

PILOT: Planning via Internalized Latent Optimization Trajectories for Large Language Models

Haoyu Zheng¹, Yun Zhu², Yuqian Yuan¹, Bo Yuan¹,
Wenqiao Zhang^{1*}, Siliang Tang¹, Jun Xiao¹
¹Zhejiang University
²Shanghai AI Laboratory

Abstract

Strategic planning is critical for multi-step reasoning, yet compact Language Language Models (LLMs) often lack the capacity to formulate global strategies, leading to error propagation in long-horizon tasks. Our analysis reveals that LLMs possess latent reasoning capabilities that can be unlocked when conditioned on explicit plans from a teacher model; however, runtime reliance on external guidance is often impractical due to latency and availability constraints. To bridge this gap, we propose **PILOT** (Planning via Internalized Latent Optimization Trajectories), a non-invasive framework designed to internalize the strategic oversight of large models into intrinsic *Latent Guidance*. Instead of altering backbone weights, PILOT employs a lightweight Hyper-Network to synthesize a query-conditioned *Latent Guidance*. This vector acts as an internal steering mechanism, guiding the model’s representations toward optimal reasoning paths. Extensive experiments on mathematical and coding benchmarks demonstrate that PILOT effectively stabilizes reasoning trajectories, consistently outperforming strong baselines (e.g., +8.9% on MATH500) with negligible inference latency. Our code is available at: <https://github.com/Chihaya-Anon-chan/PILOT>

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in complex reasoning, largely driven by the Chain-of-Thought (CoT) paradigm (Wei et al., 2022). By decomposing problems into intermediate steps, CoT enables models to tackle tasks previously out of reach. However, reliable multi-step reasoning fundamentally relies on **Strategic Planning** to maintain global coherence across long-horizon trajectories. While compact models often possess the requisite domain knowledge, they frequently struggle with this strategic

oversight—the ability to formulate a high-level approach before execution. Without a global strategy, these models are prone to “myopic” generation, where minor errors in early reasoning steps cascade into significant deviations, a phenomenon known as error propagation.

Current approaches to mitigating these failures typically rely on **external scaffolding**. Techniques such as CoT prompting encourage decomposition but do not inherently instill a global planning mechanism. More advanced strategies employ “Teacher-Student” paradigms, where a larger model acts as an external guide to correct the smaller model’s trajectory during inference. While effective, this reliance on runtime external guidance is practically prohibitive: it introduces severe latency penalties, increases computational costs, and creates a strict dependency on the availability of superior models.

To address these latency constraints, recent work has explored internal adaptation. Yet existing methods struggle to preserve the balance between improving reasoning and maintaining broad general capability. **Static intervention** methods, such as LoRA (Hu et al., 2022) and ReFT (Wu et al., 2024a), perform static parameter-efficient adaptation: the learned changes are fixed after training and provide no explicit, per-instance strategic steering at inference time, often biasing the model toward brittle reasoning templates that fail on complex, heterogeneous instances. Similarly, **activation steering** techniques (Panickssery et al., 2024) typically rely on fixed steering vectors, which cannot adapt to the diverse logical demands across queries (Venhoff et al., 2025). Conversely, approaches that attempt to internalize reasoning directly into latent representations, such as Coconut (Hao et al., 2024), often require invasive training that can disrupt the model’s native representation manifold and induce catastrophic forgetting of pre-trained knowledge (Kirkpatrick et al., 2017; Luo et al., 2025). Furthermore, **guidance-**

*Corresponding author.

based strategies such as Soft CoT (Xu et al., 2025) can suffer from distribution mismatch between the assistant and the backbone, leading to embedding misalignment that limits effectiveness. Even compute-expanding approaches like Pause Tokens (Goyal et al., 2023) provide additional thinking budget but still lack a strategic anchor to stabilize long-horizon reasoning. Consequently, current literature lacks a method capable of internalizing guidance without compromising general capability, leaving a gap for a robust, planning-centric approach.

To bridge this gap, we propose **PILOT** (Planning via Internalized Latent Optimization Trajectories), a non-invasive framework designed to internalize the strategic oversight of large language models into intrinsic *Latent Guidance*. Rather than relying on runtime external calls or altering backbone weights, PILOT employs a lightweight Hyper-Network (Ha et al., 2016) to dynamically synthesize a query-conditioned guidance vector. This vector acts as an internal steering mechanism, effectively replicating the stabilizing effect of a high-level plan within the model’s deep semantic layers (Su et al., 2025). By synthesizing these signals strictly from the input query, PILOT ensures that the intervention is tailored to the specific logical requirements of each instance, guiding the model toward optimal reasoning paths without incurring retrieval latency.

Our contributions are summarized as follows:

- **Internalized Planning Paradigm.** We propose moving beyond static tuning to directly stabilize reasoning trajectories via intrinsic *Latent Guidance*, effectively internalizing the strategic foresight of larger models.
- **The PILOT Framework.** We introduce a novel architecture employing a Hyper-Network to synthesize query-conditioned guidance vectors, acting as a non-invasive internal steering mechanism to prime the model for complex reasoning.
- **Empirical Effectiveness.** Extensive experiments on mathematical and coding benchmarks demonstrate that PILOT consistently enhances the reasoning quality of compact LLMs (e.g., up to **+8.9%** gain on MATH500). Crucially, these gains are achieved with **near-zero extra latency**, stabilizing **single-path** reasoning trajectories.

2 Related Work

2.1 Evolution of Chain-of-Thought and Verification Strategies

Chain-of-Thought (CoT) prompting (Wei et al., 2022) has revolutionized LLM reasoning by decomposing problems into intermediate steps, yet its autoregressive nature remains susceptible to error propagation, where minor early deviations lead to cascading hallucinations (Dziri et al., 2023; Turpin et al., 2023). To mitigate this, decoding-level strategies like *Self-Consistency* (Wang et al., 2022) and *Tree of Thoughts* (Yao et al., 2023) introduce post-hoc verification by sampling multiple trajectories or performing tree search. Other methods explore reasoning rectification via backward verification (Xue et al., 2023) or adaptive self-correction (Wu et al., 2024b; Zhang et al., 2025). While effective, these methods treat the model as a black box and incur massive computational overhead—often $10\times$ to $50\times$ the original inference cost—making them impractical for low-latency applications. More importantly, they mask errors through external aggregation rather than fundamentally stabilizing the model’s internal reasoning process.

2.2 Parameter-Efficient Adaptation and Latent Reasoning

PEFT methods like *LoRA* (Hu et al., 2022) and *Prefix-Tuning* (Li and Liang, 2021) adapt models via static updates; however, their instance-agnostic nature lacks the granularity for the query-specific strategic guidance required by complex tasks (Sun et al., 2025; Choi et al., 2025). Paradigms for latent reasoning (Zhu et al., 2025), such as *Quiet-STaR* (Zelikman et al., 2024) and *Coconut* (Hao et al., 2024), attempt to bypass discrete bottlenecks by moving computation into the hidden space. Yet, these methods often require invasive training that disrupts the model’s native manifold, risking catastrophic forgetting of general knowledge. Similarly, *Pause Tokens* (Goyal et al., 2023) expand inference budgets but provide no strategic anchor against semantic drift. Finally, while *Soft CoT* (Xu et al., 2025) introduces thought vectors from auxiliary models, it often suffers from distributional mismatches that hinder cross-model alignment.

2.3 Representation Engineering and Activation Steering

A more recent paradigm, *Representation Engineering* (RepE) (Zou et al., 2023), aims to control be-

havior by editing model activations. Techniques like *ReFT* (Wu et al., 2024a) learn low-rank interventions on hidden states, while *Contrastive Activation Addition* (CAA) (Panickssery et al., 2024) extracts steering vectors by averaging activation differences from contrastive prompt pairs and injects them during inference. *Prototype-Based Steering* (Kayan and Zhang, 2025) retrieves task-specific exemplars at inference time to guide generation. However, these methods (Turner et al., 2023; Zhao et al., 2025; Tang et al., 2025) typically rely on static intervention vectors or heuristic retrieval from external databases, which may not generalize to novel or diverse queries. Furthermore, naive activation editing often causes “embedding shock”—a distributional shift that disrupts the model’s feature space and degrades stability (Zhou et al., 2025).

Our work, **PILOT**, addresses these limitations by employing a *Hyper-Network* to dynamically synthesize query-specific latent anchors. Unlike static PEFT or invasive latent reasoning, PILOT provides instance-level adaptivity without modifying backbone weights or incurring recurrent costs. By incorporating *Energy-Aligned Injection*, PILOT ensures manifold consistency, offering a non-invasive and scalable solution for stabilizing single-path reasoning trajectories.

3 Preliminaries

3.1 Notations

We denote the input query by x and the target output by $y = [r; a]$, where r is the rationale and a is the final answer. Let \mathcal{M}_ϕ be a frozen causal language model with parameters ϕ . For a given sequence, we denote token-level hidden states at layer l by $\{\mathbf{h}_i^{(l)}\}_{i=1}^n$, where $\mathbf{h}_i^{(l)} \in \mathbb{R}^d$ and d is the hidden size.

We use l^\dagger to denote the *pivot layer* where the latent anchor is injected. Let \mathcal{Q} be the index set of query tokens, and \mathcal{G} be the index set of guidance tokens in the verified expert prefix $[x; g_{\text{exp}}]$. We denote the extracted homogeneous target state by $\mathbf{z}^* \in \mathbb{R}^d$ and the predicted anchor by $\hat{\mathbf{z}} \in \mathbb{R}^d$. $\text{LN}(\cdot)$ denotes layer normalization, \odot denotes element-wise multiplication, and $\|\cdot\|_2$ denotes the l_2 norm.

3.2 Problem Formulation

Given an input query x , the goal is to generate $y = [r; a]$, where r is a rationale and a is the final answer. A causal LLM \mathcal{M}_ϕ models $P_\phi(r, a | x)$

autoregressively. In standard generation, the decoding trajectory is rigidly determined by the initial latent state induced by x . We instead consider a latent-anchored generation process by introducing an anchor vector $\mathbf{z} \in \mathbb{R}^d$ that conditions the autoregressive decoding:

$$P_\phi(r, a | x, \mathbf{z}) = \prod_t P_\phi(y_t | x, \mathbf{z}, y_{<t}) \quad (1)$$

Our objective is to learn an anchor adapter $\psi_\theta : x \mapsto \hat{\mathbf{z}}$ that predicts an instance-specific anchor $\hat{\mathbf{z}}$ from x , which is then injected at a pivot layer l^\dagger to stabilize the subsequent reasoning steps.

4 The PILOT Framework

In this section, we build on the formulation in Section 3.2 and present the key components of PILOT. We first describe how to extract the homogeneous target state \mathbf{z}^* (Section 4.1), then introduce the anchor adapter (Section 4.2) and injection mechanism (Section 4.3). Finally, we present the optimization objective (Section 4.4). An overview of the pipeline is illustrated in Figure 1.

4.1 Target State Extraction

To obtain high-fidelity supervision signals, we utilize a **Construct-and-Verify** pipeline (Figure 2) to derive the *homogeneous target state* \mathbf{z}^* .

Verification and Blind-Test. For each query x , we identify failure cases of the base model \mathcal{M}_ϕ . We then generate expert *Heuristic Guidance* (g_{exp}). To ensure g_{exp} provides strategic anchoring rather than a direct shortcut to the answer, we perform a **blind test**: if the model solves the problem given g_{exp} alone (without x), the sample is discarded. This ensures $\mathcal{D}_{\text{train}} = \{(x, g_{\text{exp}}, y^*)\}$ captures genuine strategic intent.

Homogeneous Target Projection. To ensure vector-space compatibility, we process the verified sequence $[x; g_{\text{exp}}]$ through the frozen reference model. The homogeneous target state vector \mathbf{z}^* is extracted at the output of the pivot layer l^\dagger by mean-pooling over the **guidance tokens** \mathcal{G} :

$$\mathbf{z}^* = \frac{1}{|\mathcal{G}|} \sum_{i \in \mathcal{G}} \mathbf{h}_i^{(l^\dagger)} \quad (2)$$

Extracting \mathbf{z}^* from \mathcal{G} allows the vector to encapsulate the “optimized” state reached by successful reasoning trajectories, serving as the ground-truth for alignment.

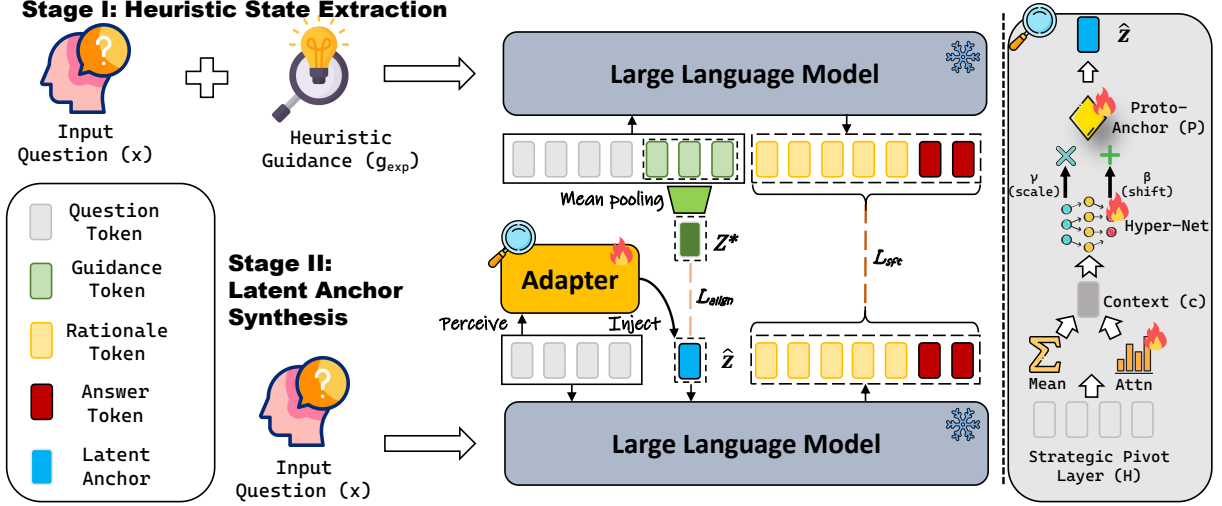


Figure 1: **The PILOT Framework Architecture.** (Top) Stage I: Heuristic State Extraction extracting the optimized latent state z^* from verified expert trajectories. (Bottom) Stage II: Latent Anchor Synthesis during inference predicting \hat{z} from query tokens. (Right) The Anchor Adapter modulates a **Proto-Anchor** \mathbf{P} via a Hyper-Network \mathcal{H}_θ and injects it into the backbone via energy-aligned injection.

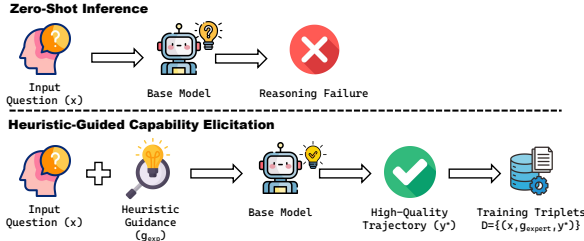


Figure 2: **Data Construction via Construct-and-Verify.** We filter for hard instances where the base model fails zero-shot but succeeds with expert guidance g_{exp} . These verified triplets (x, g_{exp}, y^*) form the training set \mathcal{D}_{train} .

4.2 Anchor Adapter Architecture

The anchor adapter ψ_θ serves as a perceiver that synthesizes an anchoring signal while strictly respecting causal constraints.

Dual-Channel Context Aggregation. The adapter captures query semantics through a residual fusion of global semantics (via Mean-Pooling) and salient entity features (via Attention-Pooling). Given question features \mathbf{H}_Q (i.e., the backbone hidden states of query tokens at the pivot depth, $\mathbf{H}_Q = \{\mathbf{h}_i^{(l^\dagger)}\}_{i \in Q}$), the context vector \mathbf{c}_Q is derived as:

$$\mathbf{c}_Q = \underbrace{\text{MeanPool}(\mathbf{H}_Q)}_{\text{Global Intent}} + \underbrace{\sum_{i \in Q} \text{softmax}(\mathbf{w}_a^T \mathbf{h}_i^{(l^\dagger)}) \mathbf{h}_i^{(l^\dagger)}}_{\text{Salient Entities}} \quad (3)$$

where $\mathbf{w}_a \in \mathbb{R}^d$ is a learnable attention query. This design ensures \mathbf{c}_Q captures both holistic sentence structure and key logical entities.

Proto-Anchor Modulation. We introduce a learnable **Proto-Anchor** vector $\mathbf{P} \in \mathbb{R}^d$ (a.k.a. a proto-thought prior), acting as a global prior of the reasoning manifold. Crucially, rather than random initialization, we **warm-start** \mathbf{P} with the global centroid of target states $\mathbb{E}[z^*]$ computed over a subset of training data.

We employ a **Hyper-Network** \mathcal{H}_θ to predict channel-wise FiLM modulation coefficients $[\gamma; \beta]$ from \mathbf{c}_Q :

$$[\gamma; \beta] = \mathcal{H}_\theta(\mathbf{c}_Q) \quad (4)$$

$$\mathbf{v}_{raw} = \gamma \odot \mathbf{P} + \beta \quad (5)$$

To ensure training stability, \mathcal{H}_θ is initialized as an **Identity Prior** (i.e., weights ≈ 0 , bias $\gamma = 1, \beta = 0$). This forces the optimization to start from the stable global prototype \mathbf{P} and progressively learn instance-specific deviations.

4.3 Anchor Injection Mechanism

Delayed Visibility Masking. To integrate the anchoring signal non-invasively, we append a placeholder token to the query. Concretely, the placeholder is appended to the end of the input context and is not part of the textual output; we assign it a dedicated learnable embedding and intervene only on its hidden state, leaving all other token states unchanged. For all layers $l < l^\dagger$, this token is isolated via a causal mask. At the pivot layer l^\dagger , its

visibility is enabled, and its hidden state is replaced by $\hat{\mathbf{z}}$.

Energy-Aligned Injection. To reconcile directional anchoring with norm-sensitive attention mechanisms, we propose an **Energy-Aligned Injection** that decouples semantic orientation from physical intensity:

$$\hat{\mathbf{z}} = \text{Softplus}(\alpha) \cdot \sigma_{\text{ctx}} \cdot \frac{\text{LN}(\mathbf{v}_{\text{raw}})}{\|\text{LN}(\mathbf{v}_{\text{raw}})\|_2} \quad (6)$$

where $\sigma_{\text{ctx}} = \text{Mean}_{i \in \mathcal{Q}} \|\mathbf{h}_i^{(t)}\|_2$ adapts the injection scale to the current context energy (computed over query tokens, as in our implementation). α is a zero-initialized learnable scalar. To prevent the injection from overwhelming intrinsic backbone features, we apply a regularization penalty if the gating scale $\text{Softplus}(\alpha)$ exceeds a threshold $\tau = 2.0$.

4.4 Optimization Objectives

The framework is optimized via a two-phase curriculum.

Phase 1: Latent Alignment. We freeze the backbone and minimize the cosine distance between the predicted anchor vector and the homogeneous target: $\mathcal{L}_{\text{align}} = 1 - \cos(\hat{\mathbf{z}}, \mathbf{z}^*)$. This phase grounds the adapter in the expert reasoning manifold.

Phase 2: Anchored Fine-Tuning. We keep the backbone frozen and optimize the adapter components (including the hyper-network, \mathbf{P} , and the gate scalar) using the SFT loss \mathcal{L}_{SFT} , while retaining the alignment loss and the gate regularization as structural constraints:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{SFT}} + \lambda_1 \mathcal{L}_{\text{align}} + \lambda_2 \mathcal{L}_{\text{gate}} \quad (7)$$

where $\mathcal{L}_{\text{gate}} = \max(0, \text{Softplus}(\alpha) - 2.0)^2$. We set $\lambda_1 = 0.1$ and $\lambda_2 = 0.01$, anchoring the signal to the expert manifold while preventing "embedding shock" via norm constraints.

5 Experiments

5.1 Experimental Setup

Datasets & Filtering. We evaluate PILOT across **Mathematics** and **Code Generation**. A core component is our *Construct-and-Verify* pipeline, which distills training sets into compact, model-specific subsets ($\mathcal{D}_{\text{train}}$). As shown in Table 2, we use a *Boundary Filter* for MATH (Hendrycks et al.,

2021) to capture the reasoning frontier, and a *Refinement Filter* for MBPP (Austin et al., 2021) to align structural logic. Performance is evaluated on MATH500 (Hendrycks et al., 2021) (primary), AIMO Val. (83 AMC-level problems) (Team, 2024), and GSM8K (robustness) (Cobbe et al., 2021) for math; and HumanEval (Chen et al., 2021) and MBPP-Test (Austin et al., 2021) for coding.

Base Models. We conduct experiments on three instruction-tuned models: Qwen2.5-1.5B-Instruct, Qwen2.5-7B-Instruct (Yang et al., 2024), and Llama-3.1-8B-Instruct (Grattafiori et al., 2024). For target construction, we utilize DeepSeek-V3.1 as the Expert Model.

Baselines. We compare PILOT against representative methods. To validate PILOT’s ability to improve **single-path reasoning trajectories**, we select baselines that, like PILOT, aim to enhance model performance within a single forward pass. For fairness, trainable baselines are fine-tuned using the same filtered subset $\mathcal{D}_{\text{train}}$.

- **Discrete Prompting:** Standard **Zero-shot CoT**, representing the model’s base reasoning capacity without any external intervention.
- **Static Tuning:** Parameter-efficient methods that learn **fixed adaptation** for the target domain, including **LoRA** (Hu et al., 2022) and assistant-guided **Soft CoT** (Xu et al., 2025).
- **Latent Intervention:** Methods that directly manipulate internal states, including **ReFT** (Wu et al., 2024a) and **CAA** (Panickssery et al., 2024). **CAA** modifies activations via static steering vectors extracted from contrastive reasoning pairs. We also compare against compute-expanding **Pause Tokens** (Goyal et al., 2023) and **Coconut** (Hao et al., 2024) to benchmark against implicit latent reasoning paradigms.

Implementation Details. Experiments are performed on NVIDIA H20 GPUs. The adapter is trained via a two-stage curriculum: Phase 1 (Alignment) with a learning rate of $1e - 4$, and Phase 2 (Anchored SFT) with $2e - 5$, using regularization weight $\lambda = 0.1$. Training epochs are adapted to data scale: we train Math models for 3 epochs per phase, while Coding models undergo prolonged training (10 epochs for Alignment, 8 for SFT) to ensure convergence. For evaluation, we employ

Table 1: **Main Results.** Pass@1 accuracy (Mean \pm Std over 5 runs). **PILOT** (blue rows) consistently outperforms baselines across all model scales (1.5B, 7B, 8B), maintaining robustness even on saturated tasks like GSM8K.

PANEL A: MATHEMATICAL REASONING									
Method	Qwen2.5-1.5B			Qwen2.5-7B			Llama-3.1-8B		
	MATH	AIMO	GSM	MATH	AIMO	GSM	MATH	AIMO	GSM
Zero-shot CoT	43.20	22.89	66.49	71.00	<u>43.37</u>	88.25	47.60	20.48	83.93
LoRA	47.24 \pm 0.67	<u>25.54</u> \pm 1.01	68.32 \pm 0.72	72.84 \pm 0.64	42.65 \pm 1.08	88.22 \pm 0.58	47.92 \pm 1.49	<u>22.17</u> \pm 0.66	<u>84.22</u> \pm 1.27
Soft CoT	<u>49.32</u> \pm 0.73	24.34 \pm 1.98	69.80 \pm 0.81	<u>73.20</u> \pm 1.44	41.93 \pm 0.54	<u>89.05</u> \pm 1.00	<u>48.48</u> \pm 0.97	19.76 \pm 1.37	84.20 \pm 1.25
ReFT	40.20 \pm 0.92	20.24 \pm 1.32	63.76 \pm 1.84	68.88 \pm 1.03	40.72 \pm 0.54	85.08 \pm 1.19	47.08 \pm 1.08	20.96 \pm 1.37	82.65 \pm 1.53
CAA	43.80	21.69	64.75	71.60	40.96	86.28	48.20	21.69	83.17
Pause Token	43.88 \pm 1.58	21.20 \pm 1.37	63.56 \pm 0.32	69.92 \pm 1.25	40.96 \pm 0.85	86.08 \pm 0.29	47.36 \pm 1.68	20.48 \pm 1.90	82.68 \pm 0.41
Coconut	46.36 \pm 0.64	23.37 \pm 1.37	65.35 \pm 1.21	71.72 \pm 0.76	42.17 \pm 0.85	86.07 \pm 0.31	48.08 \pm 1.32	18.55 \pm 0.66	83.32 \pm 1.24
PILOT (Ours)	52.08 \pm 0.59	26.27 \pm 1.01	<u>68.93</u> \pm 1.07	75.24 \pm 1.13	45.06 \pm 1.37	89.17 \pm 0.87	51.64 \pm 0.77	24.10 \pm 1.90	85.35 \pm 0.81

PANEL B: CODE GENERATION						
Method	Qwen2.5-1.5B		Qwen2.5-7B		Llama-3.1-8B	
	HEval	MBPP	HEval	MBPP	HEval	MBPP
Zero-shot CoT	46.34	42.40	71.34	<u>57.60</u>	53.05	53.20
LoRA	50.12 \pm 0.80	<u>44.24</u> \pm 0.93	<u>75.73</u> \pm 0.67	56.36 \pm 0.65	<u>57.07</u> \pm 0.33	<u>54.48</u> \pm 0.67
Soft CoT	<u>50.49</u> \pm 0.67	41.16 \pm 1.63	71.83 \pm 1.64	56.04 \pm 0.74	53.17 \pm 0.67	53.28 \pm 0.70
ReFT	42.80 \pm 1.17	39.80 \pm 0.45	68.66 \pm 1.47	54.40 \pm 0.75	51.34 \pm 1.09	52.04 \pm 0.70
CAA	43.90	40.60	69.51	55.20	51.83	52.60
Pause Token	44.88 \pm 0.70	41.88 \pm 1.24	68.41 \pm 0.80	55.64 \pm 1.32	52.07 \pm 2.05	52.12 \pm 0.87
Coconut	47.68 \pm 1.52	39.84 \pm 1.60	69.39 \pm 0.51	54.80 \pm 1.14	52.56 \pm 0.90	51.32 \pm 0.97
PILOT (Ours)	56.34 \pm 0.33	46.36 \pm 0.48	77.44 \pm 1.43	61.60 \pm 0.66	58.41 \pm 2.04	56.80 \pm 0.72

Table 2: **Data Filtering Statistics.** Retention rate reflects the percentage of samples passing the Construct-and-Verify pipeline. Larger models typically show distinct retention patterns based on task difficulty.

Base Model	Source	Original	Filtered \mathcal{D}_{train}	Retention
DOMAIN: MATHEMATICS				
Qwen2.5-1.5B			1,103	14.7%
Qwen2.5-7B	MATH	7,500	543	7.2%
Llama-3.1-8B			885	11.8%
DOMAIN: CODE GENERATION				
Qwen2.5-1.5B			204	54.5%
Qwen2.5-7B	MBPP	374	267	71.4%
Llama-3.1-8B			244	65.2%

greedy decoding (temperature=0). To account for training variance, we train 5 independent adapters with different random seeds and report the mean and standard deviation across 5 independent runs.

5.2 Main Results

Table 1 presents the comprehensive performance. We compare PILOT against discrete prompting, static tuning, and latent intervention baselines.

Dominance in Complex Reasoning. On **MATH500**, PILOT consistently outperforms all baselines. Notably, on Qwen2.5-1.5B, PILOT

achieves a remarkable gain, significantly surpassing Soft CoT and LoRA. This confirms dynamic latent anchoring effectively activates dormant reasoning capacity. Even on stronger 7B/8B models, PILOT maintains a clear edge, suggesting that static tuning struggles to generalize from limited training data (less than 1k samples).

Robustness. As seen in Table 1, several latent-intervention baselines (e.g., **ReFT** and **Pause Token**) underperform the base model on GSM8K, suggesting that directly modifying hidden states or introducing unguided extra tokens can be brittle under distribution shift. **CAA** improves over ReFT and slightly surpasses the Zero-shot baseline on MATH, but does not consistently transfer to GSM8K. **Coconut** also shows limited gains across settings. In contrast, PILOT maintains robustness on saturated benchmarks and yields consistent improvements, consistent with our Energy-Aligned Injection being non-invasive.

Code Generalization. PILOT generalizes well on **HumanEval** (e.g., surpassing LoRA on Qwen-7B). On MBPP, static tuning can sometimes underperform the base model, suggesting potential negative transfer. PILOT’s input-dependent modulation mitigates this effect while improving both

coding benchmarks.

5.3 Additional Validation of Planning Internalization

We include three targeted validations to test whether PILOT’s gains can be reproduced by *explicit text-level planning* and whether latent injection harms representation geometry.

Setup of Textual Planning Baselines. Reviewers specifically asked whether PILOT’s gains could be recovered by making the planning signal explicit in text. We therefore construct two additional baselines on the same 7B backbone. **External Planning Prompting** simulates “Stage 1 at inference time”: for each query, we first use **DeepSeek-V3.1** to generate a concise strategy hint g_{exp} *without access to the ground-truth answer*, and prepend it to the zero-shot prompt. **LoRA SFT (Text Distillation)** instead fine-tunes the model to explicitly emit heuristic guidance before solving, using the *same* Construct-and-Verify supervision as PILOT ($|D_{train}|=543$ for Math and 267 for Code). This isolates whether PILOT’s benefit comes merely from exposing the heuristic in text, rather than from internalizing it as latent control.

Latent Planning vs. Textual Planning. Table 3 shows a consistent gap between latent and text-level planning. **External Planning Prompting** improves over Zero-shot CoT on some in-domain settings (e.g., MATH500: 72.60 vs. 71.00), but still remains clearly below PILOT (75.24), and degrades on MBPP (54.00 < 57.60). **LoRA SFT (Text Distillation)** is even less stable: while it learns to verbalize heuristic guidance, it underperforms PILOT on every benchmark and falls below the zero-shot baseline on AIMO and MBPP. These results indicate that simply *showing* or *generating* a plan in natural language is insufficient.

Why Textual Planning Falls Short. The failure mode is consistent with the rebuttal hypothesis of a **text-level information bottleneck**. Once planning is serialized into text, the compact model must first parse the heuristic and then follow it within the same autoregressive stream used for execution. This introduces additional exposure bias and can conflict with the model’s native token distribution, especially on out-of-distribution tasks. By contrast, PILOT internalizes the same strategic information directly in latent space, avoiding this detour through surface-form generation while preserving

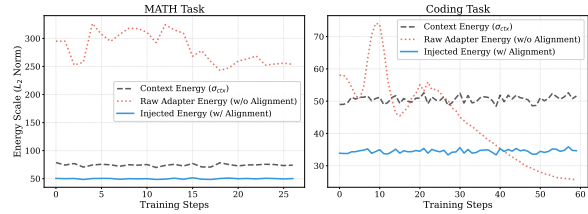


Figure 3: **Energy Alignment Dynamics (7B)**. Tracking injection vector L_2 norm. **Left (Math)**: Raw energy naturally aligns with context. **Right (Code)**: PILOT’s alignment constrains wild fluctuations, preventing “embedding shock” and ensuring stability.

the low-overhead single-path decoding property.

Isotropy and Oversmoothing. To directly test whether latent injection collapses the representation manifold, we extract the final-layer hidden state of the last generated token from $N=100$ correctly solved examples on MATH500 and HumanEval, and compute the mean pairwise cosine similarity. Table 4 shows that the average similarity *decreases* after applying PILOT in both domains (Math: 0.838 \rightarrow 0.692; Code: 0.944 \rightarrow 0.867). In other words, the induced representations become *more* dispersed rather than more collapsed. This directly argues against the oversmoothing hypothesis and supports our design goal that Energy-Aligned Injection should **steer** the trajectory without compressing the model’s native geometry.

5.4 Ablation Studies

To validate component contributions, we conduct ablation studies across model scales. Table 5 summarizes the results.

Architectural Ablation. Table 5 reveals scale-dependent dynamics. **Hyper-Network:** Removing input-dependent anchoring (Row 1) consistently degrades performance across both tasks and scales, confirming that static adapters cannot capture the complexity of reasoning manifolds. **Proto-Thought:** While critical for 1.5B, its impact on Code diminishes at 7B (77.44 \rightarrow 76.71), suggesting larger models may implicitly learn structural priors. However, it remains vital for Math (75.24 \rightarrow 74.52), where logical planning is less inherent in the pre-training objective.

Energy Alignment Analysis. A striking divergence appears in Row 3. Removing Energy-Alignment causes a severe drop in Code for 7B (77.44 \rightarrow 73.17), whereas Math remains unaffected. To explain this, we visualize the norm evo-

Table 3: **Latent Planning vs. Explicit Textual Planning.** Pass@1 (%) on Qwen2.5-7B-Instruct. **External Planning Prompting** prepends strategy hints generated by a stronger planner at inference time, while **LoRA SFT (Text Distillation)** trains the model to explicitly output heuristic text before reasoning using the same Construct-and-Verify supervision as PILOT.

Method	MATH500	AIMO	GSM8K	HumanEval	MBPP
Zero-shot CoT	71.00	43.37	88.25	71.34	57.60
External Planning Prompting	72.60	43.37	89.31	73.29	54.00
LoRA SFT (Text Distillation)	72.20	42.17	87.95	70.81	55.00
PILOT (Latent)	75.24	45.06	<u>89.17</u>	77.44	61.60

Table 4: **Representation Isotropy Diagnostics.** Average pairwise cosine similarity of final-layer hidden states over $N=100$ correctly solved examples. Lower values indicate more dispersed representations.

Domain	Base	PILOT	Δ
Math	0.838	0.692	-0.146
Code	0.944	0.867	-0.077

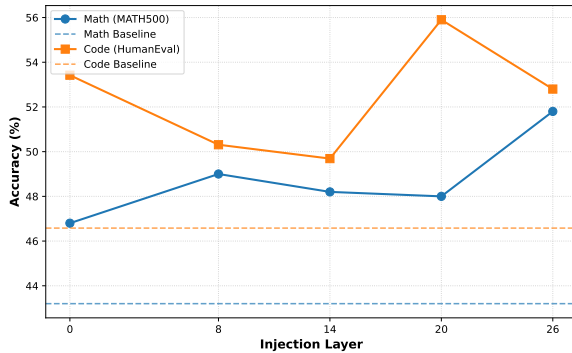


Figure 4: **Injection Depth Sensitivity (Qwen-1.5B).** Optimal pivots shift by task: **Math** peaks at the deepest layer (26), while **Code** peaks earlier (20).

lution in Figure 3. In **Code Generation**, the **Raw Adapter Energy** (without alignment) exhibits high variance and often exceeds the **Context Energy** (σ_{ctx}), risking a "structural shock" that disrupts syntax. PILOT's alignment mechanism forces the **Injected Energy** to track σ_{ctx} , stabilizing the intervention. In **Mathematics**, the raw energy naturally converges near the context norm, rendering explicit alignment less critical at this scale. This confirms that coding tasks require stricter "energy preservation" to maintain valid output distributions.

Data Efficiency. Separately, we validated our filtering protocol on 1.5B. Training on the full unfiltered dataset degraded performance to 50.85% (Math) and 52.20% (Code), confirming the value of signal concentration.

Injection Depth. Figure 4 shows that PILOT improves performance across all tested layers. The optimal pivot shifts by task nature: **Math** peaks at the deepest layer, suggesting abstract reasoning is best guided at the final stage of semantic aggregation. **Code** peaks earlier, indicating a need to guide structural logic before final syntax rigidifies.

5.5 Analysis

Efficiency & Overhead. We benchmark inference latency on a single NVIDIA H20 GPU under a unified evaluation protocol (same prompts, decoding settings, and generation budget) across all methods. Table 6 reports **prefill (TTFT)** and **end-to-end** latency averaged over 1k samples. **PILOT** introduces a small prefill overhead of +3.10 ms (21.66 ms vs. 18.56 ms), attributable to a single forward pass of our lightweight Hyper-Network. During decoding, PILOT injects a static anchor state and therefore does not add per-step recurrent computation, resulting in a negligible total latency increase of 0.2% (10,230 ms vs. 10,209 ms). **Soft CoT** incurs a higher end-to-end latency overhead in our measurement (+6.2%). We attribute this mainly to its reliance on an additional optimization pipeline for continuous prompts; nevertheless, we follow the official implementation and keep the evaluation protocol consistent across methods.

Anchoring Dynamics. To understand how PILOT alters the inference trajectory, we analyze the layer-wise cosine similarity between the base and anchored hidden states (Figure 5). We observe distinct "Phase Shift" behaviors:

- **Math (Terminal Correction):** For MATH500 (Figure 5a), anchoring occurs at the deepest semantic pivot (Layer 26). The similarity drops moderately (~ 0.7) and remains divergent. This indicates a "**last-mile correction**", where PILOT refines the final semantic representation right

Table 5: **Component Ablation across Scales.** Pass@1 accuracy (Mean \pm Std). **Math** relies heavily on the **Hyper-Network** for anchoring, while **Code** depends on **Energy-Alignment** to prevent structural collapse. The **Proto-Thought** prior becomes less critical for Code as model scale increases.

Configuration	Qwen2.5-1.5B		Qwen2.5-7B	
	MATH	HEval	MATH	HEval
PILOT (Full)	52.08 \pm 0.59	56.34 \pm 0.33	75.24 \pm 1.13	77.44 \pm 1.43
<i>Architectural Variants</i>				
(1) w/o Hyper-Net (Static)	47.72 \pm 0.99	50.37 \pm 1.70	72.84 \pm 1.44	72.68 \pm 0.67
(2) w/o Proto-Thought	48.84 \pm 1.13	54.39 \pm 1.32	74.52 \pm 1.40	76.71 \pm 1.52
(3) w/o Energy-Alignment	49.16 \pm 0.61	51.22 \pm 0.43	75.04 \pm 1.01	73.17 \pm 0.61

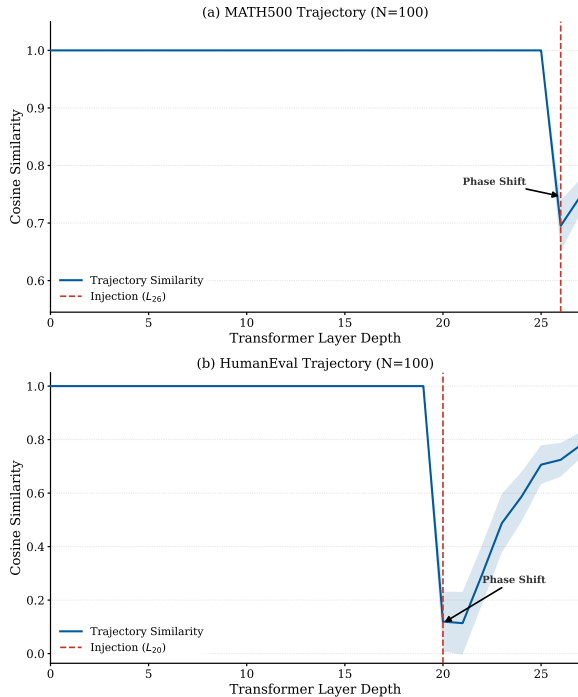


Figure 5: **Mechanistic Visualization of Latent Anchoring.** Cosine similarity between base and anchored states. **(a) Math:** Layer 26 anchoring indicates "last-mile" correction. **(b) Code:** Layer 20 injection triggers "shock" followed by recovery, implying deep restructuring. Shaded regions: std dev ($N = 100$).

before decoding, ensuring precise logical closure without disrupting the previously accumulated context.

- **Code (Deep Restructuring):** For HumanEval (Figure 5b), the intervention at Layer 20 triggers a massive "injection shock" (similarity drops to ~ 0.1). This implies that for coding, PILOT fundamentally overwrites the model's internal plan with a new structural blueprint. Crucially, we observe an "Assimilation Phase" (Layers 21-27), where the similarity recovers to ~ 0.8 . This reveals the model's mechanism: it absorbs the drastic an-

Table 6: **Inference Latency Analysis.** Benchmarked on Qwen2.5-7B (Avg. over 1k samples). **PILOT** incurs negligible decoding overhead compared to **Soft CoT**.

Method	Prefill (TTFT)		Total Latency	
	Time (ms)	Overhead	Time (ms)	Overhead
Base Model	18.56	–	10,209.42	–
PILOT (Ours)	21.66	+3.10 ms	10,230.52	+0.2%
Soft CoT	44.31	+25.75 ms	10,842.40	+6.2%

choring signal (the drop) and integrates it with its pre-trained linguistic knowledge to generate valid code. The sustained gap (1.0 \rightarrow 0.8) at the final layer confirms that the output distribution remains successfully shifted.

Parameter Efficiency. PILOT is designed for parameter efficiency. For instance, on Qwen2.5-7B (7.6B parameters), it introduces only 38.6M trainable parameters ($\sim 0.5\%$), and on Qwen2.5-1.5B (1.5B parameters), it requires just 7.1M trainable parameters ($\sim 0.46\%$). This lightweight nature allows for rapid adaptation and minimal storage overhead compared to full fine-tuning.

6 Conclusion

In this paper, we study how to stabilize reasoning trajectories in LLMs and propose **PILOT** (Planning via Internalized Latent Optimization Trajectories; **PILOT**). PILOT dynamically primes the model's internal representations with instance-specific anchor vectors, providing a non-invasive way to incorporate heuristic guidance without updating backbone weights. Experiments on challenging mathematics and code-generation benchmarks show consistent gains while introducing minimal decoding overhead in our evaluation setting. Our mechanistic analyses further indicate that the injected anchors can induce a coherent shift in intermediate representations, helping maintain logical consistency over long-horizon generation.

7 Limitations

While **PILOT** offers a robust alternative to search-based reasoning with zero recurrent decoding latency, we acknowledge specific limitations regarding deployment complexity and generalization:

Data Construction Overhead. A core component of our framework is the *Construct-and-Verify* pipeline, which distills expert guidance into high-fidelity **latent anchors**. While this process is crucial for performance, it introduces additional preprocessing complexity compared to standard Supervised Fine-Tuning on raw datasets. The requirement to synthesize and filter high-quality trajectories creates a trade-off where we accept higher offline data preparation costs to achieve maximum efficiency during online inference.

Domain-Specific Hyperparameters. Our analysis highlights that the optimal **anchoring depth** (the pivot layer) shifts depending on the task nature—deeper for abstract mathematics and shallower for structural code generation. Currently, this insertion layer is treated as a static hyperparameter per domain. Although effective, this requires empirical tuning when adapting the framework to new domains, and a fully dynamic, instance-wise layer selection mechanism remains a direction for future work.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. [Evaluating large language models trained on code](#).

Younwoo Choi, Muhammad Adil Asif, Ziwen Han, John Willes, and Rahul G Krishnan. 2025. Teaching llms how to learn with contextual fine-tuning. *arXiv preprint arXiv:2503.09032*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, and 1 others. 2023. Faith and fate: Limits of transformers on compositionality. *Advances in Neural Information Processing Systems*, 36:70293–70332.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. 2023. Think before you speak: Training language models with pause tokens. *arXiv preprint arXiv:2310.02226*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

David Ha, Andrew Dai, and Quoc V Le. 2016. Hypernetworks. *arXiv preprint arXiv:1609.09106*.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Ceyhun Efe Kayan and Li Zhang. 2025. Prototype-based dynamic steering for large language models. *arXiv preprint arXiv:2510.05498*.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, and 1 others. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

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

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2025. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*.

Nina Panickssery, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Matt Turner. 2024. [Steering llama 2 via contrastive activation addition](#). *Preprint*, arXiv:2312.06681.

- DiJia Su, Hanlin Zhu, Yingchen Xu, Jiantao Jiao, Yuandong Tian, and Qingqing Zheng. 2025. Token assorted: Mixing latent and text tokens for improved language model reasoning. *arXiv preprint arXiv:2502.03275*.
- Qi Sun, Edoardo Cetin, and Yujin Tang. 2025. Transformer-squared: Self-adaptive llms. *arXiv preprint arXiv:2501.06252*.
- Xinyu Tang, Xiaolei Wang, Zhihao Lv, Yingqian Min, Wayne Xin Zhao, Binbin Hu, Ziqi Liu, and Zhiqiang Zhang. 2025. Unlocking general long chain-of-thought reasoning capabilities of large language models via representation engineering. *arXiv preprint arXiv:2503.11314*.
- AI-MO Team. 2024. Aimo validation amc dataset. <https://huggingface.co/datasets/AI-MO/aimo-validation-amc>. Accessed: 2024-05-22.
- Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J Vazquez, Ulisse Mini, and Monte MacDiarmid. 2023. Steering language models with activation engineering. *arXiv preprint arXiv:2308.10248*.
- Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. 2023. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965.
- Constantin Venhoff, Iván Arcuschin, Philip Torr, Arthur Conmy, and Neel Nanda. 2025. Understanding reasoning in thinking language models via steering vectors. *arXiv preprint arXiv:2506.18167*.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D Manning, and Christopher Potts. 2024a. Reft: Representation finetuning for language models. *Advances in Neural Information Processing Systems*, 37:63908–63962.
- Zhenyu Wu, Meng Jiang, and Chao Shen. 2024b. Get an a in math: Progressive rectification prompting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19288–19296.
- Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. 2025. Softcot: Soft chain-of-thought for efficient reasoning with llms. *arXiv preprint arXiv:2502.12134*.
- Tianci Xue, Ziqi Wang, Zhenhailong Wang, Chi Han, Pengfei Yu, and Heng Ji. 2023. Rcot: Detecting and rectifying factual inconsistency in reasoning by reversing chain-of-thought. *arXiv preprint arXiv:2305.11499*.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.
- Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. 2024. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*.
- Dongxu Zhang, Ning Yang, Jihua Zhu, Jinnan Yang, Miao Xin, and Baoliang Tian. 2025. Ascot: An adaptive self-correction chain-of-thought method for late-stage fragility in llms. *arXiv preprint arXiv:2508.05282*.
- Yu Zhao, Alessio Devoto, Giwon Hong, Xiaotang Du, Aryo Pradipta Gema, Hongru Wang, Xuanli He, Kam-Fai Wong, and Pasquale Minervini. 2025. Steering knowledge selection behaviours in llms via sae-based representation engineering. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5117–5136.
- Yufa Zhou, Yixiao Wang, Xunjian Yin, Shuyan Zhou, and Anru R Zhang. 2025. The geometry of reasoning: Flowing logics in representation space. *arXiv preprint arXiv:2510.09782*.
- Rui-Jie Zhu, Tianhao Peng, Tianhao Cheng, Xingwei Qu, Jinfeng Huang, Dawei Zhu, Hao Wang, Kaiwen Xue, Xuanliang Zhang, Yong Shan, and 1 others. 2025. A survey on latent reasoning. *arXiv preprint arXiv:2507.06203*.
- Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, and 1 others. 2023. Representation engineering: A top-down approach to ai transparency. *arXiv preprint arXiv:2310.01405*.

A Instruction Templates

This section details the instruction templates. We categorize them into **Task-specific Solvers** (for baseline and verified responses) and **Heuristic Strategy Generators** (for distilling the expert manifold). To ensure the predicted **anchor vector** \hat{z} represents abstract strategic intent, all heuristic templates enforce strict constraints against revealing concrete values or step-by-step solutions.

A.1 Templates for Mathematical Reasoning (MATH)

Template A.1: Math Zero-shot Solver

You are an AI assistant, you are required to solve mathematical question step by step. Please provide your final answer on a separate line in `\boxed{\}` format.

Template A.2: Heuristic Strategy Generator (g_{exp})

You are an expert mathematics tutor. Provide a **strategic insight** to guide problem-solving. **CRITICAL RULES:** (1) NO numbers, specific values, or final answers. (2) NO step-by-step derivations or explicit formulas. (3) ONE core strategy in 1-2 sentences. **Focus on:** What mathematical concept/method to apply or what key transformation unlocks the problem.

A.2 Templates for Programming Tasks (MBPP)

Template A.5: Programming Zero-shot Solver

You are a Python programmer. Output ONLY the function implementation. No explanations, no markdown, no comments.

Template A.6: Programming Heuristic Generator (g_{exp})

You are an expert programming tutor. Provide a **strategic insight** to guide problem-solving. **CRITICAL RULES:** (1) NO code, syntax, or implementation details. (2) NO step-by-step logic. (3) ONE core strategy in 1-2 sentences. **Focus on:** What algorithm or data structure to use.

B Data Generation Pipeline

The **Construct-and-Verify** pipeline (as defined in Sec 4.1) identifies high-signal triplets where the **expert heuristic guidance** g_{exp} serves as a necessary condition for successful reasoning.

B.1 Pipeline Logic and Pseudocode

Algorithm 1 describes the systematic construction of \mathcal{D}_{train} . The **Blind Test** prevents **anchor vector** contamination by ensuring g_{exp} does not leak the final answer, forcing the adapter to learn strategic anchoring rather than trivial mapping.

Algorithm 1 Construct-and-Verify Pipeline

Require: Raw dataset \mathcal{D}_{raw} , Base model \mathcal{M}_ϕ , Expert model \mathcal{M}_{exp}

Ensure: Filtered dataset \mathcal{D}_{train}

```
1:  $\mathcal{D}_{train} \leftarrow \emptyset$ 
2: for each question  $x \in \mathcal{D}_{raw}$  do
3:    $y_{zs} \leftarrow \mathcal{M}_\phi(\text{Solver}(x))$  {Zero-shot baseline trial}
4:   if is_correct( $y_{zs}$ ) and domain is Math then
5:     continue {Boundary Filter: Capture the reasoning frontier}
6:   end if
7:    $g_{exp} \leftarrow \mathcal{M}_{exp}(\text{Heuristic\_Gen}(x))$  {Generate  $g_{exp}$  strategic anchor}
8:    $y^* \leftarrow \mathcal{M}_\phi(\text{Solver}(x, g_{exp}))$  {Verify guidance effectiveness}
9:   if not is_correct( $y^*$ ) then
10:    continue {Discard if guidance is insufficient to activate the correct manifold}
11:  end if
12:  {Blind Test Stage: Preventing Answer Leakage}
13:   $y_{blind} \leftarrow \mathcal{M}_\phi(\text{Solver}(g_{exp}))$  {Check for direct leakage in  $g_{exp}$ }
14:  if is_correct( $y_{blind}$ ) then
15:    continue
16:  end if
17:   $\mathcal{D}_{train} \leftarrow \mathcal{D}_{train} \cup \{(x, g_{exp}, y^*)\}$ 
18: end for
19: return  $\mathcal{D}_{train}$ 
```

B.2 Differentiated Filtering by Domain

- **Mathematics (Boundary Filtering):** We prioritize instances where the model fails zero-shot. This ensures the **Anchor Adapter** learns a corrective signal to anchor the hidden states from failure manifolds toward the reasoning manifold defined by g_{exp} .

- **Programming (Refinement Filtering):** We retain correct cases to anchor the model toward more *optimized* algorithmic structures, leveraging PILOT’s ability to reinforce efficient coding trajectories.

C Teacher Capability Sensitivity Analysis

A critical validation of PILOT is whether the performance gains stem from the **Energy-Aligned Injection** mechanism itself or merely from distillation of teacher knowledge. We decouple these via a teacher-sensitivity analysis.

- **PILOT (Self):** The teacher is the base model itself. To derive high-quality **homogeneous target states \mathbf{z}^*** from the 1.5B model, we utilize a higher sampling budget (32 retries) during the **Construct** phase to find a successful path.
- **PILOT (Strong):** The teacher is **DeepSeek-V3**, providing expert cross-model heuristics to guide the target extraction.

As shown in Table 7, **PILOT (Self)** achieves substantial gains. Since the knowledge source is internal to the model, these results empirically prove that PILOT’s **Anchor Adapter** effectively stabilizes the model’s intrinsic reasoning potential by mitigating autoregressive drift.

Table 7: **Teacher Capability Sensitivity Analysis.** Comparison of performance using the model itself (*Self*) vs. **DeepSeek-V3.1** (*Strong*) as the expert for **target construction**. Pass@1 (%). *Base* refers to zero-shot CoT.

Base Model	Benchmark	Base	PILOT (Self)	PILOT (Strong)	Δ_{S-S}
Qwen2.5-1.5B	MATH500	43.20	51.80	52.00	+0.20
	Human-Eval	46.34	54.88	56.10	+1.22
Qwen2.5-7B	MATH500	71.00	73.60	75.20	+1.60
	Human-Eval	71.34	78.66	77.44	-1.22

D Impact of Training Data Scale and Distribution

In our main experiments (Sec. 5), we utilized a filtered subset \mathcal{D}_{train} constructed via our *Construct-and-Verify* pipeline. A natural question arises: *Does reducing the data scale limit the model’s potential, and would training on the full official datasets yield better results?*

To address this, we conducted a comprehensive comparative study across all baseline methods, including **LoRA**, **Soft CoT**, **ReFT**, and **Co-**

conut. We trained these baselines using two different datasets: the full official training sets (\mathcal{D}_{full}) of **MATH** (7,500 samples) and **MBPP** (374 samples), and our filtered subset \mathcal{D}_{train} . We evaluated performance on **MATH500** and **HumanEval** to assess generalization. We also include **Zero-shot CoT** performance as a reference for the base model’s capability.

Table 8 presents the comparative results. We observe distinct trends across domains. On **MATH500**, training on the full dataset (\mathcal{D}_{full}) often leads to performance degradation compared to the zero-shot baseline for methods like LoRA, ReFT, and Coconut. This suggests that the distribution mismatch between the ground-truth solutions in \mathcal{D}_{full} and the model’s internal reasoning manifold causes catastrophic forgetting. However, **Soft CoT** is a notable exception, maintaining robustness even with full data. In contrast, our filtered subset \mathcal{D}_{train} consistently outperforms \mathcal{D}_{full} and generally improves upon the zero-shot baseline, highlighting the importance of on-manifold data. On **HumanEval**, \mathcal{D}_{train} consistently outperforms or matches \mathcal{D}_{full} , further validating our approach.

We attribute this phenomenon to two key factors:

- **Distribution Mismatch and Catastrophic Forgetting:** The official datasets contain ground-truth solutions that may not align with the base model’s internal reasoning manifold. Forcing the model to mimic these "alien" distributions can disrupt its pre-trained knowledge, leading to catastrophic forgetting of its intrinsic capabilities. This is evident in the performance drop of LoRA, ReFT, and Coconut on MATH500 when trained on \mathcal{D}_{full} . In contrast, our \mathcal{D}_{train} consists of *self-generated* valid reasoning paths, ensuring that the training signal is strictly on-manifold.
- **Robustness of Soft CoT:** Interestingly, **Soft CoT** appears less susceptible to this negative transfer, likely because it learns a soft prompt to guide reasoning rather than modifying the model weights directly (or as extensively). However, it still benefits from the cleaner, aligned signal provided by our filtered data.
- **Data Efficiency and Coding Precision:** In the coding domain (HumanEval), our filtered dataset consistently outperforms or matches the full dataset. This suggests that for code

Table 8: **Impact of Training Data Scale on Baselines.** Comparison of performance when trained on the full official dataset (\mathcal{D}_{full}) versus our filtered, model-aligned subset (\mathcal{D}_{train}). **Zero-shot CoT** represents the base model performance without fine-tuning. On MATH500, training on \mathcal{D}_{full} often hurts performance (falling below Zero-shot CoT) due to distribution shift, whereas \mathcal{D}_{train} yields consistent gains. Soft CoT is an exception, benefiting from \mathcal{D}_{full} but still performing best with \mathcal{D}_{train} . All results are reported as Mean \pm Std over 5 runs (except for CAA, which is deterministic).

Base Model	Method	MATH500			HumanEval		
		Full (\mathcal{D}_{full})	Filtered (\mathcal{D}_{train})	Δ	Full (\mathcal{D}_{full})	Filtered (\mathcal{D}_{train})	Δ
Qwen2.5-1.5B	<i>Zero-shot CoT</i>		43.20	–		46.34	–
	LoRA	42.88 \pm 1.45	47.24 \pm 0.67	+4.36	48.90 \pm 1.00	50.12 \pm 0.80	+1.22
	Soft CoT	45.20 \pm 0.87	49.32 \pm 0.73	+4.12	50.24 \pm 1.02	50.49 \pm 0.67	+0.24
	ReFT	40.12 \pm 0.79	40.20 \pm 0.92	+0.08	42.68 \pm 0.96	42.80 \pm 1.17	+0.12
	CAA	42.20	43.80	+1.60	42.68	43.90	+1.22
	Pause Token	42.32 \pm 0.48	43.88 \pm 1.58	+1.56	43.78 \pm 1.00	44.88 \pm 0.70	+1.10
	Coconut	41.08 \pm 1.68	46.36 \pm 0.64	+5.28	47.07 \pm 1.09	47.68 \pm 1.52	+0.61
Qwen2.5-7B	<i>Zero-shot CoT</i>		71.00	–		71.34	–
	LoRA	70.56 \pm 1.04	72.84 \pm 0.64	+2.28	74.39 \pm 0.96	75.73 \pm 0.67	+1.34
	Soft CoT	71.60 \pm 1.20	73.20 \pm 1.44	+1.60	71.10 \pm 1.91	71.83 \pm 1.64	+0.73
	ReFT	67.24 \pm 0.33	68.88 \pm 1.03	+1.64	68.05 \pm 0.55	68.66 \pm 1.47	+0.61
	CAA	69.80	71.60	+1.80	68.29	69.51	+1.22
	Pause Token	70.08 \pm 1.44	69.92 \pm 1.25	-0.16	69.15 \pm 1.65	68.41 \pm 0.80	-0.73
	Coconut	69.24 \pm 0.61	71.72 \pm 0.76	+2.48	67.80 \pm 0.90	69.39 \pm 0.51	+1.59
Llama-3.1-8B	<i>Zero-shot CoT</i>		47.60	–		53.05	–
	LoRA	47.36 \pm 1.55	47.92 \pm 1.49	+0.56	56.22 \pm 0.90	57.07 \pm 0.33	+0.85
	Soft CoT	47.36 \pm 0.86	48.48 \pm 0.97	+1.12	52.20 \pm 1.53	53.17 \pm 0.67	+0.98
	ReFT	45.48 \pm 0.61	47.08 \pm 1.08	+1.60	50.12 \pm 1.46	51.34 \pm 1.09	+1.22
	CAA	46.40	48.20	+1.80	50.61	51.83	+1.22
	Pause Token	46.44 \pm 0.71	47.36 \pm 1.68	+0.92	50.98 \pm 0.33	52.07 \pm 2.05	+1.10
	Coconut	45.40 \pm 1.21	48.08 \pm 1.32	+2.68	52.07 \pm 0.70	52.56 \pm 0.90	+0.49

generation, eliminating noise and ensuring correct, optimized solutions (as done in our pipeline) is more critical than raw volume.

- **Experimental Robustness:** We report the Mean \pm Std over 5 independent runs for all experiments in Table 8. The consistent trends across multiple models and methods, backed by low variance, strongly support our conclusions regarding the trade-offs between data scale and quality.

E Cross-Domain Generalization Analysis

To further evaluate the robustness of the learned reasoning patterns, we conducted a cross-domain generalization experiment. Specifically, we trained PILOT on the **MATH** dataset and evaluated it on the **HumanEval** coding task, and conversely, trained on the **Code** dataset and evaluated on **MATH500**. This setup tests whether the anchoring capabilities learned in one domain can transfer to another, indicating the acquisition of abstract, domain-agnostic reasoning strategies.

Table 9 presents the results. We observe that models trained on mathematical reasoning demonstrate strong transfer performance to coding tasks, suggesting that the logical structuring learned from math problems is highly relevant to code generation. Similarly, models trained on code show competitive performance on math tasks, although the transfer is slightly less pronounced. This asymmetry might be due to the more rigid syntax requirements of code compared to the flexible reasoning paths in mathematics. Nevertheless, the positive transfer in both directions confirms that PILOT captures underlying reasoning manifolds that are shared across domains.

F Implementation Details of PILOT

F.1 HyperNetwork Architecture

The HyperNetwork is implemented as a Multi-Layer Perceptron (MLP) that maps the context vector c to the affine transformation parameters γ and β . Specifically, it consists of two linear layers with a hidden dimension equal to the model’s hidden

Table 9: **Cross-Domain Generalization.** Transferring reasoning patterns between Math and Code. \mathcal{D}_m and \mathcal{D}_c denote models trained on Math and Code data respectively. Zero-shot (ZS) represents the base model performance.

Model	Task	Source	Accuracy (%)
Qwen2.5-1.5B	HEval	Zero-shot CoT	46.34
		Math (\mathcal{D}_m)	53.41 \pm 0.70
	MATH	Zero-shot CoT	43.20
		Code (\mathcal{D}_c)	46.96 \pm 0.38
Qwen2.5-7B	HEval	Zero-shot CoT	71.34
		Math (\mathcal{D}_m)	72.56 \pm 0.86
	MATH	Zero-shot CoT	71.00
		Code (\mathcal{D}_c)	73.84 \pm 0.41

size h (not $h/2$). The architecture is defined as:

$$[\gamma, \beta] = \mathbf{W}_2 \cdot \text{Dropout}(\text{GELU}(\text{LayerNorm}(\mathbf{W}_1 \cdot \mathbf{c}))) \quad (8)$$

where $\mathbf{W}_1 \in \mathbb{R}^{h \times h}$ and $\mathbf{W}_2 \in \mathbb{R}^{2h \times h}$. The output is split into $\gamma \in \mathbb{R}^h$ and $\beta \in \mathbb{R}^h$. We apply Layer Normalization before the activation function to stabilize training.

F.2 Normalization and Energy Scaling

To ensure the injected anchor vector $\hat{\mathbf{z}}$ remains within the valid manifold of the base model, we apply a rigorous normalization and scaling process. The process follows this specific order:

- Layer Normalization:** The raw output from the HyperNetwork is first normalized using LayerNorm.
- L2 Normalization:** The vector is then projected onto the unit hypersphere via L2 normalization: $\mathbf{v} = \frac{\mathbf{v}'}{\|\mathbf{v}'\|_2}$.
- Energy Scaling:** We re-scale the unit vector by the average energy of the context tokens (σ_{ctx}) to match the local activation magnitude.
- Gate Scaling:** Finally, a learnable scalar gate α (initialized to 0) modulates the injection strength via a Softplus activation.

F.3 Layer Selection

The choice of the insertion layer l^\dagger is critical for effective anchoring. We empirically selected the insertion layers for different models and tasks as shown in Table 10. Generally, we target the deeper layers where abstract reasoning features are formed.

Concretely, both Qwen2.5-1.5B and Qwen2.5-7B contain 28 transformer layers, and we inject at $l^\dagger = 26$ for Math to steer representations near the end of the computation pipeline, where the model aggregates long-range evidence and forms more abstract planning features. In contrast, for Programming we inject earlier (e.g., $l^\dagger = 20$) to influence algorithmic structure and constraint satisfaction before the model commits to surface-form code tokens. For Llama-3.1-8B (32 layers), we follow the same principle by choosing a late layer for Math ($l^\dagger = 31$) and a moderately earlier layer for Programming ($l^\dagger = 25$), balancing sufficient depth for the anchor to propagate while avoiding overly late interventions that primarily affect decoding style rather than problem-solving trajectory.

Table 10: Insertion layer (l^\dagger) configuration for different models and tasks.

Model	Math Task	Programming Task
Qwen2.5-1.5B	Layer 26	Layer 20
Qwen2.5-7B	Layer 26	Layer 20
Llama-3.1-8B	Layer 31	Layer 25

G Implementation Details of Baselines

We provide implementation specifications for the baseline methods used in our experiments. Training epochs are aligned with the main experiments (3 epochs for Math tasks, 10 epochs for Coding tasks) to ensure fair comparison.

G.1 LoRA Baseline

We utilize the standard Low-Rank Adaptation (LoRA) (Hu et al., 2022) as the primary baseline. We use the peft library to inject adapters into all linear projection layers of the attention mechanism. Key hyperparameters include Rank $r = 16$, $\alpha = 32$, and a learning rate of $1e-4$.

G.2 ReFT Baseline

We implement Representation Finetuning (ReFT) (Wu et al., 2024a), specifically the LoReFT variant. The intervention is modeled as a low-rank projection $h + R^T(Wh + b - Rh)$ applied to hidden states. We set the rank $r = 16$ and use a higher learning rate of $4e-3$ to facilitate convergence of the intervention parameters.

G.3 CAA Baseline

We implement Contrastive Activation Addition (CAA) using the official open-source framework

provided by Panickssery et al. (2024). Steering vectors are derived by averaging activation differences between successful guided reasoning paths and zero-shot failure cases on a held-out set. These static vectors are injected into the residual stream at all post-prompt positions. Following the official protocol, the steering coefficient α is optimized via grid search on the validation set.

G.4 Pause Token Baseline

Following Goyal et al. (2023), we insert learnable `<|pause|>` tokens to allow the model to utilize extra computation steps. During training, $N = 8$ pause tokens are inserted between the prompt and the answer. The loss for these tokens is masked, allowing the model to autonomously learn their utility. We fine-tune both the embeddings and the LM head.

G.5 Soft CoT Baseline

We adopt the Soft Chain-of-Thought mechanism (Xu et al., 2025), and implement it based on the authors' official open-source codebase. It employs a small assistant model to generate "soft thoughts" for a larger base model. The hidden states from the assistant are projected to match the base model's dimension. We set the number of thought tokens to 10 and use a learning rate of $2e-5$, fine-tuning the assistant model alongside the projection layer.

Case Study: Qwen2.5-7B-Instruct

Prompt:

You are an AI assistant; you are required to solve mathematical question step by step. Please provide your final answer on a separate line in `\boxed{}` format.

Input:

A Senate committee has 5 Democrats, 5 Republicans, and 1 Independent. In how many ways can they sit around a circular table if all the members of each party all sit next to each other? (Two seatings are considered equivalent if one is a rotation of the other.)

Baseline model output:

To solve this problem, we need to consider both the arrangement of the groups (parties) around the table and the arrangement of individuals within each group.

Step 1: Arrange the Groups

Since the table is circular, fixing one group eliminates the rotational symmetry. We have three groups: Democrats (D), Republicans (R), and the Independent (I).

- Fix one group in place. There are 2 remaining groups to arrange around the table.
- The number of ways to arrange 2 groups in a circle is given by $(2 - 1)!$.

$$(2 - 1)! = 1! = 1$$

So, there is only 1 way to arrange the groups around the table once one group is fixed.

Step 2: Arrange Individuals Within Each Group

- **Democrats:** There are 5 Democrats, and they can be arranged among themselves in $5!$ ways.

$$5! = 120$$

- **Republicans:** Similarly, there are 5 Republicans, and they can be arranged among themselves in $5!$ ways.

$$5! = 120$$

- **Independent:** There is only 1 Independent, so there is only 1 way to arrange them.

Step 3: Combine the Arrangements

Now, we multiply the number of ways to arrange the groups by the number of ways to arrange the individuals within each group:

$$1 \times 120 \times 120 \times 1 = 14400$$

Thus, the total number of ways to seat the members around the table, considering that rotations are equivalent, is:

$$\boxed{14400}$$

Model Answer:

14400

Evaluate:

Incorrect!

Case Study: Our Method

Prompt:

You are an AI assistant; you are required to solve mathematical question step by step. Please provide your final answer on a separate line in `\boxed{}` format.

Input:

A Senate committee has 5 Democrats, 5 Republicans, and 1 Independent. In how many ways can they sit around a circular table if all the members of each party all sit next to each other? (Two seatings are considered equivalent if one is a rotation of the other.)

Our model output:

To solve this problem, we need to consider the arrangement of the groups (parties) and then the arrangement within each group.

1. Arranging the Groups:

Since the arrangement is circular and rotations of the same arrangement are considered identical, we fix one group in place to avoid counting rotations multiple times. We have 3 groups: Democrats (D), Republicans (R), and Independent (I).

Fixing one group, we have $2!$ ways to arrange the remaining 2 groups around the table. This is because the Independent can be placed in 2 relative positions to the fixed group (either to the left or right).

2. Arranging Members Within Each Group:

- For the Democrats, there are 5 members, and they can be arranged in $5!$ ways.
- For the Republicans, there are also 5 members, and they can be arranged in $5!$ ways.
- The Independent has only 1 member, and there is only 1 way to arrange this member.

3. Calculating the Total Number of Arrangements:

The total number of ways to arrange the groups and the members within each group is the product of the number of ways to arrange the groups and the number of ways to arrange the members within each group.

Therefore, the total number of arrangements is:

$$2! \times 5! \times 5! \times 1$$

4. Performing the Calculations:

- $2! = 2$
- $5! = 120$

Therefore, the total number of arrangements is:

$$2 \times 120 \times 120 = 28800$$

Thus, the number of ways the committee members can sit around the table, with all members of each party sitting next to each other, is $\boxed{28800}$.

Final Answer: `\boxed{28800}`

Model Answer:

28800

Evaluate:

Correct!

Case Study: Qwen2.5-1.5B-Instruct (Baseline, AIMO)

Prompt:

You are an AI assistant; you are required to solve mathematical question step by step. Please provide your final answer on a separate line in `\boxed{}` format.

Input:

Positive real numbers x and y satisfy $y^3 = x^2$ and $(y - x)^2 = 4y^2$. What is $x + y$?

Baseline model output:

To find the value of $x + y$ given the equations $y^3 = x^2$ and $(y - x)^2 = 4y^2$, we will follow these steps:

1. Express y in terms of x using the first equation:

$$y^3 = x^2 \implies y = x^{2/3}$$

2. Substitute $y = x^{2/3}$ into the second equation:

$$(y - x)^2 = 4y^2$$

Substitute $y = x^{2/3}$:

$$(x^{2/3} - x)^2 = 4(x^{2/3})^2$$

3. Simplify the equation:

$$(x^{2/3} - x)^2 = 4x^{4/3}$$

4. Take the square root of both sides:

$$x^{2/3} - x = \pm 2x^{4/3}$$

5. Check for integer solutions: By trial and error with small integers, we find:

$$x = 1 \implies y = 1^{2/3} = 1$$

Check if it satisfies the original equations:

$$y^3 = 1^3 = 1 \quad (\text{True})$$

$$(y - x)^2 = (1 - 1)^2 = 0 \quad (\text{True})$$

Therefore, $x = 1$ and $y = 1$.

6. Calculate $x + y$:

$$x + y = 1 + 1 = 2$$

Thus, the final answer is:

$$\boxed{2}$$

Model Answer:

2

Evaluate:

Incorrect!

Case Study: Qwen2.5-1.5B-Instruct (Our Method, AIMO)

Prompt:

You are an AI assistant; you are required to solve mathematical question step by step. Please provide your final answer on a separate line in `\boxed{}` format.

Input:

Positive real numbers x and y satisfy $y^3 = x^2$ and $(y - x)^2 = 4y^2$. What is $x + y$?

Our model output:

To solve the given problem, we start with the equations:

1. $y^3 = x^2$
2. $(y - x)^2 = 4y^2$

First, let's expand and simplify the second equation:

$$(y - x)^2 = 4y^2$$

Expanding the left-hand side:

$$y^2 - 2xy + x^2 = 4y^2$$

Rearrange the equation to isolate terms involving y :

$$x^2 - 2xy = 3y^2$$

Substitute $x^2 = y^3$ into the second equation:

$$y^3 - 2xy = 3y^2$$

Factor out y and divide by $y \neq 0$:

$$y^2 - 3y = 2x$$

Substitute $x = \frac{y^2 - 3y}{2}$ into $x^2 = y^3$:

$$\left(\frac{y^2 - 3y}{2}\right)^2 = y^3$$

Simplify to get:

$$y^2(y^2 - 10y + 9) = 0$$

Since $y > 0$, solve $y^2 - 10y + 9 = 0$ giving $y = 9$ or $y = 1$. Checking the original constraints yields $y = 9$ and $x = 27$.

Finally,

$$x + y = 27 + 9 = \boxed{36}$$

Model Answer:

36

Evaluate:

Correct!

Case Study: Llama-3.1-8B-Instruct (Baseline)

Prompt:

You are a Python programmer. Output ONLY the function implementation. No explanations, no markdown, no comments.

Input:

```
HumanEval_161_solve:
def solve(s: str) -> str:
    \"\\\"You are given a string s.
    if s[i] is a letter, reverse its case from lower to
    upper or vise versa,
    otherwise keep it as it is.
    If the string contains no letters, reverse the string.
    The function should return the resulted string.
Examples
>> solve('1234')
'4321'
>> solve('ab')
'AB'
>> solve('#a@c')
'#A@c'
\"\\\"
```

Baseline model output:

```
def solve(s: str) -> str:
    result = ''
    for char in s:
        if char.isalpha():
            result += char.swapcase()
        else:
            result += char
    return result[::-1]
```

Evaluate:

Incorrect!

Case Study: Llama-3.1-8B-Instruct (Our Method)

Prompt:

You are a Python programmer. Output ONLY the function implementation. No explanations, no markdown, no comments.

Input:

```
HumanEval_161_solve:
def solve(s: str) -> str:
    \"\\\"You are given a string s.
    if s[i] is a letter, reverse its case from lower to
    upper or vise versa,
    otherwise keep it as it is.
    If the string contains no letters, reverse the string.
    The function should return the resulted string.
Examples
>> solve('1234')
'4321'
>> solve('ab')
'AB'
>> solve('#a@c')
'#A@c'
\"\\\"
```

Our model output:

```
def solve(s: str) -> str:
    result = []
    for char in s:
        if char.isalpha():
            result.append(char.swapcase())
        else:
            result.append(char)
    if not any(char.isalpha() for char in s):
        return s[::-1]
    return ''.join(result)
```

Evaluate:

Correct!