$. Choose $u = v_0/(v_0^\top v_0)$, so that $u^\top v_0 = 1$. For every $l \in \mathcal{S}$, define the *linear* expansion maps, for any $z \in \mathbb{R}^{d_s}$, as

$$A_l(z) := (u^\top z)\, W_{\downarrow,l}^\top, \qquad B_l(z) := (u^\top z)\, W_{\uparrow,l}, \qquad \beta_l(z) := (u^\top z)\, b_l.$$

Each map is linear because it is the product of a scalar $u^\top z$ (linear in z) and a constant matrix or vector. Evaluating them at the fixed v_0 recovers the LoRA constants:

$$A_l(v_0) = W_{\downarrow,l}^\top, \qquad B_l(v_0) = W_{\uparrow,l}, \qquad \beta_l(v_0) = b_l.$$

Hence

$$\Delta W_l(v_0) h^l + \beta_l(v_0) = W_{\uparrow,l} W_{\downarrow,l} h^l + b_l.$$

Thus, $\tilde{h}^l = \hat{h}^l$ for every $l \in \mathcal{S}$. A forward induction from $l = 1$ to L then gives h_T^L equality, yielding $F_{\mathrm{ATV}}(x) = F_{\mathrm{LoRA}}(x)$.

Step 2 ATV ⇒ LoRA (simulation). Conversely, using the same fixed vector v_0 as in the static ATV, set for each $l \in \mathcal{S}$,

$$W_{\uparrow,l} := B_l(v_0), \qquad W_{\downarrow,l} := A_l(v_0)^\top, \qquad b_l := \beta_l(v_0).$$

Because A_l, B_l, and β_l are linear, $W_{\uparrow,l}$, $W_{\downarrow,l}$, and b_l are constants, and the rank of $W_{\uparrow,l}W_{\downarrow,l}$ does not exceed r. For these choices, the LoRA increment equals the ATV increment layer by layer, so the same induction as above gives $F_{\mathrm{LoRA}}(x) = F_{\mathrm{ATV}}(x)$.

Induction on l. Because the maps constructed in Steps 1–2 depend only on fixed parameters (they do not depend on the input sequence), the same x is fed through identical transformations in both networks. Specifically, after each update, both constructions produce the same hidden-state sequence entering the next Transformer layer. The last-token state is matched by the layer-wise simulation above, and all other token states are left unchanged. Since the subsequent Transformer sub-layers are shared and frozen, the next-layer hidden states are identical as well. Therefore, if $h^l(x)$ matches, so does $h^{l+1}(x)$, and the base case $l = 0$ is identical, yielding $h_T^L(x)$ equality for all x.

Conclusion. Steps 1 and 2 establish the bidirectional simulation under the same rank budget and insertion layer set; hence, the two function classes are identical. □

A.3.1 Remarks and Extensions

- **Dynamic advantage of ATV.** If the fixed vector v_0 is replaced by a query-dependent vector $v(x)$, the fixed expansion maps are evaluated at $v(x)$, so the effective update $\Delta W_l(v(x))h^l + \beta_l(v(x))$ can vary across queries. In contrast, LoRA applies the same learned low-rank transformation across queries, whereas ATV generates layer-wise vectors conditioned on each input query.
- **Bias-only case ($r = 0$).** The additive ATV update in Eq. 3 corresponds to the case $\Delta W_l \equiv 0$ in the affine formulation above. Here, the scaling coefficient λ is absorbed into the bias term, yielding $\beta_l(v_0) = \lambda v_{\mathrm{ATV}}^l$ for the implemented additive ATV. Under the bias-augmented LoRA formulation considered here, this matches the bias-only update case at the same insertion layers.
- **Scope.** The proof purposefully limits itself to equality of h_T^L and hence $F(x)$. No claim is made about intermediate token states or the full sequence map.

B Proof of Proposition 3.3

B.1 Preliminaries

Relaxed linear attention. Following Dai et al. (2022), we analyze attention under the relaxed linear form

$$\mathrm{Attn}(Q, K, V) \approx QK^\top V,$$

where the softmax operation and scaling factor are omitted for qualitative analysis.

For notational simplicity, we assume that the projected query, key, value, and hidden dimensions are all d_l.

Attention outputs from Prefix-Tuning.

$$\mathrm{Attn}_{\text{prefix}} = \mathrm{Attn}(H^l W_q, [P_k; CW_k], [P_v; CW_v])$$

- $H^l \in \mathbb{R}^{T\times d_l}$: layer-$l$ hidden-state sequence on the query side, where T is the sequence length.
- $C \in \mathbb{R}^{m\times d_l}$: key/value-side context representation at layer l.
- $P_k, P_v \in \mathbb{R}^{p\times d_l}$: p tunable prefix vectors for the keys and values.

In the decoder-only self-attention setting used in our experiments, $C = H^l$ and $m = T$. We keep H^l and C separate only to distinguish query-side and key/value-side contributions in the expansion.

Attention outputs from ATV. ATV is applied to the last-token residual state at the output of a Transformer layer, thereby affecting the subsequent attention block. For analytical tractability, we omit the intervening normalization and use l to index this subsequent block. For notational simplicity, we absorb the fixed scaling coefficient λ in Eq. 3 into v^l_{ATV} throughout this derivation without loss of generality:

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

- $v^l_{\mathrm{ATV}} \in \mathbb{R}^{d_l}$
- $e_m = [0, \ldots, 0, 1] \in \mathbb{R}^{m\times 1}$ and $e_T = [0, \ldots, 0, 1] \in \mathbb{R}^{T\times 1}$: one-hot vectors selecting the last-token positions. Thus, $e_T(v^l_{\mathrm{ATV}})^\top$ and $e_m(v^l_{\mathrm{ATV}})^\top$ add v^l_{ATV} only to the last rows of H^l and C, respectively.

B.2 Proposition

Under the formulation above and the relaxed linear attention approximation, ATV attention decomposes into a content-attention term, a prefix key-value interaction term, and additional cross-interaction terms induced by additive hidden-state injection:

$$\mathrm{Attn}_{\text{ATV}} \approx T_{\text{content}} + T_{\text{prefix}} + \Delta_{\text{cross}}.$$

B.3 Proof

Linear approximation of Prefix-Tuning.

$$\begin{aligned}
\mathrm{Attn}_{\text{prefix}} &= \mathrm{Attn}(H^l W_q, [P_k; CW_k], [P_v; CW_v]) \\
&= \mathrm{softmax}(H^l W_q [P_k; CW_k]^\top)[P_v; CW_v] \\
&\approx H^l W_q [P_k; CW_k]^\top [P_v; CW_v] \\
&= H^l W_q P_k^\top P_v + H^l W_q (CW_k)^\top CW_v
\end{aligned}$$

Linear approximation of ATV.

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

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

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

B.4 Analysis of Each Term in Attn_{ATV}

For each term, we report (i) the intuition behind the interaction and (ii) its role in the cross-interaction decomposition relative to standard Prefix-Tuning, treating the ATV-generated vectors P'_k, P'_v, and P'_q as induced key, value, and query shifts (see Table 6).

Key points of the proposition.

- **Content and prefix terms (T_1, T_2):** The ATV expansion contains the standard content-attention term T_1 and a prefix key-value interaction term T_2. We denote these terms as $T_{\text{content}} = T_1$ and $T_{\text{prefix}} = T_2$, respectively. Unlike standard Prefix-Tuning, however, T_2 is not parameterized by independently learned P_k and P_v; its shifts P'_k and P'_v are both induced by the same injected vector v^l_{ATV}.
- **Mixed interactions (T_3, T_4):** ATV combines ATV-induced key or value shifts with the original content representations, yielding mixed interaction terms that are not present in the standard Prefix-Tuning expansion under this approximation.
- **Query-side interactions (T_5, T_6, T_7):** The ATV residual intervention induces the query-side shift P'_q, yielding interactions with the original content key-value representations and ATV-induced key or value shifts.
- **Fully ATV-induced cross-interaction (T_8):** The term T_8 captures the interaction among the ATV-induced query, key, and value shifts.

The remaining six terms are collected as

$$\Delta_{\text{cross}} = T_3 + T_4 + T_5 + T_6 + T_7 + T_8.$$

Together, these terms show that ATV introduces additional cross-interaction structure beyond the content-attention and prefix key-value interaction terms under the relaxed linear attention approximation.

Table 6: Qualitative roles of ATV attention terms and their relation to Prefix-Tuning (PT).

Term	Qualitative role	Relation to Prefix-Tuning (PT) / Interaction role
T_1	**Base attention.** The query attends to the original content keys and returns the original content values.	**Shared content pathway.** This term corresponds to the standard content-attention component preserved in the ATV expansion.
T_2	**ATV-induced key-value pathway.** The original query attends to the ATV-induced key and returns the ATV-induced value.	**Prefix-like pathway.** This term plays the role most closely analogous to the Prefix-Tuning key-value pathway under the linearized approximation.
T_3	**ATV-induced key $\rightarrow$ content values.** An ATV-induced key reshapes the attention weights, while the returned information still comes from the original content values.	**Key-content mixed interaction.** This term couples an ATV-induced key shift with original content values, introducing a key-content interaction beyond the standard Prefix-Tuning expansion under this approximation.
T_4	**Content keys $\rightarrow$ ATV-induced value.** Original content keys determine the attention weights, while the returned value is induced by ATV.	**Content-value mixed interaction.** This term couples original content keys with an ATV-induced value shift, introducing a content-value interaction beyond the standard Prefix-Tuning expansion under this approximation.
T_5	**ATV-induced query $\rightarrow$ content keys and values.** The query-side representation is shifted by ATV before attending to the original context.	**Query-side interaction.** This term captures the interaction between the ATV-induced query shift and the original context.
T_6	**ATV-induced query and key $\rightarrow$ content values.** Both sides of the similarity computation are affected by ATV, while the returned information still comes from the original content values.	**Query-key cross-interaction.** This term captures a joint effect of ATV-induced query and key shifts while preserving the original content values as the output source.
T_7	**ATV-induced query and value with content keys.** Original content keys determine the attention weights, while the returned value is induced by ATV.	**Query-value cross-interaction.** This term captures the interaction between an ATV-induced query shift and an ATV-induced value shift under content-based key matching.
T_8	**Fully ATV-induced query-key-value pathway.** Query, key, and value are all induced by the ATV injection.	**Fully ATV-induced pathway.** This term captures the interaction among the ATV-induced query, key, and value shifts in the linearized expansion.

C Detailed Experimental Setting

Our experimental setup follows ELICIT (Wang et al., 2025a). Below, we provide detailed specifications.

C.1 Dataset List

Our main evaluations use the same 20 in-domain tasks and 5 unseen tasks as used in ELICIT. Tasks are categorized as follows:

- **Knowledge**: CommonsenseQA (Talmor et al., 2018), OpenBookQA (Mihaylov et al., 2018), HellaSwag (Zellers et al., 2019), BoolQ (Clark et al., 2019)
- **Reasoning**: Four subsets from Big-Bench Hard (BBH) (Suzgun et al., 2022) (BBH Boolean Expressions, BBH Date Understanding, BBH Reasoning about Colored Objects, BBH Temporal Sequences), ARC-Challenge (Clark et al., 2018)
- **Mathematics**: MathQA (Amini et al., 2019), MMLU-Pro Math (Wang et al., 2024)
- **Safety**: CrowS-Pairs (Nangia et al., 2020), BBQ Age (Parrish et al., 2021), ETHICS Commonsense, ETHICS Justice (Hendrycks et al., 2021a)
- **Natural Language Understanding (NLU)**: GLUE (SST-2, QNLI, MNLI) (Wang et al., 2018), SuperGLUE (WiC, RTE) (Wang et al., 2019)
- **Unseen**: GLUE CoLA, BBQ Religion, DeepMind, MMLU High School Psychology (Hendrycks et al., 2021b), BBH Logical Deduction Five Objects
  - Following ELICIT, we refer to AQuA-RAT (Ling et al., 2017) as DeepMind.
- **Datasets from unused benchmarks**: RACE-High (Lai et al., 2017), MuSR (Sprague et al., 2024), LSAT (LSAT-AR, LSAT-LR) (Zhong et al., 2024)

Since public test labels are unavailable, we re-evaluate CoLA, SST-2, and QNLI on labeled validation data. For CoLA, which is held out as an unseen task, we sample three mutually disjoint sets of 100 examples, one per seed, from the validation split. For SST-2 and QNLI, training examples are drawn from the official training split, while checkpoint selection uses validation data. We exclude all validation examples used for checkpoint selection and sample three mutually disjoint sets of 100 remaining examples, one per seed. Across all three datasets, evaluation examples are disjoint across seeds and do not overlap with data used for training, checkpoint selection, or demonstrations. All methods use the same evaluation examples within each seed under the same prompt and scoring protocols.

C.2 Detailed Description of Baselines

ELICIT. ELICIT (Wang et al., 2025a) constructs a library of task-specific capability vectors by processing demonstration prompts and extracting hidden states from the last token. Each vector is paired with an optimal injection layer determined through validation. During inference, ELICIT retrieves the most appropriate vector from the precomputed library and injects it into the target LLM at the predetermined layer. This eliminates demonstration tokens while preserving task guidance.

I2CL. I2CL (Li et al., 2025) compresses demonstration information into a single context vector, which is generated by averaging the activations of demonstration examples. This vector is then injected into the Transformer's residual streams during inference. Separately, the method calibrates a set of injection coefficients on the same demonstrations to modulate the vector's influence, allowing it to steer the model without explicit demonstration tokens in the input.

BM25 16-shot ICL. This baseline utilizes BM25 (Robertson & Zaragoza, 2009), a classical term-based retrieval method grounded in TF-IDF scoring, to identify and retrieve 16 demonstrations from the training set that exhibit high lexical similarity to the input query. The retrieved examples are concatenated into the prompt and provided to the model as in-context demonstrations. As a prompt-based method, its effectiveness is inherently tied to the model's context window capacity. Furthermore, its reliance on lexical matching means it is primarily designed to capture surface-level relevance rather than deeper semantic nuances between the query and demonstrations.

Key Differences with ATV. The primary difference lies in how task information is adapted to each query. I2CL uses a single, fixed context vector for all inputs within a given task. ELICIT selects the most suitable vector for a query from a fixed library of vectors, while BM25 adapts demonstrations but faces context length and ordering constraints. ATV generates a new, query-specific task vector dynamically for every individual input, enabling more fine-grained adaptation.

C.3 Implementation Details and Baseline Configurations

Common Setup. Experiments are run on NVIDIA RTX 4090 24GB GPUs. We follow ELICIT's prompt and scoring protocols for fair comparison across methods. For ATV and newly implemented baselines such as I2CL and LoRA, we use the same training data splits and conduct three runs with random seeds 42, 100, and 10 to account for stochastic variation and assess the stability of the results. Unless otherwise stated, accuracy results for all methods are reported as means $\pm$ sample standard deviations across the three runs. For all methods, we use the same evaluation examples, prompt variants, and scoring protocols within each seed. Zero-shot, 16-shot ICL, BM25, and ELICIT are re-evaluated using the baseline configurations described by ELICIT (Wang et al., 2025a). For training ATV and newly implemented baselines, we sample **90 examples per task** from the official training split of each in-domain task, excluding unseen tasks.

ATV evaluation uses up to 100 examples per dataset and 80 per BBH split. For in-domain and unseen results, we average over nine prompt variants before computing statistics across runs; overall scores average the five categories and five datasets, respectively. Far-OOD scores average four tasks within each run.

ATV. For both backbones, we jointly train the GPT-2 LoRA adapters and the expansion module f_θ for 15 epochs using AdamW with weight decay 10^{-5} and an effective batch size of 16. LoRA is applied to the attention and MLP projections with rank 16, scaling factor 32, and dropout 0.05. The learning rates for the adapters and the expansion module are 8×10^{-4} and 10^{-3} for Llama-3, and 5×10^{-4} and 1.5×10^{-3} for Mistral. We select the checkpoint by validation token accuracy.

The Last-Mean expansion module in Eq. 2 uses a 256-dimensional bottleneck and $\rho = 0.7$. The down-projections D_{last} and D_{mean} each map d_s-dimensional representations to the bottleneck space. The shared up-projection U maps the combined representation to Ld_l dimensions, with the output reshaped to $L \times d_l$. Encoder-based generators use the `[CLS]` representation and mean pooling over non-special input tokens.

For the default GPT-2 generator, ATV has approximately 36.44M trainable parameters: 2.36M in the LoRA adapters and 34.08M in the Last-Mean expansion module, including biases. Both pretrained backbones remain frozen. The parameter counts in Table 5 refer to the generator backbones only.

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 $\boldsymbol{\lambda = 0.001}$ throughout all experiments. In our implementation, the hidden size of the small model is $d_s = 768$ (GPT-2), and the large models (Llama-3-8B and Mistral-7B) use $d_l = 4096$ with $L = 32$ Transformer layers.

ELICIT. For the main comparisons in Tables 1 and 2, we use ELICIT's official configuration (Wang et al., 2025a). ELICIT retrieves task vectors from a precomputed capability library and additively injects the selected vector into the frozen LLM at its designated layer.

For additional analyses requiring modified ELICIT runs, such as the layer-restricted injection comparison, we follow the official implementation and default configuration. Each ELICIT task vector is constructed from 10 exemplars per task, each using a 16-shot prompt. Although overlap can reduce the total number of unique samples, our analysis confirms a minimum of 91 unique samples per task. To match this supervision scale, we use 90 training samples per task for ATV and newly implemented baselines.

I2CL. We adopt the official I2CL implementation (Li et al., 2025), modifying only the number of training epochs to 15 for consistency with other baselines. For in-domain tasks, I2CL constructs dataset-specific context vectors from task-specific demonstrations following its original setup. Under

the unseen-task protocol, task-specific demonstrations and context vectors from the held-out tasks are unavailable because using them would introduce task-level leakage.

Therefore, for unseen-task evaluation, we retrieve the most similar context vector from those obtained on in-domain datasets, based on cosine similarity between the input query and in-domain training prompts. We use shared injection coefficients learned from the in-domain tasks for this unseen setting to avoid task-specific coefficient selection on held-out tasks.

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, we use a learning rate of 4×10^{-4}, rather than the original 10^{-3}, because the latter yields substantially lower validation accuracy in our setting.

Latency measurement. For the inference latency comparison in Figure 5, all methods are evaluated on a single GPU under the same batch size and decoding setup, with latency averaged over the test set. ATV latency is the sum of the mean generator and target LLM times over 100 ARC-Challenge examples and nine prompt variants. The generator time includes tokenization and projection.

C.3.1 i-Mix Regularization

Following i-Mix (Lee et al., 2020) and supervised contrastive learning (Khosla et al., 2020), we regularize the generator by interpolating last-token representations and their dataset labels. For a batch of B examples, let $z_i = z_{\text{last}}(x_i)$ and let $q_i \in \{0,1\}^{N_{\text{dataset}}}$ be its one-hot label over the N_{dataset} training datasets. We draw a random permutation π and independent coefficients $\mu_i \sim \text{Beta}(\alpha_{\text{mix}}, \alpha_{\text{mix}})$, following hidden-space mixup (Zhang et al., 2017; Verma et al., 2019), and form

$$\tilde{z}_i = \mu_i z_i + (1-\mu_i) z_{\pi(i)}, \qquad \tilde{q}_i = \mu_i q_i + (1-\mu_i) q_{\pi(i)}. \tag{9}$$

The normalized anchor is $a_i = \tilde{z}_i / \|\tilde{z}_i\|_2$. With clean keys, we use $k_j = z_j/\|z_j\|_2$, $t_j = q_j$, and $\mathcal{J}_i = \{1, \dots, B\}$. With mixed keys, we use $k_j = \tilde{z}_j/\|\tilde{z}_j\|_2$, $t_j = \tilde{q}_j$, and $\mathcal{J}_i = \{1, \dots, B\} \setminus \{i\}$, excluding self-pairs. For temperature τ, the predicted distribution and soft target are

$$p_{ij} = \frac{\exp(a_i^\top k_j/\tau)}{\sum_{u \in \mathcal{J}_i} \exp(a_i^\top k_u/\tau)}, \qquad w_{ij} = \tilde{q}_i^\top t_j, \qquad r_{ij} = \frac{w_{ij}}{\sum_{u \in \mathcal{J}_i} w_{iu}}, \quad j \in \mathcal{J}_i. \tag{10}$$

We average over anchors with positive label overlap, $\mathcal{I} = \{i : \sum_{j \in \mathcal{J}_i} w_{ij} > 0\}$, giving

$$\mathcal{L}_{\text{iMix}} = -\frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}_i} r_{ij} \log p_{ij}. \tag{11}$$

The loss is zero when $\mathcal{I}$ is empty. Gradients flow through both anchors and keys. Mixing affects only the auxiliary objective; task-vector generation and cross-entropy use unmixed inputs. Dataset labels are used only during training.

We use $\lambda_{\text{con}} = 1$ and $\tau = 0.1$ for both backbones. The key type and α_{mix} are selected on validation data for each backbone, resulting in clean keys with $\alpha_{\text{mix}} = 0.5$ for Llama-3 and mixed keys with $\alpha_{\text{mix}} = 1$ for Mistral.

C.4 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` (Sutawika et al., 2023) and the `chain-of-thought-hub`[1]. To preserve comparability with ELICIT, we use its three question templates for each task without modifying their original wording. The full set of question templates used for each task is listed in Table 7.

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

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`

During training, we use all three question templates with only the `A:` answer prefix, yielding three training prompt formats per task. During evaluation, we use all three question templates with all three answer prefixes, yielding nine evaluation prompt variants per task. This setting is applied consistently across all baseline methods.

Table 7: **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}
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}

Continued on next page

Table 7 – continued from previous page

Task (Dataset)	Template
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}
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.

Continued on next page

Table 7 – continued from previous page

Task (Dataset)	Template
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.
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.

Continued on next page

Table 7 – continued from previous page

Task (Dataset)	Template
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.
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}

D SENSITIVITY TO THE SCALING COEFFICIENT

We determine the scaling coefficient λ by sweeping over several candidate values on the Llama-3 backbone and selecting the value that achieves the highest validation accuracy. The coefficient controls the strength with which the injected ATV vector influences the hidden representations of the target model. Table 8 reports the validation accuracy for different values, where $\lambda = 0.001$ yields the best performance on Llama-3. We therefore use this value for all backbones without further tuning, indicating that λ can be set efficiently through a small validation sweep on a single model.

Table 8: **Validation accuracy of ATV on Llama-3 across different values of the scaling coefficient λ.** The results illustrate how the strength of the injected task vector influences performance across five task categories, with $\lambda = 0.001$ providing the best overall accuracy. Each coefficient is evaluated at its best validation checkpoint using one fixed random seed.

λ	NLU	Reasoning	Knowledge	Math	Safety	Avg.
0.0001	73.5	75.9	72.6	21.3	78.6	64.4
0.0005	76.7	83.3	73.5	24.6	80.4	67.7
0.001	**81.0**	82.6	**75.4**	**32.6**	**82.1**	**70.7**
0.005	80.7	**83.5**	72.5	28.5	82.0	69.4

E 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 three aspects of practical reliability, including adherence to required output formats, consistency of responses to paraphrased prompts, and behavior on harmful requests.

E.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 9, 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 9: **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-shot	BM25	ATV
ARC-Challenge	69.11	97.44	98.00	**99.93**
CommonsenseQA	69.11	**100.00**	**100.00**	**100.00**
MMLU-Pro Math	49.11	**100.00**	**100.00**	**100.00**
ETHICS Commonsense	75.33	**100.00**	80.78	98.96

E.2 OUTPUT CONSISTENCY

We further evaluated output consistency using the Consistency Rate (CR) from SCORE (Nalbandyan et al., 2025), defined as the fraction of agreeing answer pairs among the nine prompt variants for each question, averaged across questions. 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 10, ATV consistently outperforms both zero-shot and ICL baselines on both types of datasets. This improvement suggests that query-conditioned task vector injection enhances the coherence and reliability of model outputs across varied input formulations.

Table 10: **Analysis of output consistency using the SCORE Consistency Rate (CR).** We measure the model's ability to produce consistent answers to the same question phrased in different templates. ATV achieves higher consistency scores than both zero-shot and ICL baselines across datasets with distinct answer formats.

Dataset	Zero-shot	16-shot	ATV
ETHICS Commonsense	55.67	64.78	**79.15**
MMLU-Pro Math	31.61	62.44	**68.02**

Beyond consistency, a reliable method should also maintain accuracy under prompt variation, so we report the lowest accuracy each method achieves across the nine evaluation prompt variants. ATV is trained with only the `A:` answer prefix, so two of the three prefixes used at evaluation are held out. As shown in Table 11, ATV achieves the highest accuracy on both in-domain and unseen tasks even under its weakest prompt, indicating that its gains persist across prompt formulations.

Table 11: **Worst-case accuracy across prompt variants.** For each method we report the lowest aggregate accuracy over the nine combinations of three question templates and three answer prefixes. ATV remains the most accurate method under its weakest prompt.

Method	In-domain	Unseen
Zero-shot	16.6 ± 0.7	13.6 ± 0.3
BM25	57.6 ± 1.1	53.4 ± 1.1
ELICIT	40.6 ± 1.1	35.0 ± 0.7
I2CL	42.7 ± 0.4	30.7 ± 1.1
ATV	$\mathbf{61.1 \pm 2.6}$	$\mathbf{57.5 \pm 2.0}$

E.3 Harmful Compliance

Since ATV modifies the hidden states of the target model, we further examine whether ATV affects the model's behavior on harmful requests. We evaluate on StrongREJECT (Souly et al., 2024), using the same frozen Llama-3-8B target and greedy decoding for all methods, and report the fine-grained judge score, where lower values indicate safer behavior. As shown in Table 12, ATV shows harmful compliance comparable to that of the unmodified target.

Table 12: **Harmful compliance measured on StrongREJECT.** Lower values indicate safer behavior. All methods use the same frozen target model and decoding setup. ATV shows harmful compliance comparable to that of the unmodified target.

Method	Harmful compliance ↓
Zero-shot	0.3171
LoRA	0.3225
ELICIT	0.3178
I2CL	0.3154
ATV	**0.3152**

F Sensitivity to Answer-Option Ordering

To examine the performance gap between ATV and BM25 on DeepMind, we analyze sensitivity to answer-option ordering. We permute the answer options so that correct answers are uniformly distributed across positions, while keeping the questions, model checkpoints, prompt variants, and retrieved demonstrations fixed. Both methods use the same permutations and unrestricted next-token prediction in both conditions.

As shown in Table 13, BM25 achieves higher accuracy under the original ordering, whereas ATV and BM25 achieve similar mean accuracy when answer positions are balanced. Table 14 shows that only 10.7% of correct answers are at option E in the original order. After balancing, BM25 favors options A and D and selects E for only 10.3% of its predictions, whereas ATV's E predictions increase from 10.9% to 18.7%. These results suggest that BM25's advantage on DeepMind partly reflects sensitivity to the arrangement of answer choices, even when the retrieved evidence is unchanged.

Table 13: **Effect of answer-option ordering on DeepMind with Llama-3.** BM25 is evaluated on the same question set as ATV, with unrestricted next-token prediction used in both conditions. We report the mean and sample standard deviation across three random seeds.

Method	Original order	Balanced order
BM25	30.7 ± 1.7	25.8 ± 2.7
ATV	28.4 ± 5.2	26.0 ± 3.8

Table 14: **Distribution of correct and predicted answer options on DeepMind with Llama-3.** We compare the original and balanced answer orders, aggregating predictions over three seeds and nine prompt variants. After balancing, BM25 still favors options A and D, whereas ATV's predictions become closer to uniform.

	A	B	C	D	E
Original order					
Correct answers	20.7	21.3	26.7	20.7	10.7
BM25 predictions	21.0	12.6	27.6	31.2	7.6
ATV predictions	30.2	17.2	21.6	20.1	10.9
Balanced order					
Correct answers	20.0	20.0	20.0	20.0	20.0
BM25 predictions	29.9	11.3	20.4	28.0	10.3
ATV predictions	26.8	16.8	19.4	18.2	18.7

G Statistical Significance Analysis

To assess whether the improvements achieved by ATV are statistically meaningful, we conduct one-sided paired t-tests on Llama-3 using results paired by random seed. For each setting, ATV was compared against the strongest-performing baseline, BM25, by pairing aggregate accuracies to test whether ATV achieves higher accuracy. The results show that ATV consistently improves over the strongest-performing baseline across all evaluated settings, with statistically significant or marginally significant gains on Llama-3. Table 15 presents the full statistical results.

Table 15: **One-sided paired t-test results comparing ATV with BM25 on Llama-3.** Accuracy is reported as the mean and sample standard deviation across three random seeds. Bold indicates $p < 0.05$, and underlining indicates $0.05 \leq p < 0.1$.

Model	ID ATV	ID Best Baseline	ID p-value	Unseen ATV	Unseen Best Baseline	Unseen p-value
Llama-3	66.6 ± 0.4	58.3 ± 1.1	**0.0032**	60.4 ± 1.9	55.9 ± 1.0	<u>0.0568</u>

These results indicate that ATV's improvements are consistent and statistically supported across the evaluated settings, even in cases where standard deviations appear larger. Overall, the paired tests provide additional evidence for the reliability of the observed gains across evaluation regimes.

H 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 precomputed task vectors or demonstrations.

We formulate HANS as a two-way NLI classification task with the valid labels `entailment` and `non-entailment`. All methods use the same label mapping and normalized answer-extraction rule. Outputs outside these two labels are treated as invalid and counted as incorrect.

As shown in Table 4, ATV maintains substantially higher accuracy than all baselines, which rarely produce a valid HANS label under the standard protocol. We consider the following possible failure modes:

- **ELICIT / I2CL:** These methods rely on retrieving a precomputed vector from a fixed library of in-domain tasks. For an unseen, adversarial task like HANS, a closely matched task vector for the HANS distribution may be unavailable. As a result, the selected guidance may come from tasks that appear related to HANS but do not match its adversarial NLI setting. For instance, ELICIT retrieved a vector from GLUE MNLI, and I2CL from MathQA. Injecting this mismatched guidance may steer the model toward an inappropriate task representation and cause outputs that do not match the required HANS labels.
- **BM25:** This retrieval-based ICL method shows a similar limitation. It retrieves full demonstration examples from in-domain tasks. For HANS queries, it retrieved examples from other NLU tasks that do not share the HANS adversarial structure, providing misleading context that can disrupt the model's reasoning and answer-format adherence.

In contrast, ATV generates task vectors on the fly, rather than relying on a fixed pool of precomputed vectors or retrieved demonstrations. This adaptive generation allows the model to form input-conditioned representations for adversarial inputs, providing a mechanism consistent with the performance differences observed in Table 4. These results highlight the robustness of ATV's adaptive mechanism on HANS and its ability to handle adversarial task shifts that challenge retrieval-based or static task-vector approaches.

To disentangle answer-format failures from task-level failures, we additionally evaluate all methods under a decoding constraint that restricts the output to the two valid HANS labels. As shown in Table 4, ATV achieves the highest accuracy under this constraint, while ELICIT and I2CL remain close to the Zero-shot level. Their failure on HANS therefore does not stem from the answer format alone, since the retrieved task vectors provide little benefit even when the label space is enforced.

I Additional Analysis of Injection Strategies

I.1 Analysis of Layer-wise Injection

We further examine the effect of injection depth by restricting ATV and ELICIT to the bottom, middle, or top third of the Transformer. Table 16 reports the full comparison with the all-layer setting.

ATV achieves its best performance with all-layer injection and shows only modest degradation under bottom-layer injection. In contrast, ELICIT performs best under top-layer injection. ELICIT's all-layer accuracy is substantially below its result in Table 1, where it uses a task-specific injection layer. For Figure 3, vector norms are first averaged across queries within each task, normalized by the task-specific maximum across layers, and then averaged across tasks. For ELICIT, the profile is computed from all layer components of the retrieved capability vector, rather than only from its selected injection layer.

Table 16: **Layer-wise performance comparison between ELICIT and ATV on Llama-3.** ATV achieves its best performance with all-layer injection and remains robust under bottom-layer injection, while ELICIT performs best under top-layer injection. Reported differences are measured relative to the all-layer injection setting.

Injected Layer	ELICIT		ATV	
	Avg. acc	**Diff.**	**Avg. acc**	**Diff.**
All Layers	28.5 ± 0.9	-	$\mathbf{66.6 \pm 0.4}$	-
Bottom ⅓	24.5 ± 0.9	-4.0	62.4 ± 2.2	**-4.2**
Middle ⅓	19.4 ± 0.2	-9.1	45.9 ± 1.7	-20.7
Top ⅓	$\mathbf{30.4 \pm 0.4}$	**+1.9**	34.4 ± 0.4	-32.2

I.2 Analysis of Multi-token Injection

We further examine whether applying the injection beyond the last token can improve performance on tasks that require multi-step reasoning. The standard ATV injects the task vector only into the hidden state of the last token of the input query at each layer. To test the effect of maintaining task information across decoding steps, we extend this injection to answer-token hidden states during training and to every decoding step during inference, and train and evaluate both configurations on GSM8K using cross-entropy alone. In both configurations, the task vector is generated once from the input query and is not recomputed during generation. For tasks whose outputs contain multiple tokens, the cross-entropy loss in Eq. 4 is computed as the mean of the token-level losses across the answer sequence, including the rationale and final answer, with prompt tokens excluded.

As shown in Table 17, applying the injection throughout generation yields substantially higher accuracy, indicating that preserving task information over the course of decoding can better support multi-step reasoning under this training setting.

Table 17: **Results on GSM8K comparing single-token and multi-token injection.** The multi-token setting, which applies the task vector throughout generation, achieves substantially higher accuracy on GSM8K.

Method	Accuracy
Zero-shot	30.3 ± 1.5
ATV (Single-token injection)	31.0 ± 2.0
ATV (Multi-token injection)	51.7 ± 3.1

J Additional Ablation Studies

J.1 Cumulative Ablation of Last-Mean and i-Mix

We examine the contributions of Last-Mean and i-Mix regularization to ATV. Starting from a last-token generator trained with cross-entropy, we first introduce Last-Mean and then add i-Mix regularization. As shown in Table 18, Last-Mean improves unseen-task accuracy while maintaining similar in-domain performance. The full configuration further improves the overall average and outperforms the last-token baseline in both settings.

Table 18: **Cumulative ablation of Last-Mean and i-Mix regularization in ATV with Llama-3.** Starting from a last-token generator, Last-Mean improves unseen-task accuracy while maintaining similar in-domain performance. Adding i-Mix increases in-domain accuracy and yields the highest overall average, while remaining above the last-token baseline on unseen tasks.

Configuration	In-domain	Unseen	Avg.
Last token	65.5 ± 1.2	59.8 ± 1.5	62.7 ± 1.3
Last-Mean	65.4 ± 0.6	$\mathbf{60.7 \pm 1.6}$	63.0 ± 0.5
Last-Mean + i-Mix	$\mathbf{66.6 \pm 0.4}$	60.4 ± 1.9	$\mathbf{63.5 \pm 1.0}$

J.2 i-Mix Regularization for Existing Methods

We apply i-Mix regularization to I2CL and ELICIT under the same in-domain and unseen-task protocols. For I2CL, the i-Mix loss is applied to the target LLM's final-layer hidden state of the last query token after context-vector injection. For ELICIT, it is applied to the retriever's `[CLS]` representations of input queries and library prompts. In both cases, the i-Mix loss is added to the original training objective. As shown in Table 19, i-Mix reduces average accuracy for both methods in both settings, suggesting that its effect is method-dependent.

Table 19: **Effect of i-Mix regularization on I2CL and ELICIT with Llama-3.** We compare each method with and without i-Mix under the same in-domain and unseen-task evaluation protocols. Adding i-Mix reduces average accuracy for both methods in both settings.

Method	In-domain	Unseen
I2CL	$\mathbf{44.4 \pm 0.8}$	$\mathbf{39.0 \pm 0.4}$
I2CL + i-Mix	39.7 ± 1.4	33.7 ± 2.5
ELICIT	$\mathbf{44.7 \pm 1.4}$	$\mathbf{44.0 \pm 0.6}$
ELICIT + i-Mix	43.3 ± 0.9	41.8 ± 0.1

K Ablation Study on Static Variants

To examine the role of input-conditioned generation in ATV, we conduct an ablation comparing it with two static task-vector variants that retain the same injection mechanism but replace the generator with directly optimized static vectors.

The first variant, Static Task Vector (Global), learns a single shared vector applied uniformly to all inputs, while the second, Static Task Vector (Per-task), learns a separate vector for each training task and applies it to examples from that task. For unseen queries, the Per-task variant uses the static vector associated with the training task of the top-ranked example retrieved by BM25. This design isolates the contribution of input conditioning from other factors.

Tables 20 and 21 show that ATV achieves substantially higher average accuracy than the static variants across both in-domain and unseen evaluations. These results indicate the importance of input-conditioned task-vector generation for ATV's performance.

Table 20: **In-domain evaluation of ATV and static variants under Llama-3.** The static variants directly optimize task vectors while retaining the same injection mechanism. ATV achieves the highest average accuracy, supporting the benefit of input-conditioned task-vector generation.

Model	NLU	Reasoning	Knowledge	Math	Safety	Avg.
Static Task Vector (Global)	57.7 ± 1.2	60.7 ± 0.9	64.0 ± 0.4	26.4 ± 1.3	53.6 ± 2.1	52.5 ± 0.1
Static Task Vector (Per-task)	52.3 ± 0.6	54.0 ± 0.5	61.8 ± 1.7	23.1 ± 1.5	48.5 ± 0.9	47.9 ± 0.9
ATV	$\mathbf{77.7 \pm 2.2}$	$\mathbf{77.1 \pm 2.3}$	$\mathbf{75.8 \pm 1.9}$	$\mathbf{28.6 \pm 0.3}$	$\mathbf{73.8 \pm 1.1}$	$\mathbf{66.6 \pm 0.4}$

Table 21: **Unseen task evaluation of ATV and static variants.** ATV generalizes substantially better than both static variants, confirming that input-conditioned task vectors improve adaptability beyond training domains.

Model	GLUE CoLA	BBQ Religion	DeepMind	MMLU Psychology	BBH Five Objects	Avg.
Static Task Vector (Global)	51.6 ± 2.5	63.9 ± 3.2	27.9 ± 2.2	$\mathbf{83.5 \pm 0.6}$	45.9 ± 0.4	54.5 ± 0.8
Static Task Vector (Per-task)	52.7 ± 5.0	46.1 ± 18.2	17.7 ± 4.5	65.4 ± 10.4	24.3 ± 0.9	41.3 ± 6.0
ATV	$\mathbf{58.2 \pm 5.6}$	$\mathbf{83.5 \pm 2.6}$	$\mathbf{28.4 \pm 5.2}$	83.2 ± 1.7	$\mathbf{48.8 \pm 0.7}$	$\mathbf{60.4 \pm 1.9}$