Toward Latent Language Model Skills Steering and Optimization: An Empirical Study

Xunyi Jiang[1], **Junda Wu**[2], **Yuxin Xiong**[3], **Sheldon Yu**[3], **Tong Yu**[2], **David Arbour**[2], **Ritwik Sinha**[2], **Julian McAuley**[3], **Hongyi Wen**[1,*]
[1]New York University [2]Adobe Research [3]UC San Diego
{xj2511,hongyi.wen}@nyu.edu {jundaw,tyu,arbour,risinha}@adobe.com
{y7xiong,ziy040,jmcauley}@ucsd.edu
*Corresponding author.

Abstract

Skills, as a useful abstraction for the procedural capabilities of large language models (LLMs), capture how models perform structured, multi-step reasoning and program execution. Existing approaches typically treat skills as explicit, surface-level constructs specified through prompts or programs, leaving open the question of how such procedural capabilities are represented inside the model and whether they can be manipulated as structured objects in latent space. In this empirical study, we investigate whether procedural LLM skills can be *represented* as directions in activation space and whether vector-space operations over these directions can express skill-level behaviors. We find that procedural skills admit a vector-space representation: individual skill directions can be activated to shift model behavior; independently extracted directions can compose to form higher-level skills. Contrastive directions yield context-conditioned algorithmic personalization and optimization trajectories over skill directions evolve non-monotonically, with intermediate states often surpassing fully optimized solutions. These results support a representation-level view of procedural LLM skills: they admit a latent vector-space organization that allows direct manipulation through internal interventions.

1 Introduction

Skills provide a central abstraction for describing the procedural capabilities of large language models (LLMs). Skills usually can be described as structured, multi-step behaviors that govern how models perform reasoning, program execution, and flexible problem solving[1]. Recent work has studied how such skills emerge, compose, and can be reused across tasks (Chen et al., 2023; Park et al., 2025; Wei et al., 2026; Wang et al., 2026; Zhang et al., 2026). These approaches model skills as composable reasoning operators (Yin et al., 2025), executable modules (Zheng et al., 2025; Wang et al., 2025), or structured training units (Xia et al., 2026), but primarily treat them as explicit or surface-level constructs learned during training or specified through prompts (Kaur et al., 2024; Yang et al., 2025; Alzubi et al., 2026). In parallel, activation steering has shown that intervening on hidden activations can control LLM behavior at inference time (Lee et al., 2025; Cao et al., 2024; Cho et al., 2025; Scialanga et al., 2025; Li et al., 2026). However, prior steering work has largely targeted stylistic or attribute-level variation such as tone, persona, refusal, and factual associations (Arditi et al., 2024; Oozeer et al., 2025), leaving open how *procedural skills* are organized inside the model.

In this work, we ask a question: *To what extent are procedural LLM skills represented as directions in activation space, and can vector-space operations over these directions express skill-level behaviors?* We approach this question by extracting skill-conditioned directions from hidden representations with multiple steering methods, each matched to the structure of the corresponding sub-study, and by studying how these directions behave under inference-time intervention and gradient-based optimization. Compared with prior task-vector and behavioral-steering work, we focus on procedural skills with verifiable execution, where the behavioral target is the model's internal procedure rather than a stylistic surface attribute.

An LLM skill, in this study. We use *LLM skill* to refer to a procedural capability that the model applies during generation. We constructed benchmarks SKILLSET-MATH and SKILLSET-CODE. In SKILLSET-MATH, atomic skills are specified at the operation level, for example, adding two single digits, and composite skills are explicit compositions of them. In SKILLSET-CODE, a skill corresponds

[1]https://en.wikipedia.org/wiki/Skill

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 17373–17392
October 24-29, 2026 ©2026 Association for Computational Linguistics

to one element of an algorithmic strategy pair, for instance, DFS vs. BFS.

Main Findings. Our findings are organized by two properties: **steerability** and **optimizability**. At inference time, we examine whether latent interventions can activate individual skills and support their composition. Under optimization, we study whether latent skill representations can be adapted to different preferences and evolve under objectives. The main findings are:

1. Skills can be activated directly through latent steering (Section 4.2 Ⓐ).

2. Latent skill composition is effective and often outperforms prompting-based composition (Section 4.3 Ⓒ).

3. Latent skills enable skill personalization, allowing different execution strategies under varying preferences (Section 4.4 Ⓟ).

4. Skill evolution is non-monotonic, with intermediate latent states often outperforming fully optimized solutions (Section 4.5 Ⓔ).

These results suggest that procedural capabilities in LLMs are encoded as structured latent objects that can be manipulated and adapted through internal interventions. This provides a new perspective on model control, shifting from surface-level prompting to representation-level skill control.

2 Related Work

Skills in Large Language Models. Recent work has examined how skills emerge and compose in large language models (LLMs). Some approaches view skills as *atomic reasoning operations*. For example, SKiC (Chen et al., 2023) and Learning Composable Chains-of-Thought (Yin et al., 2025) represent skills as composable textual operators. Similarly, Park et al. (2025) study skill composition induced by reinforcement learning and define atomic skills as elementary operators for reasoning. Other work treats skills as executable modules in agent systems. Agent Skill Induction (Wang et al., 2025) represents skills as Python functions verified by re-execution, while SkillWeaver (Zheng et al., 2025) enables agents to discover reusable APIs. More recent work organizes skills into explicit taxonomies or structured training pipelines, including skill-taxonomy guided data synthesis (Wei et al., 2026) and instruction mixing across skill sets (Kaur et al., 2024). Overall, these approaches focus on *learning* or *discovering* skills during training, and treat skills as explicit or surface-level constructs rather than latent representations that can be directly manipulated in the model's internal space.

Skill Discovery and Skill Libraries in Agents. Another line of work studies skill acquisition in autonomous agents. Automated Skill Discovery (Yang et al., 2025) enables agents to iteratively explore environments and refine reusable capabilities through interaction and feedback. Self-improving agents with skill libraries store executable programs that can be reused across tasks (Wang et al., 2026). Recent work further investigates automated skill evolution, where agents iteratively refine and expand their capabilities through exploration and feedback, including failure-driven refinement mechanisms (Alzubi et al., 2026). Memory-centric and embodied settings further study how skills evolve and transfer across tasks. For example, MemSkill (Zhang et al., 2026) models memory operations as learnable skills, while SkillRL (Xia et al., 2026) evolves a hierarchical skill bank during reinforcement learning. Collectively, these approaches emphasize skills as reusable and evolving units that support generalization and efficiency in agent systems.

Activation Steering in Language Models. Activation steering methods manipulate hidden representations at inference time to control model behavior without modifying model parameters. Prior work derives steering vectors from contrastive examples or learned representations. Representative methods include Contrastive Activation Addition (Rimsky et al., 2024), Representation Engineering (Zou et al., 2023), and Conditional Activation Steering (Lee et al., 2025), which construct or condition steering directions from contrastive prompts and conceptual axes. More recently, CorrSteer (Cho et al., 2025) extracts steering directions from sparse autoencoder features, while steering vector fields introduce context-dependent control directions (Li et al., 2026). Activation steering has also been applied to knowledge editing and behavioral control. SAKE (Scialanga et al., 2025) performs knowledge editing via activation transport, and persona-based methods control stylistic or personality traits through activation vector algebra (Chen et al., 2025; Pai et al., 2026; Feng et al., 2026). However, existing approaches primarily focus on controlling *style*, *persona*, or *fac-*

tual attributes, rather than procedural capabilities. Whether activation steering can directly control *skills*, for example, task execution modes, remains largely unexplored. Rather than proposing a new steering mechanism, we address this gap with an empirical study of skill-level steering across tasks and domains.

3 Problem Formulation

We formalize the object of study, **latent LLM skill directions**, as vectors in activation space that capture procedural behavior in a pretrained language model. Our central question is whether such directions, extracted from skill-conditioned hidden representations, can be *activated*, *composed*, *personalized*, and *evolved* via internal interventions. This perspective recasts skills as representation-level controls rather than surface-level specifications. We then state four observable properties used throughout the paper: **steerability** Ⓐ **(Section 3.2)** and **composition** Ⓒ at inference time, and **personalization** Ⓟ and **evolution (Section 3.3)** Ⓔ under gradient-based optimization.

3.1 Latent LLM Skill

Definition 3.1 (Latent LLM Skill Direction)**.** *Let f_θ denote a pretrained language model with hidden representations $\{h_\ell(x)\}_{\ell=1}^{L}$, where $h_\ell(x) \in \mathbb{R}^{d_\ell}$ is the hidden state at layer ℓ for input x, and let $\mathcal{S} \subseteq \{1, \ldots, L\}$ denote the set of intervened layers. A latent LLM skill direction is a collection of layerwise intervention vectors*

$$v = \{v_\ell\}_{\ell \in \mathcal{S}}, \qquad v_\ell \in \mathbb{R}^{d_\ell}, \tag{1}$$

which, together with a scalar activation strength $\alpha \in \mathbb{R}$, defines intervened representations $\tilde{h}_\ell$ and the corresponding skill-conditioned model $f_\theta^{(v,\alpha)}$ by

$$\begin{aligned} \tilde{h}_\ell(x;\alpha) &= \begin{cases} h_\ell(x) + \alpha v_\ell, & \ell \in \mathcal{S}, \\ h_\ell(x), & \ell \notin \mathcal{S}, \end{cases} \\ f_\theta^{(v,\alpha)}(x) &:= f_\theta\Big(x; \{\tilde{h}_\ell(x;\alpha)\}_{\ell=1}^{L}\Big) \end{aligned} \tag{2}$$

Intuitively, v specifies a direction in representation space, and α controls its activation strength. Thus, a latent skill direction intervenes on hidden states during the forward pass without modifying the model parameters, and we write $\hat{y} \sim f_\theta^{(v,\alpha)}(x)$ for the resulting output. However, not every direction in hidden-state space is behaviorally meaningful. We therefore introduce a validity criterion that requires a candidate direction to induce a non-trivial change in the expected task objective when activated.

Definition 3.2 (Valid Skill Direction)**.** *A latent LLM skill direction v is said to be valid on a task distribution $\mathcal{D}$ if*

$$\begin{aligned} &\mathcal{J}(v,\alpha) := \mathbb{E}_{x\sim\mathcal{D}}\big[\mathcal{J}_0(f_\theta^{(v,\alpha)}(x))\big], \\ &\text{s.t.} \quad \frac{\partial \mathcal{J}(v,\alpha)}{\partial \alpha}\Big|_{\alpha=0} \neq 0, \end{aligned} \tag{3}$$

where $\mathcal{J}_0$ is a sample-level task objective (e.g., the indicator of an incorrect answer). Equivalently, an infinitesimal activation of v produces a first-order change in the expected task behavior.

We use $\mathcal{J}$ for the expected task objective throughout to avoid overloading $\mathcal{L}$, which we reserve for indexing layers (e.g., the layer set $\mathcal{S}$ above and the model depth L).

3.2 Latent LLM Skill Steering

We next study whether such skill directions can act as effective inference-time controls. Latent activation steering enables a steering vector to be added to intermediate hidden states to modulate model behavior (Subramani et al., 2022; Turner et al., 2023; Rimsky et al., 2024). In our setting, however, the target of control is a procedural *LLM skill*. We therefore formulate latent skill steering as the problem of whether activating a valid skill direction can produce a measurable gain in task performance, and whether multiple such steerable directions can be further composed in latent space.

Ⓐ **Steerability.** Intuitively, steerability requires that v act as a meaningful latent control direction: changing the activation strength α should not merely perturb hidden states, but should improve the expected task objective over the unsteered model.

Definition 3.3 (Latent Skill Steerability)**.** *Let v be a valid skill direction according to Equation (3). We say that v is* steerable *on $\mathcal{D}$ if*

$$\exists \alpha \in \mathbb{R} \quad \text{s.t.} \quad \Delta(v,\alpha) := \mathcal{J}(0,0) - \mathcal{J}(v,\alpha) > 0. \tag{4}$$

Because the intervention is continuous in α, this also captures graded inference-time control. In our evaluation, we instantiate the task objective $\mathcal{J}$ as the error rate over the dataset. Accordingly, $\Delta(v,\alpha)$ corresponds to the reduction in error rate induced by activating the latent skill direction. Empirically, this is measured as accuracy improvement over the unsteered baseline.

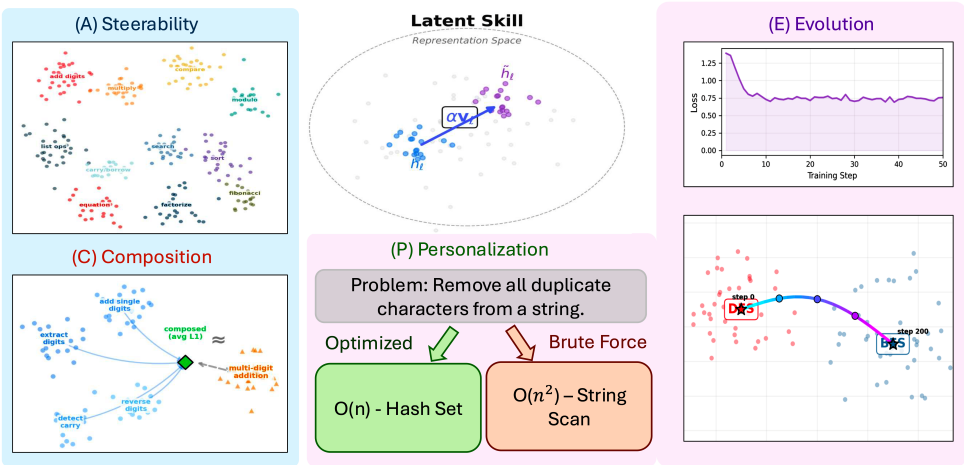

Figure 1: **Latent LLM Skills as Representation-Level Objects.** We study four observable properties of skill-conditioned directions in activation space: steerability, composition, personalization, and evolution.

Ⓒ **Skill Composition.** Once individual skill directions are steerable, we further study whether they can be jointly activated to express composite skills. Compositional metric learning constructs richer representations by recombining learned primitives or sub-embeddings (Atzmon et al., 2020; Naeem et al., 2021; Zheng et al., 2021; Kaya and Bilge, 2019). Motivated by this compositional view, we define vector-space skill composition through joint hidden-state intervention and characterize it by the resulting interaction gain between two steerable directions.

Definition 3.4 (Vector-Space Skill Composition)**.** *Let v_i and v_j be two steerable skill directions. We define their joint intervention by*

$$\tilde{h}_\ell(x;\alpha_i,\alpha_j) = h_\ell(x) + \alpha_i v_{i,\ell} + \alpha_j v_{j,\ell},$$
$$f_\theta^{(v_i,v_j;\alpha_i,\alpha_j)}(x) := f_\theta\Big(x;\{\tilde{h}_\ell(x;\alpha_i,\alpha_j)\}_{\ell=1}^{L}\Big), \tag{5}$$

and their interaction gain by

$$\begin{aligned}\Delta_i &:= \mathcal{J}(0,0) - \mathcal{J}(v_i,\alpha_i),\\ \Delta_{ij} &:= \mathcal{J}(0,0) - \mathcal{J}(v_i,v_j;\alpha_i,\alpha_j),\\ \Gamma_{ij} &:= \Delta_{ij} - \Delta_i - \Delta_j.\end{aligned} \tag{6}$$

We say that v_i and v_j admit vector-space composition *if*

$$\exists \alpha_i,\alpha_j \in \mathbb{R} \quad s.t. \quad \Gamma(v_i,v_j;\alpha_i,\alpha_j) \neq 0. \tag{7}$$

Intuitively, Γ measures whether the effect of jointly activating two skill directions is reducible to the sum of their individual effects. In our evaluation, we do not explicitly estimate Γ. Instead, we assess composition by testing whether composed skill directions improve performance on composite tasks beyond the unsteered baseline and, more importantly, beyond *text-based composition* baselines. This comparison is the central axis of our composition study: we ask whether vector-space composition expresses something that text-based composition cannot.

3.3 Latent LLM Skill Optimization

We next study whether latent LLM skill directions can be *optimized* as continuous objects under downstream objectives (Zou et al., 2023). While latent skill steering asks whether a valid skill direction can be activated as an inference-time control variable, latent skill optimization asks whether the same latent object can be improved through gradient-based updates in latent space. To this end, let Φ denote a parameterization of a latent skill direction, with v_Φ denoting the corresponding intervention. Given a downstream objective $\mathcal{J}_{\text{opt}}(\Phi;\sigma)$ under contextual preference σ, we perform gradient-based updates

$$\begin{aligned}\Phi^{(t+1)} &= \Phi^{(t)} - \eta\nabla_\Phi \mathcal{J}_{\text{opt}}(\Phi^{(t)};\sigma),\\ \mathcal{T} &= \{\Phi^{(0)},\Phi^{(1)},\ldots,\Phi^{(T)}\}.\end{aligned} \tag{8}$$

Based on this optimization view, we study *personalized* skill directions and the *evolution* of skill directions along training trajectories, complementing prior work on optimization of representation interventions (Wu et al., 2024; Cho et al., 2025; Wu et al., 2025; Subramani et al., 2022).

Ⓟ **Skill Personalization.** If skill directions are optimizable, then conditioning optimization on different contextual preferences should yield

distinct but still task-effective configurations. We therefore formulate latent skill personalization as a preference-conditioned optimization problem in latent space, instantiated in this paper for paired algorithmic strategies (e.g., DFS vs. BFS).

Definition 3.5 (Latent Skill Personalization)**.** *Given an initial latent skill parameterization $\Phi^{(0)}$ and a contextual preference σ, the latent skill personalization problem is to solve*

$$\begin{aligned} \Phi^*_\sigma &= \arg\min_{\Phi} \mathcal{J}_{\text{opt}}(\Phi;\sigma) \\ \text{s.t.} \quad & \exists\, \alpha \in \mathbb{R} \text{ with } \Delta(v_\Phi, \alpha) > 0. \end{aligned} \tag{9}$$

For two preferences $\sigma_i \neq \sigma_j$, personalization is realized when the corresponding solutions $\Phi^*_{\sigma_i}$ and $\Phi^*_{\sigma_j}$ induce distinguishable preference-aligned behaviors while remaining task-effective.

This formulation asks whether the same underlying latent skill direction can be specialized into multiple preference-conditioned variants while preserving its utility on the task.

(E) **Skill Evolution.** A natural follow-up question is whether the *optimization trajectory* itself contains additional structure—specifically, whether intermediate checkpoints correspond to meaningful and progressively improved skill configurations. Prior work on representation-intervention optimization has reported non-monotonic dynamics: steering vectors can over-fit late in training, with intermediate checkpoints generalizing better than the final ones (Wu et al., 2025; Subramani et al., 2022). The following definition is the *operational* definition of skill-evolution behavior used in this empirical study; we do not claim it as a theoretical contribution, but use it to make the trajectory-level evaluation in Section 4.5 precise.

Definition 3.6 (Latent Skill Evolution)**.** *Given an initial latent skill parameterization $\Phi^{(0)}$, a downstream objective $\mathcal{J}_{\text{opt}}$, and the induced optimization trajectory*

$$\begin{aligned} \Phi^{(t+1)} &= \Phi^{(t)} - \eta \nabla_\Phi \mathcal{J}_{\text{opt}}(\Phi^{(t)};\sigma), \\ \mathcal{T} &= \{\Phi^{(0)}, \Phi^{(1)}, \ldots, \Phi^{(T)}\}, \end{aligned} \tag{10}$$

we say the trajectory exhibits (non-monotonic) skill evolution *if there exist optimization states $\Phi^{(t_1)}$ and $\Phi^{(t_2)}$ with $t_1 < t_2$ such that*

$$\mathcal{M}(\Phi^{(t_2)}) < \mathcal{M}(\Phi^{(t_1)}), \tag{11}$$

where $\mathcal{M}$ is a downstream evaluation metric.

Taken together, this formulation views a latent LLM skill direction as a representation-level object endowed with **steerability** and **optimizability**, and four observable signatures: (A) steerability, (C) composition, (P) personalization, and (E) evolution. These four signatures define the scope of our evaluation in Section 4.

4 Empirical Study of Vector-Space Skill Representations

Our experiments empirically probe the four observable properties of latent LLM skill directions defined in Section 3: (A) steerability, (C) composition, (P) personalization, and (E) evolution. We organize them into two experimental settings.

Controlled foundational study. Skill directions are extracted and evaluated on our constructed benchmarks SKILLSET-MATH (RQ1, RQ2) and SKILLSET-CODE (RQ3, RQ4), where each skill is procedurally defined and the desired behavior is verifiable.

Real-benchmark evaluation. The math-skill directions extracted in SKILLSET-MATH are applied to the standard math reasoning benchmarks GSM8K and MATH500, across Qwen3 models from 0.6B to 14B (Team, 2025). In this real-benchmark setting, we ask whether the same vector-space directions can transfer beyond the controlled benchmark.

4.1 Datasets and Models

Controlled Skill Datasets. We construct two domain-specific datasets with explicit skill annotations: SKILLSET-MATH ($\mathcal{D}_{\text{math}}$) and SKILLSET-CODE ($\mathcal{D}_{\text{code}}$), with details in Section A.

SKILLSET-MATH is a math-reasoning dataset with two levels of granularity: atomic skills (e.g., `add_two_single_digits`, `detect_carry`) and composite skills (e.g., `multi_digit_addition`) formed by composing multiple atomic skills. This structure enables controlled evaluation of (A) steerability and (C) composition, since we can directly test whether single skill directions induce the intended atomic behaviors and whether independently extracted directions compose to solve higher-level tasks.

SKILLSET-CODE consists of 9 contrastive paired skills, where each pair represents two valid but distinct algorithmic approaches to the same problem (e.g., DFS vs. BFS). This dataset is used to evaluate (P) personalization and (E) evolution.

Both benchmarks are controlled by design. Isolable skills, known compositional structure, and verifiable outputs are what make the four properties measurable at all, and open-ended benchmarks do not expose this structure. We therefore do not claim they exhaust the complexity of procedural skills in practice, but rather treat them as a testbed for establishing whether such skills have a manipulable representation-level structure.

Real Benchmarks. For the real-benchmark evaluation, the skill directions extracted on SKILLSET-MATH are transferred to GSM8K and MATH500.

Models. In the controlled foundational study, we use Qwen3-0.6B and Qwen3-1.7B. In the real-benchmark evaluation, we use Qwen3 models ranging from 0.6B to 14B (Team, 2025).

Skill-Direction Extraction. All experiments use the extraction procedures formalized in Section C: single-class PCA for math skills (RQ1 and RQ2) and contrastive mean-difference for paired algorithmic strategies (RQ3); RQ4 optimizes a parameterized skill direction along training trajectories.

4.2 (A) Do skill directions applied at inference time improve LLM math reasoning?

Setup We extract one PCA-based skill direction v per skill on Qwen3-0.6B, using the procedure in Section C. Each direction is extracted at every layer and applied to the final 50% of layers with $\alpha = 2.0$. Evaluation is on the test split of $\mathcal{D}_{\text{math}}$ (100 samples per skill), compared to the zero-shot baseline $\text{Acc}(0, 0; \mathcal{D}_{\text{math}})$.

Result 1: Skill directions shift behavior on SKILLSET-MATH for Qwen3-0.6B. Figure 2 compares per-skill accuracy on $\mathcal{D}_{\text{math}}$ between the unsteered baseline and the steered model for Qwen3-0.6B. The PCA-extracted direction improves accuracy on 29 of 34 skills, with an average gain of 19.1 percentage points. The largest gains appear in the composite skills, e.g., `algebraic_simplification` (+86 points) and `solve_linear_equation` (+58 points), suggesting that skill-conditioned directions help activate multi-step procedures that the model fails to execute under zero-shot prompting.

Result 2: Math-skill directions transfer to real benchmarks across model sizes. Table 1 reports zero-shot through 4-shot prompting and our **Steered** configuration (the math-skill direction is applied at inference time on GSM8K and MATH500) across Qwen3 models from 0.6B to 14B. We highlight two qualitative observations. First, few-shot prompting can be *unstable* on smaller models; for example, 1-shot prompting reduces Qwen3-0.6B's GSM8K accuracy from 74.0% to 46.7%. The Steered configuration does not exhibit such degradation and matches or exceeds the zero-shot baseline in 8 out of 10 (model, dataset) cells. Second, on the larger models (4B, 8B, 14B) the Steered numbers are competitive but modest relative to the strongest prompting baseline, indicating that the absolute headroom for this kind of intervention shrinks as the underlying model becomes more capable on the benchmark.

Model	0-shot	1-shot	2-shot	3-shot	4-shot	Steered
GSM8K						
Qwen3-0.6B	74.00	46.70	65.35	69.37	64.74	**75.66**
Qwen3-1.7B	82.18	78.17	77.94	77.94	78.92	**83.93**
Qwen3-4B	**93.56**	92.42	90.30	90.37	91.13	92.34
Qwen3-8B	95.07	94.77	**95.45**	94.84	95.30	95.38
Qwen3-14B	95.83	95.60	95.98	**96.97**	95.98	96.21
MATH500						
Qwen3-0.6B	**67.00**	65.20	63.80	65.40	51.00	66.20
Qwen3-1.7B	72.60	65.80	67.60	70.40	68.20	**73.80**
Qwen3-4B	89.60	80.80	84.20	80.60	81.40	**91.00**
Qwen3-8B	89.80	**92.00**	89.80	88.40	90.20	90.60
Qwen3-14B	92.40	91.60	90.80	90.80	91.00	**93.80**

Table 1: Accuracy (%) on real math benchmarks across the Qwen3 models from 0.6B to 14B. **Steered** is the model with one PCA-extracted SKILLSET-MATH skill direction added to hidden states at $\alpha = 2.0$ on the last 50% of layers.

4.3 (C) Does vector-space composition outperform text-based composition?

Setup. Each composite skill is associated with a set of k constituent atomic skill directions. We approximate joint activation by composing these atomic directions with equal weights $\alpha_i = 1/k$ and re-normalizing: $v_{\text{comp}} = \frac{\sum_{i=1}^{k} \alpha_i v_i}{\|\sum_{i=1}^{k} \alpha_i v_i\|}$. We compare vector-space composition $f_\theta^{(v_{\text{comp}}, \alpha)}$ against: (i) the unsteered zero-shot baseline $f_\theta^{(0,0)}$ (**Base**); (ii) **text-concise** (Txt-C), where the composite prompt is augmented with a concise text description; (iii) **text-CoT** (Txt-CoT), where the prompt is augmented with a fuller chain-of-thought text decomposition (see Section B.2 for prompt examples). This comparison is the central axis of RQ2: *text-based composition vs. vector-based composition*. Because we do not directly estimate the interaction gain Γ_{ij} of Section 3.2, we use an empirical proxy $\tilde{\Gamma} := \text{Acc}(v_{\text{comp}}, \alpha; \mathcal{D}) - \text{Acc}_{\text{Txt-CoT}}(\mathcal{D})$, where

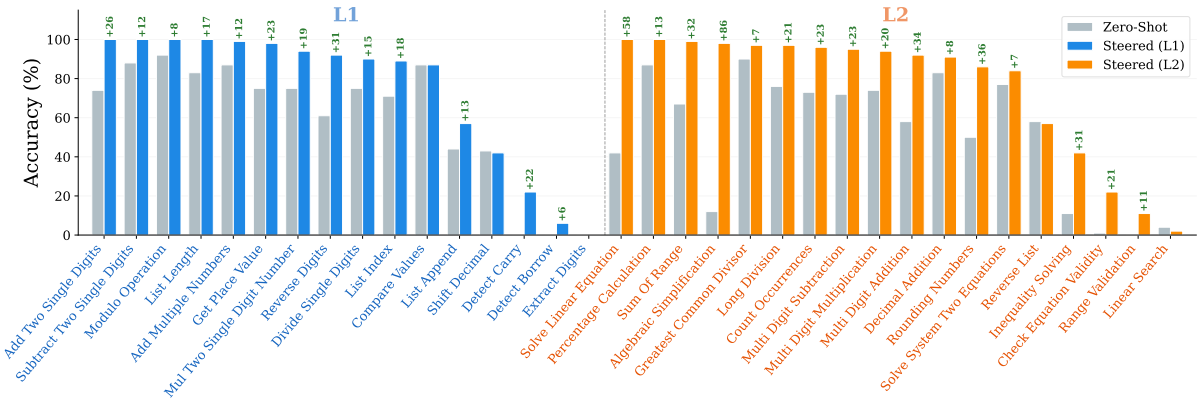

Figure 2: Per-skill accuracy on SKILLSET-MATH for Qwen3-0.6B: unsteered baseline vs. steered (PCA-extracted skill direction).

$\tilde{\Gamma} > 0$ indicates that vector-space composition outperforms text-based composition.

Result: Vector-space composition outperforms text-based composition on most composite skills. Vector composition achieves the highest accuracy on 13 of 18 skills (Table 2), improving the average from 51.9% (Base) to 77.1% (+25.2 points), compared to +23.8 points for Txt-C (text-concise) and +22.3 points for Txt-CoT (text-CoT). The gains are most pronounced on procedurally complex tasks, such as `solve_system_two_equations` and `inequality_solving`. Importantly, the composed direction v_{comp} is constructed only from atomic directions, with no composite skill supervision; the gain is therefore evidence that independently extracted skill directions admit vector-space recombination that text-based decomposition does not match.

4.4 (P) Do skill directions enable context-conditioned personalization?

Setup. We evaluate whether preference-conditioned skill directions, estimated via contrastive hidden-state differences (Section C), can induce context-dependent algorithmic behavior at inference time. SKILLSET-CODE consists of two components: (i) a controlled evaluation set of 36 hand-crafted tasks with paired solution strategies and situational contexts σ, yielding 72 task–situation instances; and (ii) a contrastive training set for direction estimation, where each approach pair defines two context-conditioned distributions $\mathcal{D}_{\sigma_1}$ and $\mathcal{D}_{\sigma_2}$, each containing 200 solutions. These distributions implicitly specify the preference-conditioned objective $\mathcal{J}_{\text{opt}}(\cdot;\sigma)$ used to estimate skill directions. We instantiate the latent skill parameterization by representing each context-conditioned skill as the expectation of hidden states, $\Phi_\sigma := \mathbb{E}_{h\sim\mathcal{D}_\sigma}[h]$, so that the latent direction between two contexts is given by their difference, $v = \Phi_{\sigma_1} - \Phi_{\sigma_2}$. At inference time, the situational context σ determines the activation direction of the skill direction, selecting either $+v$ or $-v$. For example, in the DFS vs. BFS pair, a memory-constrained setting favors DFS, whereas a large-scale search setting favors BFS. We evaluate Qwen3-0.6B and Qwen3-1.7B.

Evaluation. We measure preference alignment using GPT-4o-mini as an LLM judge (the full judge prompt is given in Section B.3) that determines whether the generated code follows the prescribed algorithmic *approach* (e.g., DFS vs. BFS), and report *approach match rate*, the fraction of records where the judge confirms the prescribed approach.

Result: Contrastive directions enable preference controllability. In Table 3, steering the LLM with the skill vector improves the match rate across categories on Qwen3-0.6B, indicating that the contrastively estimated direction v generalizes to unseen task–situation instances.

4.5 (E) How do skill directions evolve along optimization trajectories?

Setup. We study how a latent skill direction evolves along its optimization trajectory, where gradient-based updates provide a natural mechanism for continuously transforming the direction under a downstream objective. Starting from a zero initialization $\Phi^{(0)}$, we optimize the skill-direction parameterization using Reference-free Preference Steering (RePS) (Wu et al., 2025) for $T = 200$

Skill	Base	Txt-C	Txt-CoT	Vec	Skill	Base	Txt-C	Txt-CoT	Vec
multi_digit_mul	74	94	93	96↑	gcd	90	97	97	99↑
multi_digit_add	58	92	82	93↑	range_valid	0	11	11	5↓
multi_digit_sub	72	95	98	98=	linear_search	4	2	2	7↑
sum_of_range	67	99	97	96↓	reverse_list	58	57	58	52↓
decimal_add	83	91	93	95↑	alg_simplify	12	98	98	99↑
solve_sys_2eq	77	84	86	96↑	count_occur	73	96	96	94↓
rounding_num	50	86	86	88↑	check_eq_valid	1	22	3	24↑
long_division	76	97	100	100=	solve_linear_eq	42	100	100	100=
pct_calc	87	100	99	99=	inequality_solv	11	42	36	47↑
					Average: Base 51.9	Txt-C 75.7	Txt-CoT 74.2		**Vec 77.1**

Table 2: RQ2: text-based vs. vector-based composition on Qwen3-0.6B (accuracy %). **Base**: zero-shot. **Txt-C/Txt-CoT**: text-based composition (concise / chain-of-thought). **Vec**: composition of atomic skill directions in vector space. Highlighted cells indicate the best-performing method per skill.

	Qwen3-0.6B		Qwen3-1.7B	
Category	**Base**	**Steered**	**Base**	**Steered**
Graph Search	16.7	**33.3**	25.0	16.7
DP / Recursion	16.7	**41.7**	8.3	8.3
Data Struct. / String	16.7	**45.8**	8.3	**20.8**

Table 3: Approach match rate (%, controllability) by category, $\alpha = 1.0$.

		Qwen3-0.6B			Qwen3-1.7B		
Cat.	**Pair**	Base	Peak	Step	Base	Peak	Step
Graph Search	DFS vs BFS	8	**24**†	20	10	14†	5
	DFS vs UnionFind	6	**22**	200	22	**28**†	15
	BFS vs Dijkstra	22	22†	0	18	22	200
	BFS vs BiDirBFS	64	**72**†	9	74	**84**†	30
DP / Recursion	Recursion vs DP	0	**24**	200	2	6†	100
	QuadDP vs Opt.	40	**58**†	100	52	**60**†	4
Data Struct.	Stack vs Recursion	4	**16**†	8	42	**54**	200
	BruteForce vs Opt.	10	12†	20	22	26†	1
	IndexScan vs TwoPtr	10	14†	1	14	**20**†	9

Table 4: Approach match rate (%, controllability) at initialization (**Base**, step 0) and at the best observed checkpoint during training (**Peak**), with the corresponding training step (**Step**). **Bold** = gain ≥ 5 pp over baseline. † Peak occurs strictly before the final checkpoint (step 200), indicating that continued optimization *degrades* performance.

steps, yielding a trajectory $\mathcal{T} = \{\Phi^{(t)}\}_{t=0}^{T}$ and recording the corresponding direction $v_{\Phi^{(t)}}$ at each step. At each checkpoint, we evaluate the direction using the *approach match rate* (the same controllability metric defined in Section 4.4), via the LLM-as-judge in Section B.4. This trajectory-based evaluation lets us characterize the evolution dynamics of skill directions: in particular, whether quality improves monotonically along optimization or peaks at intermediate states.

Result: Optimal skill directions emerge before convergence. In 14 of the 18 (pair, model) settings, the highest in-distribution approach match rate is achieved before the final step 200 rather than at convergence. On Qwen3-0.6B, 7 of 9 pairs peak strictly before step 200 (†), with early gains such as *DFS vs. BFS* ($8\% \rightarrow 24\%$ at step 20, then declining to 18%) and *Stack vs. Recursion* ($4\% \rightarrow 16\%$ at step 8, then falling to 10%). A similar pattern holds on Qwen3-1.7B (e.g., *QuadDP vs. Opt.* $52\% \rightarrow 60\%$ at step 4, then degrading to 42%). This trajectory-level non-monotonicity is consistent with the prior observations above: early updates appear to capture a broad, transferable skill direction, while continued optimization drives the direction toward task-specific features that degrade the approach match rate at convergence.

5 Conclusion

In this work, we studied whether procedural skills in large language models can be *represented* as directions in activation space and whether vector-space operations over these directions can express skill-level behaviors that text-based control cannot. Through controlled experiments on SKILLSET-MATH and SKILLSET-CODE, and a real-benchmark transfer evaluation on GSM8K and MATH500 across Qwen3 0.6B–14B, we examined four observable properties of vector-space skill representations: **steerability**, **composition**, **personalization**, and **evolution**. Empirically, we found that (i) skill-conditioned directions extracted via PCA enable direct activation of procedural behaviors on SKILLSET-MATH and transfer to real math benchmarks; (ii) independently extracted

atomic directions can be composed in vector space to express composite skills, often outperforming text-based composition; (iii) contrastive directions over paired algorithmic strategies enable context-conditioned approach controllability; and (iv) directions optimized along training trajectories evolve non-monotonically, with intermediate states frequently outperforming fully converged solutions.

These findings support a representation-level view of procedural LLM skills: such skills admit a vector-space organization in the model's activations, complementing surface-level control via prompting.

6 Limitations

We close by noting the scope of this work. As an empirical study, our claims are bounded by the controlled SKILLSET benchmarks and the Qwen3 models we evaluate, and several natural extensions to a fuller treatment of latent skills remain open; we highlight the most direct one here. Our composition study only considers equal-weighted, additive composition of atomic skill directions. A natural extension we leave to future work is richer arithmetic over skill directions, for example, $v(\text{multi_digit_add}) - v(\text{single_digit_add}) + v(\text{single_digit_mul})$, which would test whether the geometry of these directions encodes the underlying procedural relationships beyond a plain sum.

Acknowledgments

This work is partially supported by NSF IIS-2432486. We thank the anonymous reviewers for their constructive feedback. AI assistants were used for writing assistance, including polishing portions of the related work, experimental discussion, and appendix text, and for generating plotting. All content was reviewed and verified by the authors. The use of LLMs in dataset construction and evaluation is described in Section A.

References

Sheikh Abdur Raheem Ali, Justin Xu, Ivory Yang, Jasmine Xinze Li, Ayse Arslan, and Clark Benham. 2025. Scaling laws for activation steering with llama 2 models and refusal mechanisms. *CoRR*, abs/2507.11771.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. 2026. Evoskill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*.

Andy Arditi, Oscar Obeso, Aaquib Syed, Daniel Paleka, Nina Panickssery, Wes Gurnee, and Neel Nanda. 2024. Refusal in language models is mediated by a single direction. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Yuval Atzmon, Felix Kreuk, Uri Shalit, and Gal Chechik. 2020. A causal view of compositional zero-shot recognition. *Advances in Neural Information Processing Systems*, 33:1462–1473.

Yuanpu Cao, Tianrong Zhang, Bochuan Cao, Ziyi Yin, Lu Lin, Fenglong Ma, and Jinghui Chen. 2024. Personalized steering of large language models: Versatile steering vectors through bi-directional preference optimization. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Jiaao Chen, Xiaoman Pan, Dian Yu, Kaiqiang Song, Xiaoyang Wang, Dong Yu, and Jianshu Chen. 2023. Skills-in-context prompting: Unlocking compositionality in large language models. *CoRR*, abs/2308.00304.

Runjin Chen, Andy Arditi, Henry Sleight, Owain Evans, and Jack Lindsey. 2025. Persona vectors: Monitoring and controlling character traits in language models. *CoRR*, abs/2507.21509.

Seonglae Cho, Zekun Wu, and Adriano Koshiyama. 2025. Corrsteer: Generation-time llm steering via correlated sparse autoencoder features. *Preprint*, arXiv:2508.12535.

Xiachong Feng, Liang Zhao, Weihong Zhong, Yichong Huang, Yuxuan Gu, Lingpeng Kong, Xiaocheng Feng, and Bing Qin. 2026. PERSONA: dynamic and compositional inference-time personality control via activation vector algebra. *CoRR*, abs/2602.15669.

Simran Kaur, Simon Park, Anirudh Goyal, and Sanjeev Arora. 2024. Instruct-skillmix: A powerful pipeline for llm instruction tuning. *arXiv preprint*.

Mahmut Kaya and Hasan Şakir Bilge. 2019. Deep metric learning: A survey. *Symmetry*, 11(9):1066.

Bruce W. Lee, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Erik Miehling, Pierre L. Dognin, Manish Nagireddy, and Amit Dhurandhar. 2025. Programming refusal with conditional activation steering. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Jiaqian Li, Yanshu Li, and Kuan-Hao Huang. 2026. Steering vector fields for context-aware inference-time control in large language models. *CoRR*, abs/2602.01654.

Muhammad Ferjad Naeem, Yongqin Xian, Federico Tombari, and Zeynep Akata. 2021. Learning graph embeddings for compositional zero-shot learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 953–962.

Narmeen Oozeer, Luke Marks, Fazl Barez, and Amirali Abdullah. 2025. Beyond linear steering: Unified multi-attribute control for language models. *Findings of the Association for Computational Linguistics: EMNLP*, pages 23513–23557.

Tsung-Min Pai, Jui-I Wang, Li-Chun Lu, Shao-Hua Sun, Hung-yi Lee, and Kai-Wei Chang. 2026. BILLY: steering large language models via merging persona vectors for creative generation. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2026 - Volume 1: Long Papers, Rabat, Morocco, March 24-29, 2026*, pages 7870–7915. Association for Computational Linguistics.

Simon Park, Simran Kaur, and Sanjeev Arora. 2025. How does rl post-training induce skill composition? a case study on countdown. *Preprint*, arXiv:2512.01775.

Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Turner. 2024. Steering llama 2 via contrastive activation addition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15504–15522, Bangkok, Thailand. Association for Computational Linguistics.

Marco Scialanga, Thibault Laugel, Vincent Grari, and Marcin Detyniecki. 2025. SAKE: steering activations for knowledge editing. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 15966–15978. Association for Computational Linguistics.

Nishant Subramani, Nivedita Suresh, and Matthew E Peters. 2022. Extracting latent steering vectors from pretrained language models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 566–581.

Qwen Team. 2025. Qwen3 technical report. *CoRR*, abs/2505.09388.

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*.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. 2026. Reinforcement learning for self-improving agent with skill library. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pages 1529–1550. Association for Computational Linguistics.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. 2025. Inducing programmatic skills for agentic tasks. *CoRR*, abs/2504.06821.

Yifan Wei, Li Du, Xiaoyan Yu, Yang Feng, and Angsheng Li. 2026. Towards compositional generalization of llms via skill taxonomy guided data synthesis. *CoRR*, abs/2601.03676.

Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. 2024. Reft: Representation finetuning for language models. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Zhengxuan Wu, Qinan Yu, Aryaman Arora, Christopher D. Manning, and Christopher Potts. 2025. Improved representation steering for language models. *CoRR*, abs/2505.20809.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. 2026. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *CoRR*, abs/2602.08234.

Haolei Xu, Xinyu Mei, Yuchen Yan, Rui Zhou, Wenqi Zhang, Weiming Lu, Yueting Zhuang, and Yongliang Shen. 2025. Easysteer: A unified framework for high-performance and extensible LLM steering. *CoRR*, abs/2509.25175.

Yongjin Yang, Sinjae Kang, Juyong Lee, Dongjun Lee, Se-Young Yun, and Kimin Lee. 2025. Automated skill discovery for language agents through exploration and iterative feedback. *Preprint*, arXiv:2506.04287.

Fangcong Yin, Zeyu Leo Liu, Liu Leqi, Xi Ye, and Greg Durrett. 2025. Learning composable chains-of-thought. *CoRR*, abs/2505.22635.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. 2026. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. 2025. Skillweaver: Web agents can self-improve by discovering and honing skills. *CoRR*, abs/2504.07079.

Wenzhao Zheng, Chengkun Wang, Jiwen Lu, and Jie Zhou. 2021. Deep compositional metric learning. In

Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 9320–9329.

Andy Zou, Long Phan, Sarah Li Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, Shashwat Goel, Nathaniel Li, Michael J. Byun, Zifan Wang, Alex Mallen, Steven Basart, Sanmi Koyejo, Dawn Song, Matt Fredrikson, and 2 others. 2023. Representation engineering: A top-down approach to AI transparency. *CoRR*, abs/2310.01405.

A SkillSet Construction

SKILLSET consists of two domains: math and code, designed for controlled evaluation of latent skill properties.

A.1 Programmatic Synthesis

We construct SKILLSET as a controlled benchmark for skill-level evaluation. In the current setup, SKILLSET-MATH contains 16 atomic skills and 18 composite skills. Composite skills are compositions of atomic skills as shown in Table 5. The construction of SkillSet can be divided into three parts: the instruction for each skill, the question, and the answer. For composite skills, we use deterministic code to generate the answer. Each skill contains 1000 questions, split into training, validation, and testing sets with an 8:1:1 ratio. In later sections, we use the training data to construct steering vectors and the testing set to evaluate the effectiveness of steering.

SkillSet-Math We construct **SkillSet-Math** as a hierarchical math-reasoning benchmark with two levels.

Atomic contains 16 atomic skills, including single-digit arithmetic (addition, subtraction, multiplication, division), digit manipulation (extraction, reversal), carry/borrow detection, list primitives (indexing, length, append), and value comparison. **Composite** contains 18 composite skills, covering multi-digit arithmetic (addition, subtraction, multiplication, long division), decimal and percentage computation, algebraic tasks (linear equations, systems of two equations, inequality solving, simplification), number-theoretic routines (GCD, range validation, rounding), and list algorithms (linear search, reverse, count occurrences). Each composite skill is explicitly defined as a composition of 2–4 atomic sub-skills (Table 6).

The dependency structure is explicit. For example, `multi_digit_addition` decomposes into `extract_digits` → `add_two_single_digits` → `detect_carry` → `reverse_digits`, and `solve_system_two_equations` decomposes into `mul_two_single_digit_number` → `subtract_two_single_digits` → `divide_single_digits`.

This compositional hierarchy enables our core experiment: we extract one steering vector per atomic skill from 200 training examples, and compose multiple atomic vectors at inference time to steer composite task performance without any composite skill training. This setup tests whether independently learned skill vectors combine to produce emergent compositional capabilities. Table 5 lists the full inventory, and Table 6 provides the dependency mapping. Examples of atomic and composite skills are shown in Figure A.1

Level	#	Skills
atomic	16	`extract_digits`, `reverse_digits`, `add_two_single_digits`, `subtract_two_single_digits`, `mul_two_single_digit_number`, `divide_single_digits`, `add_multiple_numbers`, `compare_values`, `detect_carry`, `detect_borrow`, `modulo_operation`, `get_place_value`, `shift_decimal`, `list_index`, `list_length`, `list_append`
composite	18	`multi_digit_addition`, `multi_digit_subtraction`, `multi_digit_multiplication`, `long_division`, `decimal_addition`, `sum_of_range`, `rounding_numbers`, `percentage_calculation`, `greatest_common_divisor`, `solve_linear_equation`, `solve_system_two_equations`, `check_equation_validity`, `inequality_solving`, `algebraic_simplification`, `range_validation`, `linear_search`, `reverse_list`, `count_occurrences`

Table 5: Skill inventory used in our experiments. atomic skills are atomic operations (single-digit arithmetic, list manipulation, comparisons); composite skills are composite procedures that each depend on 2–4 atomic sub-skills.

Atomic Skill Example: add_multiple_numbers

Instruction: Add all numbers in a list together sequentially. Output type: int. Example: Add all numbers in this list: [1, 2, 3, 4]. Answer: 10

Question	Answer
Add all numbers in this list: [1, 2, 3, 4].	10
Add all numbers in this list: [5, 1].	6
Add all numbers in this list: [0, 9, 9].	18

Composite Skill Example: algebraic_simplification

Instruction: Simplify an algebraic expression by combining like terms (format: coefficient*x + constant). Output type: string. Steps: 1) Identify like terms (same variable and power), 2) Add coefficients of like terms, 3) Combine constants. Your final answer must be a simplified expression (e.g., "21x+13" or "5x-3").

Question	Answer
Simplify 1x+4x	1. Using Skill <compare_values>, identify like terms: x-terms=['1x', '4x'], constants=[]. 2. Using Skill <add_multiple_numbers>, combine x-terms: 1+4=5. 3. So the answer is 5x.

A.2 LLM-Based Data Generation.

To construct diverse, verified problem-solution datasets for each approach pair (e.g., DFS vs. BFS), we design a multi-stage generation pipeline driven by large language models, as shown in Figure 3. First, **GPT-4o** (temperature 0.9) generates novel coding problems in batches, guided by pair-specific domain descriptions, function signature templates, and two diversity mechanisms: a pool of 20 *thematic contexts* (e.g., grid traversal, tree search) assigned round-robin across batches, and a pool of 20 *output framings* (e.g., count components, return reachability) enforced to be unique within each batch. We generate $3.5\times$ the target count to absorb downstream attrition. The raw problems are then semantically deduplicated using cosine similarity over `text-embedding-3-small` embeddings with a threshold of 0.85. Next, a **solution generator** (GPT-4o, temperature 0.4) produces two solutions per problem—one for each approach—using a self-test and repair loop: each candidate solution is executed against the problem's test cases in a sandboxed subprocess, and if any test fails, the error trace is fed back to the LLM for up to two repair attempts. The resulting solution pairs undergo a **three-stage verification**: (i) *execution verification*, requiring both solutions to run successfully on at least two test inputs; (ii) *cross-check agreement*, requiring identical outputs on all mutually successful inputs; and (iii) *approach authenticity verification*, where **GPT-4o-mini** (temperature 0.2) inspects each solution's code structure and confirms the claimed algorithmic approach with confidence ≥ 0.7. An adaptive top-up loop re-generates problems for up to five rounds to guarantee the target count (default 200) is met. The accepted items are formatted as contrastive triplets (x, y, y_c)—where x is the problem, y is the approach to steer away from, and y_c is the approach to steer toward—and split into train/test sets (67%/33%) for downstream steering vector extraction and evaluation.

SKILLSET-CODE Dataset The SKILLSET-CODE dataset is designed to evaluate two capabilities: **(RQ3)** (P) situational personalization, where the model selects among algorithmic approaches based on context (e.g., memory constraints, input scale, readability preference), and **(RQ4)** (E) skill evolution, where a model's default approach preference is shifted along a contrastive trajectory via Reference-free Preference Steering (RePS). Both research questions share the same underlying data but consume it in different formats.

Approach Pairs and Steering Vectors. We define 9 contrastive approach pairs (Table 7), each representing two algorithmically valid strategies for a common problem domain (e.g., DFS vs. BFS for graph traversal, recursion with memoization vs. bottom-up DP for optimization). For each pair, a steering vector is computed as $v = \text{mean}(h^+) - \text{mean}(h^-)$ over hidden-state activations from 200 positive-approach and 200 negative-approach solutions, enabling fine-grained control over which strategy the model favors.
The final acceptance rate is approximately 30%, yielding 200 verified pairs per approach pair at a total API cost of ~$63.

Data Formats. The accepted items are exported in two formats corresponding to the two research questions:

- **Approach groups** (for RQ3 (P), personalization). Each pair produces `positive/train.jsonl` and `negative/train.jsonl`, where

Table 6: Composite skill dependencies. Each composite skill is composed of 2–4 atomic sub-skills applied in sequence.

Composite Skill	Atomic skill Dependencies
multi_digit_addition	extract_digits, add_two_single_digits, detect_carry, reverse_digits
multi_digit_subtraction	extract_digits, subtract_two_single_digits, detect_borrow, reverse_digits
multi_digit_multiplication	extract_digits, mul_two_single_digit_number, add_multiple_numbers, shift_decimal
long_division	extract_digits, mul_two_single_digit_number, subtract_two_single_digits, compare_values
decimal_addition	extract_digits, add_two_single_digits, detect_carry, get_place_value
sum_of_range	add_multiple_numbers, compare_values, subtract_two_single_digits
rounding_numbers	extract_digits, get_place_value, compare_values
percentage_calculation	mul_two_single_digit_number, divide_single_digits, shift_decimal
greatest_common_divisor	modulo_operation, compare_values
solve_linear_equation	subtract_two_single_digits, divide_single_digits, compare_values
solve_system_two_equations	mul_two_single_digit_number, subtract_two_single_digits, divide_single_digits
check_equation_validity	mul_two_single_digit_number, add_multiple_numbers, compare_values
inequality_solving	subtract_two_single_digits, divide_single_digits, compare_values
algebraic_simplification	add_multiple_numbers, mul_two_single_digit_number, compare_values
range_validation	compare_values
linear_search	list_index, compare_values, list_length
reverse_list	list_index, list_length, list_append
count_occurrences	list_index, compare_values, list_length

every entry is formatted as "Problem: . . . \n\nSolution (Approach):\n<code>". Steering vectors are extracted from the hidden-state contrast between positive and negative groups. At inference time, a *situational prompt* determines the steering direction: e.g., a memory-constrained environment steers toward DFS, while a large-grid setting steers toward BFS (Table 8).

- **Contrastive triplets** (for RQ4 (E), skill evolution). Each problem yields a triplet (x, y, y_c) where x is the problem statement, y is the solution using the approach to steer *away* from, and y_c is the solution using the approach to steer *toward*. Triplets are split 67%/33% into train/test sets with no overlapping problems. These triplets define the contrastive trajectory for RePS training, enabling the model's default approach preference to be progressively shifted.

B Experiments Details

B.1 Steerability

Prompt As for prompt during the inference, we require LLM to write the answer in \boxed{} to help proper evaluation. In MATH500, we follow the prompt in EasySteer (Xu et al., 2025).

MATH500 inference prompt

Please reason step by step, and put your final answer within \boxed{}.
User: Convert the point $(0, 3)$ in rectangular coordinates to polar coordinates. Enter your answer in the form (r, θ), where $r > 0$ and $0 \leq \theta < 2\pi$.
Assistant: <think>

SkillSet inference prompt

Add all numbers in a list together sequentially. Output type: int. Example: Add all numbers in this list: [1, 2, 3, 4]. Answer: 10
Problem: Add all numbers in this list: [65, 3, 64, 22, 38, 85]
Please solve this problem. Put your final answer within \boxed{}.

B.2 Compositionality

Text-based composition prompts (Txt-C and Txt-CoT). We use two text-based composition baselines for RQ2 (Section 4.3): **Txt-C** (text-concise) augments the composite skill prompt with a concise description of the procedural decomposi-

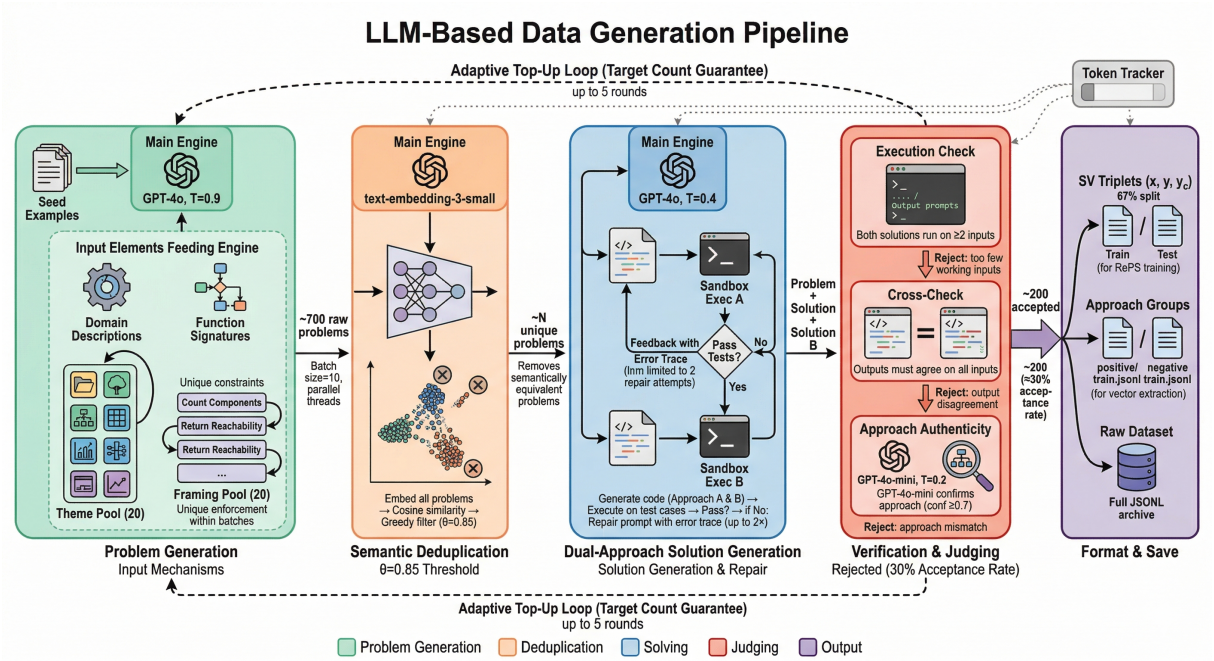

Figure 3: LLM-Based Data Generation

Table 7: The 9 contrastive approach pairs in SKILLSET-CODE.

#	Pair	Approach A	Approach B
1	`dfs_vs_bfs`	DFS	BFS
2	`recursion_vs_dp`	Recursion + Memo	Bottom-Up DP
3	`brute_force_vs_optimized`	Brute Force $O(n^2)$	Optimized $O(n)$
4	`stack_vs_recursion`	Explicit Stack	Recursive Descent
5	`bfs_vs_dijkstra`	BFS (unweighted)	Dijkstra (weighted)
6	`dfs_vs_union_find`	DFS traversal	Union-Find
7	`bfs_vs_bidirectional_bfs`	Standard BFS	Bidirectional BFS
8	`quadratic_dp_vs_optimized`	$O(n^2)$ DP	$O(n \log n)$ Binary Search
9	`index_scan_vs_two_pointer`	Index Scan	Two-Pointer

tion; **Txt-CoT** (text-CoT) augments it with a longer chain-of-thought-style decomposition. Worked examples for both are shown below.

Txt-C (text-concise)

```
Simplify an algebraic expression by
    combining like terms (format:
    coefficient*x + constant). Output type:
     string. Steps: 1) Identify like terms
    (same variable and power), 2) Add
    coefficients of like terms, 3) Combine
    constants. Your final answer must be a
    simplified expression (e.g., ``21x+13''
     or ``5x-3'').

Question: Simplify 1x+4x.

Answer: 5x
```

Txt-CoT (text-CoT)

```
Simplify an algebraic expression by
    combining like terms (format:
    coefficient*x + constant). Output type:
     string. Steps: 1) Identify like terms
    (same variable and power), 2) Add
    coefficients of like terms, 3) Combine
    constants. Your final answer must be a
    simplified expression (e.g., ``21x+13''
     or ``5x-3'').

Question: Simplify 1x+4x.

Answer: 1. Using Skill \texttt{<compare\
    _values>}, identify like terms: x-terms
    =[`1x', `4x'], constants=[]. 2. Using
    Skill \texttt{<add\_multiple\_numbers>},
     combine x-terms: 1+4=5. 3. So the
    answer is 5x.
```

Table 8: Overview of the SKILLSET-CODE evaluation dataset. Each task presents two alternative solution methods and two situational contexts that each favor one method.

Category	Task	Method A	Method B
Graph Search (6 tasks)	Number of Islands	DFS	BFS
	Connected Components	DFS	Union-Find
	Shortest Path (Grid)	BFS	Dijkstra
	Word Ladder	BFS	Bidirectional BFS
	Course Schedule	DFS	BFS (Kahn's)
	Bipartite Check	DFS	BFS
DP & Recursion (6 tasks)	Climbing Stairs	Memoization	Iterative DP
	Coin Change	Memoization	Bottom-Up DP
	Decode Ways	Memoization	Bottom-Up DP
	Longest Increasing Subseq.	$O(n^2)$ DP	$O(n \log n)$ Binary Search
	Edit Distance	Memoization	Bottom-Up DP
	Partition Equal Subset	Memoization	0/1 Knapsack DP
DS & String (6 tasks)	Valid Parentheses	Stack	Counter
	Decode String	Stack	Recursion
	Evaluate Expression	Stack Parsing	Recursive Descent
	Run-Length Encoding	Index Scan	Two-Pointer
	Basic Calculator	Stack	Recursion
	Next Greater Element	Brute Force	Monotonic Stack

B.3 Personalization Experimental Details

Dataset. SKILLSET-CODE comprises 18 LeetCode-style coding problems, each paired with two situational descriptions that favor different algorithmic approaches. Each problem contributes 4 evaluation records, giving 72 records in total. Training data for vector extraction consists of 200 model-generated solutions per approach.

Steering Vector Extraction. We compute a mean-difference vector for each approach pair $(\mathcal{A}^+, \mathcal{A}^-)$ as:

$$v_\ell = \frac{1}{|\mathcal{A}^+|} \sum_{x \in \mathcal{A}^+} h_\ell(x) \; - \; \frac{1}{|\mathcal{A}^-|} \sum_{x \in \mathcal{A}^-} h_\ell(x), \tag{12}$$

where $h_\ell(x)$ denotes the residual-stream hidden state at layer ℓ for input x.

Inference-Time Steering. At inference, the steering direction is injected on the last 50% of layers at the last prefill token and at every generated token:

$$h_\ell \; \leftarrow \; h_\ell + \alpha \, \hat{v}_\ell, \tag{13}$$

where $\hat{v}_\ell$ is ℓ_2-normalised. The activation strength reported in Section 4.4 is $\alpha = 1.0$ (see Section D for the full hyperparameter table and α disclosure). Generation uses greedy decoding (temperature $=$ 0) with a maximum of 4096 tokens.

Evaluation. All generated solutions are scored by GPT-4o-mini acting as an LLM judge. The judge is prompted with the problem statement, the target approach name, and the generated code; it returns a binary verdict on whether the implementation follows the intended approach. We report *approach match rate*—the fraction of records where the generated code matches the prescribed approach—separately for the baseline (no steering) and steered conditions. As emphasized in Section 4.4, this metric measures procedural *controllability*, not functional correctness.

Prompt of LLM-as-a-Judge for Personalization

System: You are an expert code reviewer. Your task is to determine which algorithmic approach a Python solution uses. Analyze the code structure and logic, not just variable names or comments. Output valid JSON only.

User: Given the following coding problem and a generated solution, determine which of the two approaches the solution uses.
`## Problem`
{problem}
`## Approach A:` *{approach_a}*
`## Approach B:` *{approach_b}*
`## Generated Solution`
```python
{code}
```
Analyze the code's actual algorithmic structure (data structures used, traversal order, control flow) and determine which approach it follows.
Output JSON:
```json
{
"approach": "A" or "B" or "neither",
"confidence": 0.0-1.0,
"reasoning": "one sentence explaining why"
}
```

Prompt of LLM-as-a-Judge for Evolution

System: You are a precise code evaluation judge. Your task is to read a model output and determine which programming approach it uses or describes.

Rules:

- Answer with exactly one word: `positive`, `negative`, or `none`.
- `positive` means the output uses or clearly describes the **first** (target) approach.
- `negative` means the output uses or clearly describes the **second** approach.
- If the output is ambiguous, uses neither approach, or contains only a problem description without a clear algorithmic choice, answer `none`.
- Base your judgment on actual code, algorithmic choice, or explicit approach description in the text.

User:
`Pair:` {*pair_key*}
`POSITIVE approach:` {*positive_description*}
`NEGATIVE approach:` {*negative_description*}
Model output to judge:

{*model_output*} *(truncated to 1,500 characters)*

Does this output use the POSITIVE or NEGATIVE approach?
Answer: `positive` / `negative` / `none`

B.4 Evolution Details

Evolution Evaluation Prompt Below is the LLM-as-a-Judge prompt to evaluate the approach match rate.

Approach Match Rate Over Training Figure 4 shows the in-distribution approach match rate over training steps for RePS-optimized steering directions on Qwen3-0.6B (left) and Qwen3-1.7B (right). Each curve corresponds to an approach pair, with markers indicating peak values. Performance is consistently non-monotonic, with many pairs reaching their maximum at early or intermediate steps before declining, indicating that the optimal direction emerges before convergence.

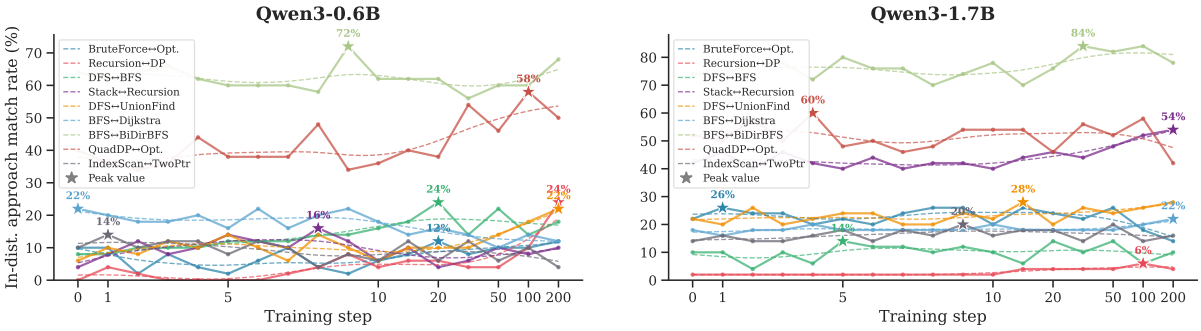

Figure 4: In-Distribution Approach Match Rate Over Training

C Skill-Direction Extraction Procedures

We use two complementary extraction schemes, each chosen to match the structure of the corresponding sub-study.

Procedural-execution axis (motivating the single-class extraction). We model skill-conditioned activations as concentrating near a low-dimensional manifold along a procedural-execution axis. Concretely, for a fixed skill s and a layer ℓ, we assume the hidden state on a skill-s example x_i admits the decomposition

$$h_\ell^{(s)}(x_i) = \mu_\ell^{(s)} + a_i u_\ell^{(s)} + \epsilon_i, \qquad (14)$$

where $\mu_\ell^{(s)}$ is the skill-conditioned mean, $u_\ell^{(s)} \in \mathbb{R}^{d_\ell}$ is the procedural-execution direction (the axis along which skill-s execution varies), $a_i \in \mathbb{R}$ is a per-example scalar reflecting how strongly that procedure is engaged on x_i, and ϵ_i is an instance-specific residual. Under this decomposition, the leading principal component of the centered skill-conditioned activations $\{h_\ell^{(s)}(x_i) - \mu_\ell^{(s)}\}_i$ recovers the procedural-execution direction $u_\ell^{(s)}$ up to a sign. This view is what motivates the single-class PCA extraction below: in many of our math skills (e.g., `add_two_single_digits`) there is no natural negative class to form a contrastive direction, so the procedural-execution axis is the most direct target for single-class extraction.

Single-class PCA (RQ1/2, SkillSet-Math). For each math skill s (atomic or composite), we collect up to 500 training examples per skill and record the hidden state at every layer, taken at the *last token* position of the formatted input. Each formatted input consists of the skill instruction, the problem, and the corresponding answer text. For atomic skills the answer text is the final numeric answer; for composite skills the answer text is the full step-by-step derivation (i.e., a longer answer that exposes the procedural composition). We center these per-skill activations and compute the leading principal component (top-1 PCA), giving one ℓ_2-normalized direction $v_\ell^{(s)}$ per layer. This single-class, positive-only extraction targets the procedural-execution direction $u_\ell^{(s)}$ in Equation (14): for many math skills there is no natural negative class with which to construct a contrastive direction (e.g., the negation of `add_two_single_digits` is not a single, well-defined alternative procedure), so the procedural-execution axis is the most direct target. We disclose two methodological details that read this design more carefully. First, because the prompt used for extraction includes the correct answer, the extracted direction reflects the hidden state of the model conditioned on observing problem-and-answer together, not the hidden state of the model when it is generating the answer on its own. Second, the principal-component sign is indeterminate; we use the raw PCA direction without sign correction, and the sign of α in evaluation is fixed across skills.

Contrastive mean-difference (RQ3, SkillSet-Code). For each algorithmic-strategy pair (e.g., DFS vs. BFS), we form 200 positive (target-strategy) and 200 negative (alternative-strategy) solutions per pair. We record the hidden state at the last token of each formatted input (`Problem: ...\n\nSolution:\n<code>`) at every layer, and compute the mean-difference direction $v_\ell = \mu_\ell^+ - \mu_\ell^-$, which we then ℓ_2-normalize. Because the pair is itself contrastive, this construction yields a sign-determinate direction (pos $\rightarrow$ neg) and a natural negative class that the single-

class PCA extraction lacks.

Shared inference-time choices. Across both extractions, we use *per-layer* directions, intervene on the last 50% of layers (e.g., layers 14–27 for Qwen3-0.6B's 28-layer stack), and add the direction to hidden states at the last prefill token and at every generated token via $\tilde{h}_\ell \leftarrow h_\ell + \alpha v_\ell$. We report the activation strength α used in each sub-study explicitly: RQ1/2 use $\alpha = 2.0$; RQ3 evaluations in this paper report results at $\alpha = 1.0$, which is the value used in the tracker runs underlying the reported numbers (the configured sweep grid was $\{0, 0.5, 1, 2, 4\}$). Full hyperparameter details (token positions, sample counts, prompt formats, target-layer scope, α grids) are provided in Section D below.

Why two schemes. The use of two extraction schemes is principled rather than ad hoc: single-class PCA matches the structure of atomic/composite math skills (one positive class, no natural negative), while mean-difference matches the structure of paired algorithmic strategies (two structured classes whose contrast *is* the skill of interest). This pairing of extraction scheme to the structural form of the skill is essential to our claims; in particular, in RQ1/2 we do not claim PCA is the only or best extractor, only that it provides a single, consistent procedure with which to study the four observable properties on SKILLSET-MATH.

D Skill-Direction Extraction Hyperparameters

This section provides the full hyperparameter specification for the two extraction schemes described in Section C. We describe these in methodological terms (token positions, sample counts, prompt formats, target-layer scope, α values).

RQ1/RQ2 setting (single-class PCA on SKILLSET-MATH). Table 9 summarizes the extraction settings used for steerability and composition.

RQ3 setting (contrastive mean-difference on SKILLSET-CODE). Table 10 summarizes the extraction settings used for personalization.

Shared inference-time choices. Both extractions use per-layer directions, intervene on the last 50% of layers, and trigger at the last prefill token and at every generated token. Generation uses greedy decoding (temperature $= 0$) with a maximum of 4096 tokens.

Disclosure: RQ1/2 prompts include the correct answer. For RQ1 and RQ2, the formatted input from which PCA directions are extracted includes the instruction, the problem, and the correct answer (final answer for atomic skill; full step-by-step derivation for composite skill). The extracted direction therefore reflects the model's hidden state conditioned on observing problem-and-answer together, not its hidden state when generating the answer autonomously. We make this design choice explicit because it materially affects how the extracted direction should be interpreted: the direction targets the procedural-execution axis present when the model is conditioned on a worked example, and is then re-injected on unseen problems at inference time.

E Cross-Family Results on Llama-2-7B-chat

All experiments in the main paper use the Qwen3 family, which shares a single architecture and training recipe. To test whether the reported properties depend on that choice, we repeat part of the controlled study on Llama-2-7B-chat, a 2023-era model with a different tokenizer, training corpus, and instruction-tuning procedure. We cover skill composition on all 18 composite skills (Table 11), transfer to GSM8K and MATH500 (Table 13), and a sweep over intervention strength on five representative skills (Table 12). Our goal is not a new generic steering method, but a controlled framework for latent procedural skills; these results extend the evidence beyond Qwen3.

Skill Composition. The central composition result of Section 4.3 reproduces across model families. Vector composition outperforms both text-based composition baselines on 13 of 18 composite skills, the same ratio observed on Qwen3-0.6B, and attains the highest average accuracy (31.5, versus 21.6 for Txt-C and 20.7 for Txt-CoT). Because Llama-2-7B-chat differs from Qwen3 in tokenizer, training corpus, and instruction-tuning procedure, this provides direct evidence that the composition finding is not limited to the Qwen3 model family. Absolute accuracies are substantially lower than on Qwen3, consistent with the weaker base model.

Transfer to Real Benchmarks. Skill directions extracted on SKILLSET-MATH also transfer pos-

Item	Setting
Extraction method	Single-class PCA (positive samples only; non-contrastive)
Principal component	First principal component (top-1)
Layer scope	Per-layer (independently extracted at every layer)
Token position	Last token of the formatted input
Sample size	Up to 500 examples per skill (capped at available count)
Prompt format	Instruction + Problem + Answer
Answer text (atomic)	Final answer
Answer text (composite)	Full step-by-step derivation
Normalization	ℓ_2 normalization of the direction
Skill inventory	SKILLSET-MATH (16 atomic + 18 composite = 34 skills)
Evaluation datasets	SKILLSET-MATH; transfer to GSM8K and MATH500
Intervention	Additive ($\tilde{h}_\ell = h_\ell + \alpha v_\ell$)
Activation strength	$\alpha = 2.0$
Target layers	Last 50% of layers (e.g., layers 14–27 on Qwen3-0.6B)
Trigger tokens	Last prefill token + every generated token

Table 9: RQ1/RQ2 extraction setting: single-class PCA on SKILLSET-MATH.

Item	Setting
Extraction method	Mean-difference contrastive: $v = \mu^+ - \mu^-$
Layer scope	Per-layer (independently extracted at every layer)
Token position	Last token of the formatted input
Normalization	ℓ_2 normalization of the direction
Sample size	200 positive + 200 negative per approach pair
Prompt format	"Problem: ...\n\nSolution (Approach):\n<*code*>"
Skill inventory	9 algorithmic-strategy pairs (e.g., DFS vs. BFS)
Evaluation set	72 evaluation records
Intervention	Additive ($\tilde{h}_\ell = h_\ell + \alpha v_\ell$)
Target layers	Last 50% of layers
Trigger tokens	Last prefill token + every generated token

Table 10: RQ3 extraction setting: contrastive mean-difference on SKILLSET-CODE.

Composite skill	Base	Txt-C	Txt-CoT	Vec
multi_digit_addition	84	7	24	**85**
multi_digit_subtraction	**74**	11	21	71
multi_digit_multiplication	**11**	1	1	**11**
long_division	58	19	18	**65**
decimal_addition	**78**	65	36	77
sum_of_range	5	1	2	**7**
rounding_numbers	29	37	**40**	31
percentage_calculation	29	28	18	**32**
greatest_common_divisor	13	**23**	16	11
solve_linear_equation	90	54	49	**93**
solve_system_two_equations	0	0	0	**1**
check_equation_validity	0	**55**	40	0
inequality_solving	**1**	0	0	**1**
algebraic_simplification	25	8	19	**26**
range_validation	14	39	**43**	10
linear_search	6	**29**	21	5
reverse_list	**11**	2	2	10
count_occurrences	27	9	23	**31**
Average	30.8	21.6	20.7	**31.5**

Table 11: Skill composition on Llama-2-7B-chat. Accuracy (%) on the 18 composite skills of SKILLSET-MATH. Bold marks the best method per skill.

Skill	Base	α=0.5	α=1.0	α=2.0	Best Δ
subtract_two_single_digits	84	88	88	**95**	+11
multi_digit_subtraction	74	78	79	**84**	+10
inequality_solving	1	0	1	**10**	+9
compare_values	51	53	53	**60**	+9
solve_linear_equation	90	91	92	**95**	+5

Table 12: Sensitivity to intervention strength α on Llama-2-7B-chat, for five representative skills under single-class PCA extraction. Accuracy (%).

itively on Llama-2-7B-chat, improving GSM8K by 3.26 and MATH500 by 2.80 points over zero-shot decoding. These gains are considerably larger than those observed on the stronger Qwen3 models in Table 1, which we attribute to a headroom effect: an inference-time procedural intervention can only correct errors the base model actually makes, and Llama-2-7B-chat starts from a far weaker baseline (25.25 on GSM8K, 6.00 on MATH500) than Qwen3-14B (95.83 and 92.40). This pattern is consistent with recent scaling work on activation steering, which reports that steering effects can diminish with model size (Ali et al., 2025).

Benchmark	0-shot	Steered	Δ
GSM8K	25.25	28.51	+3.26
MATH500	6.00	8.80	+2.80

Table 13: Transfer of SKILLSET-MATH skill directions on Llama-2-7B-chat. Accuracy (%).

Sensitivity to Intervention Strength. The activation strength α is an intervention hyperparameter rather than a universal constant, and prior steering work likewise tunes it per setting rather than fixing one global coefficient (Rimsky et al., 2024; Oozeer et al., 2025). This is expected in our setting because the sub-studies use different benchmarks and different vector definitions: single-class PCA on SKILLSET-MATH versus contrastive mean-difference on SKILLSET-CODE. We therefore report a consistent α within each dataset and extraction regime rather than forcing a single value across all sub-studies. Table 12 shows the corresponding sweep on Llama-2-7B-chat: the useful range is skill-dependent, with inequality_solving responding only at α=2.0 while solve_linear_equation varies little across the sweep, and large α can over-steer the model. These results support calibrating intervention strength to the model, representation scale, and target behavior.