SKILLSMITH: LEARNING TO COMPOSE PARAMETRIC SKILLS AND TEXTUAL KNOWLEDGE

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic systems driven by large language models (LLMs) regularly feature two key mechanisms to autonomously solve complex problems: synthesizing text-based knowledge and procedures from past experiences and building parametric (weight-space) skill libraries for recurring sub-goals. To date, research has largely treated these as orthogonal pursuits: either organizing textual knowledge through composition and reflection, or consolidating parametric skills via weight-space merging. Consequently, the seamless integration of text and model weights for targeted performance improvements remains largely unexplored. This work bridges this modality gap by treating model weights as an additional modality that an LLM can natively reason over. We instantiate parametric learning via prefix-tuning and augment an LLM to ingest both prefix weights and rich textual data which capture relationships to a target capability. Our augmented LLM, which we call SkillSmith, synthesizes these inputs to perform instruction-steered parametric synthesis, directly outputting new prefix weights that manifest the target skill. We demonstrate that our approach significantly outperforms both text-only and weight-space-only baselines, unlocking performance gains that are out of reach for uni-modal (text-only or weight-only) adaptations.

1 INTRODUCTION

LLMs (OpenAI, 2023; Anthropic, 2024; Qwen, 2025; Google, 2025b) have evolved from static conversational interfaces into drivers of agentic systems capable of solving problems that require complex multi-step reasoning (Kwa et al., 2026; Huang & Yang, 2025). Central to the efficacy of these systems is their ability to learn and adapt from past experiences. Currently, this adaptation is driven by two powerful, yet largely siloed mechanisms. The first is the synthesis of text-based knowledge, where agents use natural language—such as self-reflection (Madaan et al., 2023; Renze & Guven, 2024), structured memory (Chhikara et al., 2025; Xu et al., 2025a;b), or prompt generation (Wang et al., 2024; Agrawal et al., 2026)—to guide future reasoning and planning. The second involves building parametric skill libraries via parameter-efficient fine-tuning (PEFT) (Shekar & Krishnan, 2026) for consolidating learned behaviors into modular weights which can be retrieved and used independently or merged with other skills to efficiently address recurring sub-goals (Huang et al., 2023; Pfeiffer et al., 2020).

To date, the agentic research community (Sun et al., 2025; Zhang et al., 2025; Zhou et al., 2025) has largely treated text-based reasoning and parametric skill acquisition as orthogonal pursuits. We believe this separation limits the potential of agentic systems. Empowering agents to seamlessly reason over both weight-space and text could unlock compositional generalization (Dziri et al., 2023; Keysers et al., 2019; Lake & Baroni, 2018) across parametric skills, with the added advantage that the axes of composition and generalization can be flexibly steered via instruction text. As a concrete example, consider an agent that has developed a set of models (parametric skills) for solving various subtasks, alongside a rich, text-based strategy for deploying them. Suppose that through past user interactions, this agent trained a prefix-cache (Li & Liang, 2021) for translating English to Twi in one session, and compiled comprehensive notes on analyzing English legal documents in another. When encountering a novel task, e.g., *analyzing a legal document written in Twi*, the agent reasons that the solution requires a combination of its prior skills: *English-to-Twi translation*, *Twi language modeling* and *legal document analysis*. However, while the agent can articulate this transfer textually, there is currently no mechanism for it to leverage this reasoning along with the weight-space *translation skill* and *Twi language skill* to directly synthesize the corresponding task weights for the new problem.

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

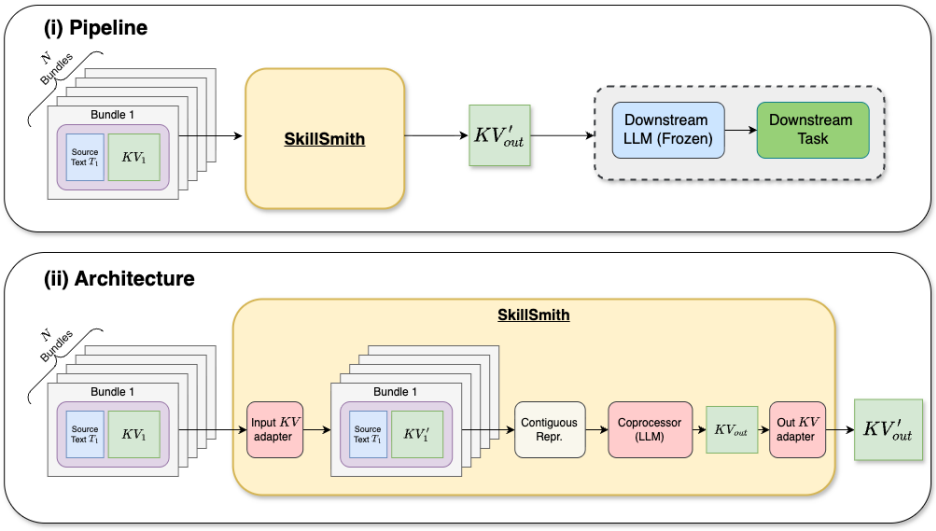

Figure 1: SkillSmith pipeline and architecture. Source task bundles consisting of text and prefix-cache weights are fed as inputs to an augmented LLM to directly generate the prefix-cache that will be used to modulate a downstream LLM to the target task.

This work takes initial steps toward the frictionless composition of text- and weight-based agentic artifacts. We propose to treat weight-space inputs as simply an additional modality that can be natively processed by an appropriately augmented pre-trained LLM. This augmented LLM – which we term **SkillSmith** – is trained to simultaneously reason over both text and weight-space inputs, enabling it to directly generate task-specific parametric skills for downstream deployment (Figure 1). Specifically, we first instantiate parametric skill learning as prefix-tuning (Li & Liang, 2021). We then train SkillSmith to ingest (i) prefix-weights along with rich text metadata about the tasks solved by the weights and (ii) extra text meta-data that captures the functional relationships between the weights-space skills and a desired target capability. The target capability is specified via a high level text description or few shot exemplars and the output prefix-weights from SkillSmith are directly optimized on the target task in an end-to-end fashion as in Liu et al. (2024).

To demonstrate that SkillSmith flexibly composes text- and weight-based agentic artifacts into target skills, we make the following contributions:

- To study mixed-modality composition at scale, we need datasets where ground-truth relationships between tasks are known. To this end, we contribute a data-generation workflow that we use to produce Composite SNI, a synthetic compositional generalization dataset. By prompting Gemini 2.5 (Google, 2025a) to generate tasks that require the combined skill sets of two input tasks from the Super Natural Instructions (SNI) dataset (Wang et al., 2022), we create a setting where the ground-truth input tasks for a particular target capability are known. We will show that we can effectively bootstrap this synthetic data to jump-start learning text-and-weight composition in settings with a limited number of tasks.
- Using the 4B Gemma 3 model (Google, 2025b) for both SkillSmith and downstream task-solving, we show that SkillSmith outperforms standard zero-shot, weight-space composition baselines on three datasets: Composite SNI, SNI and MMLU-ProX (Xuan et al., 2025). Furthermore, we demonstrate that SkillSmith serves as a highly superior parameter initialization; fine-tuning the prefix-weights generated by our SkillSmith on the target task yields representations that either outperform or are competitive with all other methods studied, including the direct training of a prefix-cache initialized from in-context examples.
- Finally, we introduce a retriever-based approach for settings where the ground-truth mapping of a set of input tasks to target task capability is unknown. We show that even in this noisy setting, SkillSmith is able to extract and compose the relevant signal (if it exists) from the candidate source tasks and leverage it to generate prefix-weights that outperform baselines under equalized conditions.

Ultimately, SkillSmith demonstrates that language models can natively reason over (their own) modular weights just as they do with text, thus providing a blueprint for more holistic agentic architectures. Our results underscore that treating parameter-space as a readable, synthesizable modality provides a distinctly superior initialization for targeted adaptation, facilitating the scalable and effective composition of agentic skills.

2 Related Work

Text-Based Agentic Adaptation. Modern LLM-driven agents rely heavily on past experiences to solve complex problems. Currently, this adaptation has been almost exclusively driven by synthesizing text-based knowledge. Agents leverage natural language to guide future planning through mechanisms such as self-reflection (Madaan et al., 2023; Shinn et al., 2023), structured memory (Chhikara et al., 2025; Xu et al., 2025b), and prompt optimization (Agrawal et al., 2026; Wang et al., 2024).

Parametric Skill Acquisition and Weight Merging. Conversely, Parameter-Efficient Fine-Tuning (PEFT) adapts language models to downstream tasks by consolidating learned behaviors into efficient, modular weights (Han et al., 2024) via adapters (Houlsby et al., 2019; He et al., 2022; Lei et al., 2023), LoRA (Hu et al., 2021), or prompt or prefix tuning (Lester et al., 2021; Li & Liang, 2021). Prior frameworks transfer these behaviors via shallow arithmetic weight-merging operations, such as averaging or routing, which fail to leverage semantic task metadata (Vu et al., 2022; Asai et al., 2022; Pfeiffer et al., 2020; Huang et al., 2023). To bridge this gap and achieve true compositional generalization (Dziri et al., 2023; Keysers et al., 2019; Lake & Baroni, 2018), we employ an LLM as a hyper-network (Shenaj et al., 2025). Unlike existing hyper-networks, SkillSmith natively processes parameters, specifically prefix KV caches, alongside textual metadata to translate an agent's reasoning into instruction-steered parametric synthesis.

3 Bridging Text and Parameter Modalities with SkillSmith

We study settings where an agent, as part of its learning procedure, is able to build up both text and weight-space artifacts to solve tasks. We assume the agent encounters tasks that are sufficiently related such that new tasks can be solved by synthesizing relevant information from past experiences.

Let $\mathcal{T}_{src}$ be the set of all tasks that an agent has encountered before. We associate each task $T_i \in \mathcal{T}_{src}$ with a task bundle $b_i = (m_i, w_i)$ where m_i is a learned PEFT module and w_i represents task-relevant textual meta-data, such as in-context learning (ICL) demonstrations or task reflections (Shinn et al., 2023). Though the module, m_i, can conceptually represent any PEFT method, this work focuses specifically on prefix-tuning. Therefore, each m_i is a Key-Value (K-V) prefix cache associated with a pre-specified, frozen base model M_ϕ. The choice of prefix-tuning is motivated by the fact that text snippets can naturally be translated into K-V caches via forward passes through M_ϕ. This provides *a priori* confidence that it is feasible to bridge the modality gap by learning relationships between parametric K-V caches trained from scratch and those derived directly from text.

3.1 Problem Statement

Consider an agent presented with a new task, T_{new}, to solve. Our goal is to produce a new PEFT module m_{new} which can solve instances of T_{new} that the agent may encounter. A baseline approach would be to train a completely new PEFT module using only data from T_{new}; however, this fails to leverage potentially useful information stored within the agent's historical experiences, $\mathcal{T}_{src}$. Hence, we assume that the agent can access a subset of the previously encountered source tasks, $\mathcal{T}_{src}[T_{new}] = \{T_1, \ldots, T_N\}$, deemed relevant to constructing a solution to T_{new}. Uni-modal strategies exist to leverage $\mathcal{T}_{\text{src}}[T_{new}]$, but their siloed nature presents limitations:

Text-only strategies: One can aggregate all available text metadata, i.e., ICL examples and reflections from the source tasks and combine them directly with those of T_{new} to serve as in-context data for solving T_{new}. While highly flexible, this approach is computationally expensive and constrained by inference-time context limits.

Weight-space strategies: One can initialize from the relevant trained modules $\{m_i\}$ associated with the tasks in $\mathcal{T}_{src}[T_{new}]$ using standard weight-space merging techniques (e.g., averaging) with the

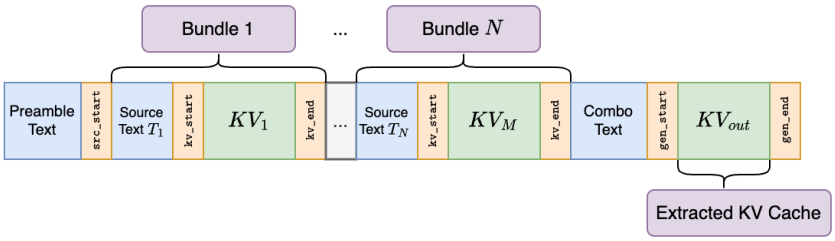

Figure 2: Contiguous representation of task bundles to be processed by the co-processor.

option of subsequently tuning the merged weights on data from T_{new}. While inference-efficient, arithmetic merging ignores the rich relationships in text metadata.

We contend that these isolated approaches do not maximally reuse the computation spent to construct $\mathcal{T}_{src}$ and they miss out on performance gains that could be achieved by synthesizing information across interleaved modalities. We are thus motivated to answer:

How can we bridge these modalities, effectively combining both text- and weight-space artifacts from $\mathcal{T}_{src}[T_{new}]$, to construct an m_{new} that definitively outperforms purely uni-modal baselines?

3.2 SkillSmith Architecture

Given the limitations of unimodal approaches, we introduce SkillSmith (Figure 1), an augmented pre-trained transformer language model capable of synthesizing text- and weight-space prefix-cache artifacts. To solve a new task T_{new}, SkillSmith composes existing related source task bundles $\mathcal{T}_{src}[T_{new}]$ alongside descriptive textual meta-data to generate m_{new}.

Specifically, given a set of N task bundles $\{b_i\}_{i=1}^N$ as input, SkillSmith first projects the weight-space modules $\{m_i\}_{i=1}^N$ into the latent space of the language model via an Input K-V Adapter—parameterized as a multi-layer perceptron (MLP)—to obtain adapted caches $\{KV_i'\}_{i=1}^N$. We then construct an input sequence for the SkillSmith coprocessor (its pre-trained LLM backbone) by interleaving the textual metadata $\{w_i\}_{i=1}^N$ and adapted K-V caches using a specialized vocabulary of control tokens (Figure 2): (1) **Preamble Text**, which primes the model on the overarching compositional objective; (2) **Interleaved Bundles**, which sequentially pair each source task's textual metadata with its adapted module KV_i', demarcated by `<src_start>`, `<kv_start>`, and `<kv_end>` tokens; (3) **Combination Text**, which provides auxiliary relational information across source tasks or in-context exemplars of the target task; and (4) a **Generation Block**, which appends placeholder tokens $(z_1, \ldots, z_L)$ bounded by `<gen_start>` and `<gen_end>` to earmark the target positions for K-V cache synthesis.

The entire constructed block is forwarded through SkillSmith's coprocessor. To extract the synthesized parametric skill, we isolate the KV-caches corresponding to the placeholder latent tokens $(z_1, \ldots, z_L)$, yielding the raw output weights KV_{out}. After applying transformations to strip positional information (inverse RoPE de-rotation), KV_{out} is passed through Out K-V Adapter (MLP) to synthesize weights m_{new}, which can be deployed directly to a frozen downstream base model M_ϕ to solve T_{new}.

3.3 End-to-End Meta-Training of SkillSmith

We take an end-to-end approach to train SkillSmith by bootstrapping the pre-constructed library $\mathcal{T}_{src}$ into a meta-training dataset $\mathcal{D}^{\text{train}}$. For each task $T \in \mathcal{T}_{src}$, $\mathcal{D}^{\text{train}}$ contains a structured tuple comprising: (i) the task's original text examples $T\{\mathbf{x}, \mathbf{y}\}$ (inputs $\mathbf{x}$ and targets $\mathbf{y}$); (ii) $\mathcal{T}_{src}[T]$, the ground-truth or retrieved source tasks (and their corresponding task bundles) mapped to T; and (iii) the **Combination Text** w, which captures metadata about T and its relationship to the source tasks. A concrete example of an entry in $\mathcal{D}^{\text{train}}$ is provided in Appendix K.1.

Having constructed $\mathcal{D}^{\text{train}}$, we learn the parameters of SkillSmith, denoted by θ, by minimizing the following objective:

$$\theta^* = \text{argmin}_\theta \sum_{\left(T\{\boldsymbol{x},\boldsymbol{y}\},\, \mathcal{T}_{src}\left[T\right],\, w\right) \sim \mathcal{D}^{\text{train}}} \mathcal{L}\left(M_\phi(\mathbf{x}; m_T), \mathbf{y}\right)$$

where $m_T = \text{SkillSmith}_\theta(\{b_k\}_{k \in \mathcal{T}_{src}[T]}, w)$ is the generated KV-cache. Input to SkillSmith $(\cdot)$ is constructed as discussed in Section 3.2. $\mathcal{L}$ depends on the task and setting, e.g. regular cross-entropy (CE) loss or a policy loss for reinforcement learning. For the remainder of this work, we take $\mathcal{L}$ as the CE loss. Note that though we back-propagate through M_ϕ to get the gradients for θ, ϕ is kept fixed.

4 Experiments

4.1 Dataset Preparation

We first build our library of source tasks $\mathcal{T}_{src} = \{T_1, \ldots, T_N\}$. For the remainder of this work, we assume $N = 2$. Given the dataset $\mathcal{D}(T_i) = \{\{\mathbf{x}, \mathbf{y}\}, w\}$, we train a prefix K-V cache with $\mathbf{x}$ as the input text and $\mathbf{y}$ as the target over which we compute a negative log-likelihood (NLL). To inject diversity into our library and discourage overfitting, the input prefix-cache sequence length for each source task is randomly sampled. We optimise these modules over a hyperparameter grid, selecting the setting that minimises validation NLL to form the final task module m_i (See Appendix F).

With $\mathcal{T}_{src}$ prepared, we construct the training dataset for SkillSmith following the procedure outlined in 3.3. If ground-truth task mappings are available (as with Composite-SNI), $\mathcal{T}_{src}[T_i]$ is directly mapped. In their absence, we employ the heuristic task retrieval pipeline described in 4.4.2 to construct $\mathcal{T}_{src}\left[T_i\right]$. We obtain **Combination Text** by prompting Gemini 2.5 (Appendix J) to describe semantic relationships between $\mathcal{T}_{src}[T_i]$ and T_i.

4.2 SkillSmith Training

We train SkillSmith on the objective described in Section 3 with cross-entropy loss (hyperparameter choices are in Appendix K). As regularization, we dynamically sample the output length of the generated target cache from $\{16, 32, 64, 128\}$ during training. For evaluation, we fix the length of the generated prefix-cache to 32. SkillSmith's pre-trained LLM trainable coprocessor is Gemma 3 4B.

4.3 Evaluation Protocol

For each evaluation task T_{eval}, we construct the target evaluation tuple ($T_{\text{eval}}\{\mathbf{x}, \mathbf{y}\}, \mathcal{T}_{src}\left[T_{\text{eval}}\right]$, **Combination Text**). All evaluated methods have access to this tuple and can leverage the text fields or weights to generate the solution prefix-cache $m_{T_{\text{eval}}}$.

For each task, we measure the model's average negative log-likelihood (NLL) loss of the targets given its inputs on the heldout set. Because evaluation benchmarks vary significantly in scale and numerical boundaries, aggregating raw NLL values directly can bias results toward high-variance tasks. To provide a scale-invariant and balanced global evaluation, we map model performance into an adapted global Elo rating system (Appendix D) where each method is treated as a tournament competitor.

To ensure a fair comparison between SkillSmith and the baselines, we retrain the prefix weights of the source tasks for each T_{eval} such that they have length 32 and set the output cache size of SkillSmith and each baseline method to 32 (except for the Concat baseline which ends up with 64). Performance is reported on the heldout-set of T_{eval}. We explore evaluating $m_{T_{\text{eval}}}$ in both zero-shot and fine-tuning settings. By zero-shot, we mean inference over T_{eval} using the synthesized prefix weights without first directly training via gradient descent on its task samples – thus we consider text-only, in-context methods as ostensibly zero-shot. If fine-tuning, we execute the same grid protocol described in Section 4.1 above, sweeping 4 random configurations and keeping the configuration with the best validation performance.

4.4 Data Environments and Benchmarks

4.4.1 Synthetic Task Composition: Composite-SNI (CSNI) Dataset

In order to effectively study mixed-modality composition, we need a robust validation environment where ground-truth task lineages are known. This allows us to isolate and ablate SkillSmith's synthesis capability without the confounding factor of source task retrieval quality. Using Super-natural Instructions (SNI) (Wang et al., 2022) as a foundation, we created a synthetic dataset, Composite-SNI, by presenting pairs of SNI tasks to Gemini 2.5 Pro.

Specifically, we prompted it to generate a new composite task, $T_{\{1,2\}}$, that naturally leverages a subset of the skills needed to solve T_1 and T_2 independently (see Appendix A). Our final Composite-SNI dataset consists of $\sim$21K composite tasks, which we split into meta-train ($\sim$17K tasks seen by SkillSmith during the training phase) and meta-eval sets. We further stratify the CSNI dataset into three subsets based on whether their constituent source tasks appear in the meta-train set, which would enable us to do additional analysis on the generalisation capabilities of SkillSmith (Appendix H).

4.4.2 SkillSmith in the Wild: Datasets and Heuristic Source Task Retrieval

In real-world deployment scenarios, the ground-truth source tasks required to synthesize a novel target capability are rarely known. Composite-SNI is effective for understanding SkillSmith, we need to investigate our mixed-modality composition approach in realistic scenarios by experimenting with Super-natural Instructions (SNI) (Wang et al., 2022), a dataset consisting of diverse natural language processing tasks, and MMLU-ProX (Xuan et al., 2025), a multi-lingual dataset consisting of multiple subject domains. More details on these datasets and our split setup can be found in Appendix E.

Since these datasets lack ground-truth source-task mappings, we introduce a two-stage heuristic pipeline. First, a **Semantic Retrieval** stage uses a retriever (Appendix B) that leverages Gemini Embeddings (Lee et al., 2025) to rank source tasks by relevance into P candidate pairs. Second, an **LLM Selection** stage (Appendix C) routes these candidate task profiles to Gemini 2.5 Pro, which selects the most relevant pair as $\mathcal{T}_{src}\big[T_{new}\big]$.

4.5 Baselines

We compare SkillSmith against standard approaches categorized into three distinct paradigms (see Appendix G for complete baseline formulations): (i) **Weight-Space Only:** We evaluate standard parameter-space model merging methods, including linear interpolation (**LERP**), prefix cache concatenation (**Concat**), averaged source-task transfer, and singular value decomposition (SVD) applied across sequence, head, or embedding dimensions, (ii) **Text-Space Only:** We evaluate **In-Context Learning (ICL)** using k-shot prompts, sweeping $k \in \{16, 32, 64\}$ to select the configuration minimizing validation negative log-likelihood, and (iii) **Transfer-less Adaptation:** We optimise a fresh target task prefix KV-cache from scratch using **Direct Prefix tuning** under standard initialisation, ICL-token initialisation, and a text augmented setting where all auxiliary text metadata is directly prepended to the input examples before prefix-tuning (Section 6.2).

5 Results

5.1 SkillSmith is effective in Composite-SNI

We analyse the performance of SkillSmith on Composite-SNI, where ground truth source tasks are known. This allows us to perform an analysis without the confound of noisy source task retrieval.

Figure 3 demonstrates the effectiveness of SkillSmith in synthesizing text and prefix-cache information from ground-truth source tasks to solve novel target capabilities. In the zero-shot regime, where we evaluate the synthesized prefix-caches before any direct fine-tuning on the downstream task, SkillSmith solidly outperforms all baseline weight-space merging techniques. We do note that standard in-context learning (ICL) remains highly competitive in the zero-shot regime, indicating that the base model maintains an edge over zero-shot parametric generation without any adaptation.

When we allow for task-specific fine-tuning, however, SkillSmith is the clear winner across all evaluated methods. Fine-tuning the prefix-weights initialised by SkillSmith yields representations

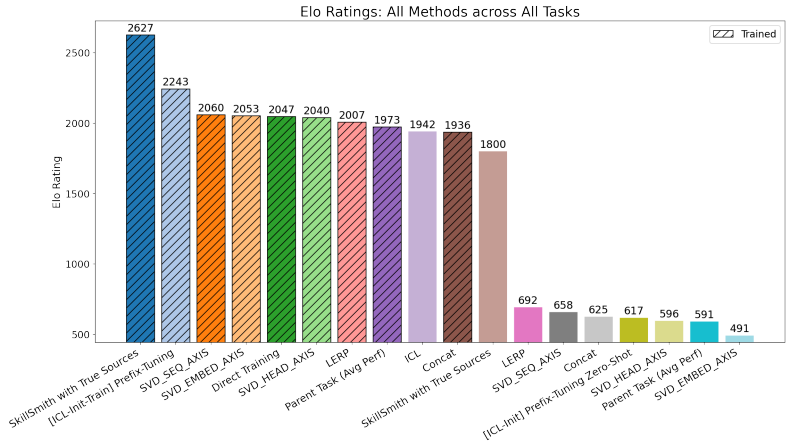

Figure 3: Elo scores of various methods on 15 heldout CSNI tasks.

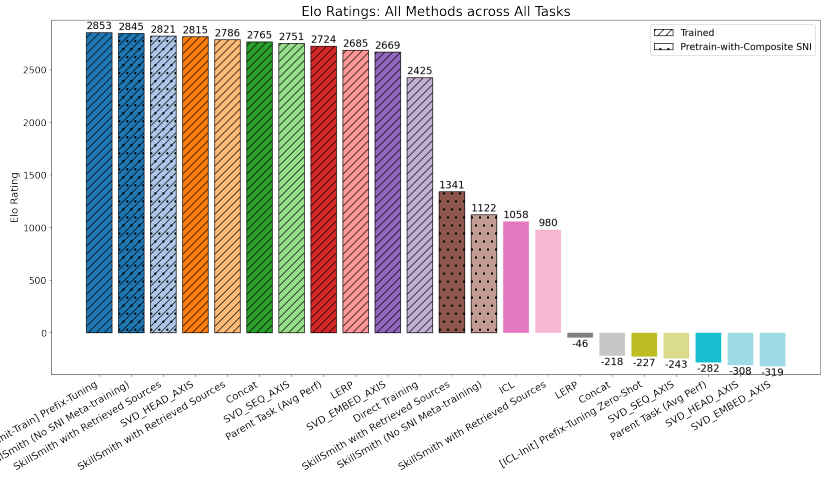

Figure 4: Elo scores are computed over 10 heldout SNI tasks used as meta-test tasks.

that drastically outperform both the strongest non-compositional baseline (prefix-tuning initialized via ICL) and continuing training after traditional arithmetic weight-merging. These results underscore that reasoning over both text metadata and parametric skills provides superior parameter initialization, allowing the model to adapt to tasks much more effectively than relying on either modality in isolation.

5.2 SkillSmith is effective in the wild

Composite-SNI served as an ideal test bed for SkillSmith because it has a large number of meta-train tasks ($\sim$17K) and the ground truth set of source tasks is known. With the SNI and MMLU-ProX datasets, we investigate SkillSmith's ability to handle settings where (i) we are meta-task poor (SNI has $\sim$800 tasks, MMLU-ProX has $\sim$250) and (ii) the relationship between tasks can only be inferred.

We use the retrieval based approach outlined in Section 4.4.2 to map each task to a relevant pair of parent source tasks. For any meta-training task $T \in \mathcal{T}_{train}$, its source tasks must be drawn from the training pool excluding itself: $\mathcal{T}_{src} \subset \mathcal{T}_{train} \setminus \{T\}$. Further, for any meta-evaluation task $T \in \mathcal{T}_{eval}$, its source tasks must also be drawn from the training set: $\mathcal{T}_{src}[T] \subset \mathcal{T}_{train}$ with $\mathcal{T}_{train} \cap \mathcal{T}_{eval} = \emptyset$. We also investigate bootstrapping the SkillSmith trained for the Composite SNI task by using it as initialization in the meta-poor setting of MMLU-ProX and SNI.

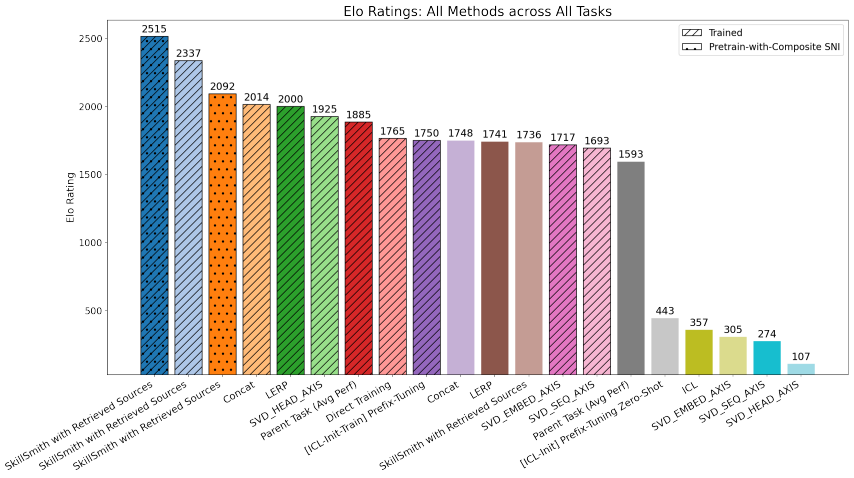

Figure 5: Elo scores over 6 MMLU-ProX tasks used as meta-test tasks.

5.2.1 SUPER-NATURAL INSTRUCTIONS

Figure 4 showcases SkillSmith's efficacy even under the above constraints. In the zero-shot regime, the trends mirror those of Figure 3: SkillSmith variants and the ICL baseline outperform all other methods. Training SkillSmith exclusively on SNI data without bootstrapping from a pre-trained Composite-SNI checkpoint underperforms ICL. However, evaluating the raw CSNI checkpoint directly on the original SNI tasks (SkillSmith (No SNI Meta-Training)) outperforms ICL, showing that the capabilities acquired on synthetic CSNI data natively transfer to an organic task distribution. The best-performing zero-shot method is achieved by fine-tuning the CSNI checkpoint with SNI data, which addresses the concern of data scarcity whilst performing in-domain adaptation to the SNI data distribution.

In contrast with the zero-shot results, task-specific fine-tuning on downstream data yields marginal separation in Elo ratings among the top three evaluated methods. The win rate between these methods converges to 0.5, indicating no significant performance differences. We posit that this convergence is likely due to two factors: first, SNI tasks feature significantly higher instance counts per task ($O(1000)$) relative to Composite-SNI; second, because SNI was established in 2022 (Wang et al., 2022), it primarily comprises primitive NLP tasks—such as sentiment classification or character concatenation—that are arguably easy for contemporary language models. These two effects cause the top fine-tuning based methods – including SkillSmith– to cluster around the same performance ceiling.

5.2.2 MMLU-PROX

Unlike SNI, MMLU-ProX has no genealogical relationship with CSNI, yet consistent with the SNI results, zero-shot [Pretrain-with-Composite SNI] SkillSmith with Retrieved Sources, which uses the CSNI checkpoint as an initialisation to be trained on MMLU source tasks, outperforms all zero-shot baselines. More surprisingly, this configuration also outperforms all baselines that are allowed to *fine-tune* on downstream tasks. We can see concretely the value of *pre-training* SkillSmith on synthetic compositions, as training SkillSmith exclusively on the limited MMLU-ProX tasks without bootstrapping (second column from the left) yields a substantially lower zero-shot Elo of 1736.

While the downstream fine-tuning results on SNI clustered at the ceiling due to task simplicity, in the case of MMLU-ProX, a large performance gap is revealed. Fine-tuning the bootstrapped SkillSmith trained on MMLU task sources achieves the highest overall ELO rating, maintaining a clear margin over both the unbootstrapped model variant and the strongest weight-space baseline (Concat).

We attribute this pronounced performance gap to the dataset's sparsity and inherent difficulty. MMLU-ProX contains roughly 250 tasks total, and our evaluation targets subject categories across low-performing languages for the fixed, downstream model. Because these tasks are harder than SNI and lack an abundance of task instances, the methods do not converge around a performance ceiling. Under data limits, the superior parameter initialisation synthesised by the bootstrapped SkillSmith provides an advantage for downstream adaptation that direct training or uniform weight merging cannot match.

6 Analysis

6.1 Does SkillSmith actually use the K-V caches?

It is possible that SkillSmith does not actually leverage the source task prefix caches we provide as input. To verify, we ablate the impact of dropping out different parts of the input to SkillSmith.

The table on the right shows that SkillSmith learns to use both text meta-data and parametric information to generate prefix caches. Using only text-metadata outperforms using only weight space inputs, but results are best when both input types are present.

Input Configuration	Elo
No inputs	1209
Only K-V Caches	1455
All Inputs (no K-V Caches)	1622
All Inputs	**1714**

6.2 Where do SkillSmith's performance gains come from?

To isolate whether SkillSmith's performance gains stem from its architecture or simply from access to richer textual context, we evaluate a text-augmented direct training baseline. We extract all auxiliary textual metadata provided to SkillSmith (source texts w_i and combination text) and prepend it to the target task's input examples during standard prefix tuning. As illustrated in Figure 6, while auxiliary text improves standard direct training, it consistently falls short of SkillSmith across the task set. This performance gap demonstrates that SkillSmith successfully learns a synergistic composition of textual metadata and parametric weights.

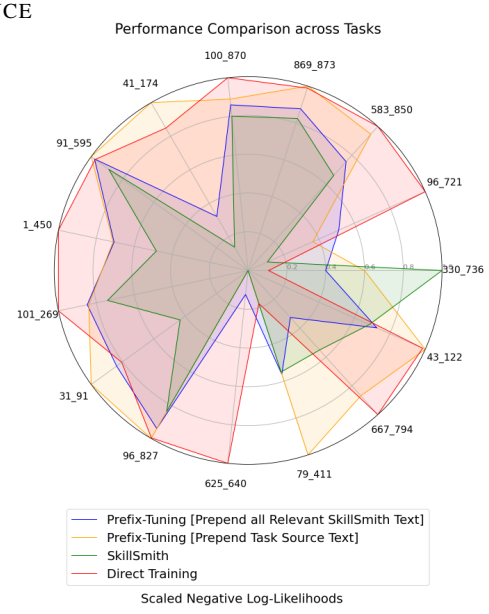

Figure 6: NLLs scaled to (0, 1). ↓ area is better.

7 Conclusion

This work introduces SkillSmith, an architecture designed to bridge the gap between textual reasoning and parametric adaptation. Our evaluation on the Composite-SNI dataset demonstrates that SkillSmith consistently outperforms standard weight-space merging techniques in both zero-shot settings and finetuning settings. Further, our experiments on SNI and MMLU-ProX highlight SkillSmith's effectiveness in "in-the-wild" scenarios where ground-truth task mappings are unavailable. While our findings on SNI indicate that abundant downstream training data on simpler tasks allows standard baselines to reach a similar performance ceiling, SkillSmith proves remarkably advantageous in challenging, data-sparse environments. On the MMLU-ProX benchmark, the superior parameter initialization synthesized by a bootstrapped SkillSmith provides an advantage for downstream adaptation that direct training or weight merging cannot match.

Our work does not come without limitations. Training SkillSmith involves constructing $\mathcal{D}^{\text{train}}$, which requires training hundreds of KV caches – a compute intensive undertaking. We note, however, that once trained, SkillSmith effectively amortizes the (from-scratch) prefix-tuning cost across every downstream target task. Also, we train SkillSmith using cross entropy loss and correspondingly evaluate methods using target negative log-likelihood (NLL) as our metric. Ideally, for MMLU-ProX, we would train and evaluate with accuracy as the metric but training SkillSmith with reinforcement learning was beyond our compute budget. We hope such an exploration will be conducted in future work.

Ultimately, SkillSmith represents a step toward more holistic agentic architectures—systems that no longer view "knowing" (text) and "doing" (weights) as orthogonal, but as a unified basis for instruction-steered adaptation.

AI USE STATEMENT

We used generative AI to assist in the writing of this paper, particularly in polishing the narrative and fixing grammar issues. Additionally, we used generative AI to synthetically generate the CSNI dataset (details above). The code used to prompt generative AI to do this was manually written. To verify correctness, we sampled the generated instances (in the order of 10-30 instances) and verified their coherence. Code to train, evaluate, and analyse SkillSmith was manually written and polished with the help of generative AI. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r.

Anthropic. Introducing the next generation of claude, 2024. URL https://www.anthropic.com/news/claude-3-family.

Akari Asai, Mohammadreza Salehi, Matthew E. Peters, and Hannaneh Hajishirzi. Attempt: Parameter-efficient multi-task tuning via attentional mixtures of soft prompts, 2022. URL https://arxiv.org/abs/2205.11961.

Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995. doi: 10.1137/0916069. URL https://doi.org/10.1137/0916069.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, et al. Faith and fate: Limits of transformers on compositionality. *Advances in neural information processing systems*, 36:70293–70332, 2023.

Google. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025a. URL https://arxiv.org/abs/2507.06261.

Google. Gemma 3 technical report, 2025b. URL https://arxiv.org/abs/2503.19786.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=lIsCS8b6zj.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning, 2022. URL https://arxiv.org/abs/2110.04366.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019. URL https://arxiv.org/abs/1902.00751.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. Lorahub: Efficient cross-task generalization via dynamic lora composition. *arXiv preprint arXiv:2307.13269*, 2023.

Yichen Huang and Lin F. Yang. Winning gold at imo 2025 with a model-agnostic verification-and-refinement pipeline, 2025. URL https://arxiv.org/abs/2507.15855.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic, 2023. URL https://arxiv.org/abs/2212.04089.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. Measuring compositional generalization: A comprehensive method on realistic data. *arXiv preprint arXiv:1912.09713*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring ai ability to complete long software tasks, 2026. URL https://arxiv.org/abs/2503.14499.

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pp. 2873–2882. PMLR, 2018.

Jinhyuk Lee, Feiyang Chen, Sahil Dua, Daniel Cer, Madhuri Shanbhogue, Iftekhar Naim, Gustavo Hernández Ábrego, Zhe Li, Kaifeng Chen, Henrique Schechter Vera, Xiaoqi Ren, Shanfeng Zhang, Daniel Salz, Michael Boratko, Jay Han, Blair Chen, Shuo Huang, Vikram Rao, Paul Suganthan, Feng Han, Andreas Doumanoglou, Nithi Gupta, Fedor Moiseev, Cathy Yip, Aashi Jain, Simon Baumgartner, Shahrokh Shahi, Frank Palma Gomez, Sandeep Mariserla, Min Choi, Parashar Shah, Sonam Goenka, Ke Chen, Ye Xia, Koert Chen, Sai Meher Karthik Duddu, Yichang Chen, Trevor Walker, Wenlei Zhou, Rakesh Ghiya, Zach Gleicher, Karan Gill, Zhe Dong, Mojtaba Seyedhosseini, Yunhsuan Sung, Raphael Hoffmann, and Tom Duerig. Gemini embedding: Generalizable embeddings from gemini, 2025. URL https://arxiv.org/abs/2503.07891.

Tao Lei, Junwen Bai, Siddhartha Brahma, Joshua Ainslie, Kenton Lee, Yanqi Zhou, Nan Du, Vincent Y. Zhao, Yuexin Wu, Bo Li, Yu Zhang, and Ming-Wei Chang. Conditional adapters: Parameter-efficient transfer learning with fast inference, 2023. URL https://arxiv.org/abs/2304.04947.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021. URL https://arxiv.org/abs/2104.08691.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation, 2021. URL https://arxiv.org/abs/2101.00190.

Luyang Liu, Jonas Pfeiffer, Jiaxing Wu, Jun Xie, and Arthur Szlam. Deliberation in latent space via differentiable cache augmentation. *arXiv preprint arXiv:2412.17747*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 46–54, 2020.

Qwen. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Matthew Renze and Erhan Guven. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*, 2024. URL https://arxiv.org/abs/2405.06682.

Pavan C Shekar and Aswanth Krishnan. Adaptive minds: Empowering agents with lora-as-tools, 2026. URL https://arxiv.org/abs/2510.15416.

Donald Shenaj, Ondrej Bohdal, Mete Ozay, Pietro Zanuttigh, and Umberto Michieli. Lora. rar: Learning to merge loras via hypernetworks for subject-style conditioned image generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 16132–16142, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

George Stoica, Pratik Ramesh, Boglarka Ecsedi, Leshem Choshen, and Judy Hoffman. Model merging with svd to tie the knots, 2024. URL https://arxiv.org/abs/2410.19735.

Weiwei Sun, Xuhui Zhou, Weihua Du, Xingyao Wang, Sean Welleck, Graham Neubig, Maarten Sap, and Yiming Yang. Training proactive and personalized llm agents. *arXiv preprint arXiv:2511.02208*, 2025.

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou, and Daniel Cer. Spot: Better frozen model adaptation through soft prompt transfer, 2022. URL https://arxiv.org/abs/2110.07904.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=22pyNMuIoa.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. In *Proceedings of the 2022 conference on empirical methods in natural language processing*, pp. 5085–5109, 2022.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025a. URL https://arxiv.org/abs/2502.12110.

Xiwei Xu, Robert Mao, Quan Bai, Xuewu Gu, Yechao Li, and Liming Zhu. Everything is context: Agentic file system abstraction for context engineering, 2025b. URL https://arxiv.org/abs/2512.05470.

Weihao Xuan, Rui Yang, Heli Qi, Qingcheng Zeng, Yunze Xiao, Aosong Feng, Dairui Liu, Yun Xing, Junjue Wang, Fan Gao, Jinghui Lu, Yuang Jiang, Huitao Li, Xin Li, Kunyu Yu, Ruihai Dong, Shangding Gu, Yuekang Li, Xiaofei Xie, Felix Juefei-Xu, Foutse Khomh, Osamu Yoshie, Qingyu Chen, Douglas Teodoro, Nan Liu, Randy Goebel, Lei Ma, Edison Marrese-Taylor, Shijian Lu, Yusuke Iwasawa, Yutaka Matsuo, and Irene Li. Mmlu-prox: A multilingual benchmark for advanced large language model evaluation, 2025. URL https://arxiv.org/abs/2503.10497.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025.

Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, et al. Memento: Fine-tuning llm agents without fine-tuning llms. *arXiv preprint arXiv:2508.16153*, 2025.

A Composite SNI Dataset Construction

We construct Composite-SNI a synthetic dataset, where each task $t_1_t_2$ is constructed by combining pairs of tasks t_1 and t_2 from the SNI (Wang et al., 2022) dataset. Below, we outline the steps we took to construct this dataset and to ensure that the final set of generated tasks are of high quality.

A.1 Data Generation Procedure

A.1.1 Initial Task Generation

We first create a set of all pairs of tasks from the base SNI dataset. We prompt Gemini 2.5 Pro with each of pair of tasks (task description and few-shot examples) and ask for it to generate a new task that leverages the underlying set of skills required to solve the individual tasks in the original pair. We also prompt the model to give a rationale for why the generated task is a good one, and a score from 1-5 of the perceived quality of the generation. This step results in O(350K) generated tasks. Below is the full prompt that we use.

Prompt 1: Synthetic Dataset Generation Task

You are a polymath, tasked with creating a synthetic instruction-following dataset. Your expertise spans a wide range of domains and technical areas, allowing you to find deep connections between disparate tasks. Your specific task is to take a set of existing instructions and their example solutions, and synthesize them into a new, well-thought-out instruction. This new instruction must be an interesting, realistic, and relatively challenging task that a language model could reasonably solve. It should represent a deep, semantic, and logical combination of the provided seed instructions.
For the new instruction you create, YOU MUST ALSO PROVIDE 128 TO 512 EXAMPLE SOLUTIONS. After creating the combined instruction, you will score its quality on a scale of 1 (very poor) to 5 (excellent) and provide a rationale for your score.
Your input will follow this exact format:

 # Begin Input Instructions #

Instruction 1
Begin Task Description
((task description goes here))
Example Begin
\t[Input]: ((example goes here))
[Targets]: ((example goes here))
. . .
Example Begin
\t[Input]: ((example goes here))
[Targets]: ((example goes here))

Instruction 2
Begin Task Description
((task description goes here))
Example Begin
\t[Input]: ((example goes here))
[Targets]: ((example goes here))
. . .
Example Begin
\t[Input]: ((example goes here))
[Targets]: ((example goes here))

 # End Input Instructions #

Your output should follow the following format:

 # Begin Output #

Begin Instruction
((task description goes here))
Example Begin
\t[Input]: ((example goes here))

\t[Targets]: ((example goes here))
...
Example Begin
\t[Input]: ((example goes here))
\t[Targets]: ((example goes here))
...
End Instruction
\t[[Score]]: ((score))
\t[[Rationale]]: ((rationale))

 # End Output #

Note that ellipses means there could be more examples / instructions your can generate.
PLEASE STICK TO THIS OUTPUT FORMAT EXACTLY ELSE WE WILL HAVE DOWNSTREAM PARSING ERRORS !
DO NOT FORGET YOU MUST ALSO PROVIDE 128 TO 512 EXAMPLE SOLUTIONS.
If an example of the instruction does not have a target, you can leave it blank
Here is an example to guide you:

>>> User Input:
 # Begin Input Instructions #

Instruction 1
Begin Task Description
Give an example of an object that is spherical in shape
Example Begin
\t[Input]: basket ball
[Targets]:
Example Begin
\t[Input]: orange
[Targets]:
Example Begin
\t[Input]: eyeball
[Targets]:

Instruction 2
Begin Task Description
Write a short poem a comedian
Example Begin
\t[Input]: There was once a funny man, who did funny things, he slipped and fell.
[Targets]: The whole crowd laughed

 # End Input Instructions #

>>> Your Output:
 # Begin Output #

Begin Instruction
Present a joke a comedian would tell about a ball
Example Begin
\t[Input]: I tried to start a juggling club, but I couldn't get the ball rolling. After many failed attempts, I decided to just throw in the towel.
\t[Targets]:

End Instruction
\t[[Score]]: 5
\t[[Rationale]]: This is a good instruction because it recognizes that a poem by a comedian can be a joke, and also realises that a ball has spherical shape. The example given is also a good one that semantically integrates the essence of the two instructions.

 # End Output #

OK, NOW HERE WE GO WITH AN ACTUAL TASK.

{{text}}

A.1.2 TASK FILTRATION

Given the large number of tasks generated, we take the following steps to prune the list down to only a highest quality set of generations:

1. We remove all tasks that have a self-reported (as part of the task generation) quality score ≤ 3.
2. We remove all tasks that have ¡ 16 accompanying generated task examples.
3. We perform 2 rounds of LLM-mediated task filtration. Specifically, we group the remaining tasks randomly into pairs, and again prompt Gemini 2.5 to select which of the tasks is better based on a pre-defined rubric with axes such as clarity, actionability and diversity of task examples. We expose our exact prompt below.

These filtration steps result in a final dataset size of O(90K) tasks.

Prompt 2: LLM Prompt Evaluator Task

You are an expert evaluator of instruction-following prompts for large language models. Your primary goal is to identify instructions that are strong candidates for inclusion in a dataset focused on interesting and complex, multi-step composite instructions. You will compare two instruction-following examples, each consisting of an instruction (prompt) and examples of how an LLM followed that instruction.

Criteria for a Superior Instruction:
When evaluating, prioritize the following properties in the instruction itself and the quality of its examples:
Clarity and Specificity: Is the instruction unambiguous, easy to understand, and precise? Does it clearly define the task, expected output format, and any critical constraints?
Conciseness: Is the instruction brief and to the point, avoiding unnecessary jargon, repetition, or verbose explanations, without sacrificing clarity or specificity?
Actionability: Does the instruction describe a task that an LLM can perform effectively and directly, leading to a clear, well-defined, and consistent output?
Output Diversity & Quality: Does it encourage a robust and diverse range of high-quality, appropriate outputs from the LLM, rather than being overly narrow, trivial, or leading to repetitive responses?
Quality of Examples: Do the provided examples demonstrate clear, correct, and varied adherence to the instruction? Do they effectively showcase the intended behavior and the range of acceptable outputs?

Your Task:
Evaluate which of the two provided instruction-following examples is better overall, considering all the desirable properties listed above, with a strong emphasis on correctness, diversity and robustness.
The instructions will come in the following input template:

 # Begin Input Instructions #
Instruction 1
. . .
Instruction 2
. . .
 # End Input Instructions #

Where the ellipses (. . .) represent the actual instruction-following task.

Output Format:
[[Rationale]]: ((reason))
[[Result]]: ((output))

Return 0 as output if the first instruction is superior.
Return 1 as output if the second instruction is superior.
Populate the rationale field with the reasons why you think your chosen instruction is superior to the other. Make sure to keep your rationale short and succint.

PLEASE STICK TO THIS OUTPUT FORMAT EXACTLY ELSE WE WILL HAVE DOWNSTREAM PARSING ERRORS !
Strictly adhere to the output format (either '1' or '0').

Now below is a task for your to solve:

{{text}}

A.1.3 TASK CORRECTNESS FILTRATION

The previous filtration step was based on comparison with other tasks. For this stage, we filtered based on task correctness. We first performed a manual inspection of a random sample of the generated tasks to come up with good examples of violations that render a task incorrect. This manual inspection also allows us to estimate what we should a-priori expect to be a good retention rate after bulk LLM based filtration based on correctness. Informed by the manual inspection of tasks, we construct the following prompt for Gemini 2.5 to select tasks based on correctness, internal consistency and factuality.

Prompt 3: Expert Evaluator Task

You are a rigorous and critical expert evaluator tasked with curating high-quality instruction-following datasets for Large Language Models. Your primary goal is to assess candidate tasks and identify those suitable for a dataset focused on interesting, complex, and multi-step composite instructions.

You will be presented with a candidate task, which includes an instruction (prompt) and several example executions (input/target pairs).

Your Objective:
Evaluate the candidate task against three critical criteria (detailed below) and decide whether to include it (1) or exclude it (0) from the dataset.

Input Format:
The task will be provided within the following delimiters:

Begin Task Description
[Instruction definition]
Example 1
[Input] . . .
[Target] . . .
. . .
Example N
[Input] . . .
[Target] . . .
End Task Description

Output Format:
You must adhere strictly to the following output format. Do not add any extra text before or after this structure. Failure to comply will cause downstream parsing errors.

[[Rationale]]: ((Your reasoning))
[[Score]]: ((1 or 0))

- **[[Rationale]]:** Provide a justification for your decision. Be concise but make sure to include relevant details even in the case where the task passes all the relevant details, explain succinctly why. Ensure it is specific enough to justify the result, particularly when detailing errors found in examples. You must explicitly mention which criteria (e.g., "Criterion 1") were decisive in your evaluation.
- **[[Score]]:** Return 0 if the task fails any of the criteria. Return 1 if the task passes all criteria.

Evaluation Criteria
A superior instruction-following task must satisfy ALL THREE of the following criteria. Evaluate them sequentially.

Criterion 1: Absolute Correctness (Critical Veto)
The example executions (Targets) must be demonstrably correct and perfectly follow the instruction definition. This is the most important criterion.

Action: Meticulously verify the execution of every example provided.
Fails if:

- ANY example is factually incorrect.
- ANY example fails to adhere to the constraints or steps specified in the instruction.

Example of a Failure (Criterion 1):

Begin Task Description

You are given a sentence containing a blank (_) and two strings, A and B. Your task is to first identify the longest common substring between strings A and B. Then, convert this substring to lowercase and sort its characters alphabetically. Replace the original substring in both A and B with this modified version. Finally, determine which of the modified strings, A or B, logically fills the blank in the sentence based on its content. Your answer should be the modified string (either A or B) that best fits the sentence.
Example 0
[input] Sentence: The _ was used to build the houses instead of the other material as it was plentiful.
String A: boardsXYZplentiful
String B: bricksXYZplentiful
[target] boardsxyzplentiful
Example 1
[input] Sentence: The insecticide could not kill the insect flying in the house because the _ is weak.
String A: insecticideWEAKsubstance
String B: insectWEAKsubstance
[target] insecticideaekswubstance
Example 2
[input] Sentence: Tom decided to repaint his bedroom yellow instead of the gray it was now. The _ is bright.
String A: yellowBRIGHTcolor
String B: grayBRIGHTcolor
[target] yellowbceghilorrt
End Task Description

[[Rationale]]: This task fails Criterion 1 (Absolute Correctness). In Example 0, the instruction requires the longest common substring ("XYZplentiful") to be lowercased and sorted alphabetically. The target "boardsxyzplentiful" shows the substring lowercased but NOT sorted (the sorted version would be "efillnptuxyz"). Example 1 has the same sorting issue. Example 2 is also incorrect: the substring "BRIGHTcolor" sorted alphabetically is "bcghiloorrt", but the target shows "bceghilorrt" (an extra 'e').
[[Score]]: 0

Criterion 2: Executability and Unambiguity
The instruction must be clearly defined, executable, and unambiguous. A reasonably strong language model should be able to execute the task consistently, leading to a single, objective answer.

Action: Analyze the instruction for clarity, subjectivity, and hidden assumptions.
Fails if:

- The task cannot be executed given the information in the prompt, possibly because of too much ambiguity.
- The instructions are ambiguous, contradictory, or poorly defined.
- The task requires unspecified real-time access or external knowledge that is not provided in the prompt.

Example of a Failure (Criterion 2):

Begin Task Description
Given a command in a limited form of natural language, provide the correct sequence of actions that executes the command to navigate an agent in its environment. Additionally, classify the overall emotion conveyed by the sequence of actions into one of the six emotions: 'joy', 'love', 'anger', 'fear', 'sadness', or 'surprise'.

[. . . instruction details on movement commands omitted for brevity . . .]

The emotion classification should reflect the overall feeling evoked by the agent's movement pattern. For example, rapid, erratic movements might suggest 'fear' or 'anger'. . .
Example 0
[input] run opposite left twice and jump opposite right
[target] Actions: I_TURN_LEFT I_TURN_LEFT I_RUN I_TURN_LEFT I_TURN_LEFT I_RUN I_TURN_RIGHT I_TURN_RIGHT I_JUMP, Emotion: anger
End Task Description

[[Rationale]]: This task fails Criterion 2 (Executability and Unambiguity). The requirement to classify emotion based on the movement pattern ("The emotion classification should reflect the overall feeling evoked...") is highly subjective. There is no objective standard for determining if the emotion is 'anger' versus 'fear', leading to multiple valid interpretations.
[[Score]]: 0

Criterion 3: Complexity, Diversity, and Interest

The task should be non-trivial, aligning with the goal of collecting complex, multi-step composite instructions. This means the task should require multiple steps or the composition of different skills (e.g., combining math and text analysis, or logic and formatting). Furthermore, the provided examples must be diverse.

Action: Assess the cognitive load required (is it multi-step? does it combine skills?) and the variation across the examples.
Fails if:

- The task is too simple (e.g., basic sentiment classification, simple data extraction).
- The task is repetitive or easily gameable (e.g., multiple-choice questions where the answer is always A).
- The examples lack diversity in input structure, content, logic, or output format.

Example of a Pass (All Criteria):

Begin Task Description
You are given a snippet of a privacy policy and a multiple-choice question about basic statistics related to data privacy. Your task is to first identify the type of personal information mentioned in the privacy policy snippet. Then, answer the multiple-choice statistics question related to data privacy practices. Provide the identified information type and the letter corresponding to the correct multiple-choice answer.
Example 0
[input] Privacy Policy Snippet: "Our website uses cookies and similar tracking technologies..."
Statistics Question: A company analyzes website traffic data. They find that the average time spent on their site is 180 seconds with a standard deviation of 30 seconds. Assuming the time spent follows a normal distribution, what percentage of users spend between 150 and 210 seconds on the site?
(A) 34% (B) 68% (C) 95% (D) 99.7%
[target] Information Type: Cookies and tracking elements
Answer: B
Example 1
[input] Privacy Policy Snippet: "To create an account, we require your email address and a chosen username..."
Statistics Question: A survey asks users if they are comfortable providing their email address for account creation. 75 out of 100 surveyed users said yes. What is the sample proportion of users comfortable providing their email address?
(A) 0.25 (B) 0.50 (C) 0.75 (D) 1.00
[target] Information Type: Contact
Answer: C
End Task Description

[[Rationale]]: The task passes all criteria. Criterion 1 (Correctness): The solutions to the statistics problems are correct. Criterion 2 (Executability): The instructions are clear and the answers are objective. Criterion 3 (Complexity/Diversity): The task is complex as it requires composing text analysis (privacy policy) with mathematical reasoning (statistics). The examples are diverse (different statistical concepts and variable answer positions).
[[Score]]: 1

Evaluation Process Checklist
When evaluating the task below, strictly follow this sequential process. If a task fails a step, stop evaluating and reject it immediately.

1. Verify Correctness (C1): Check every example meticulously. If any fail, reject (0) and explain the specific error.
2. Assess Executability (C2): If C1 passes, check for ambiguity or subjectivity. If found, reject (0).
3. Evaluate Complexity and Diversity (C3): If C1 and C2 pass, assess if the task is sufficiently complex and the examples diverse. If not, reject (0).

Accept (1): Only if the task passes C1, C2, and C3.

BELOW IS YOUR TASK TO EVALUATE. Remember to strictly follow the output format.

A.2 Final Dataset Statistics

Figures 7, 8 show relevant statistics relating to the final Composite SNI dataset.

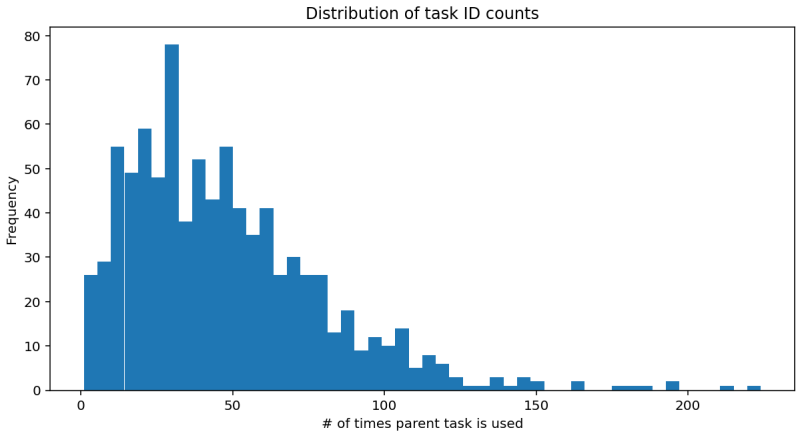

Figure 7: Distribution of occurrence statistics of base SNI task appearance in final composite-SNI task list. Some SNI tasks feature in over 200 composite-SNI tasks in the final dataset but most appear in an average of 48 downstream tasks.

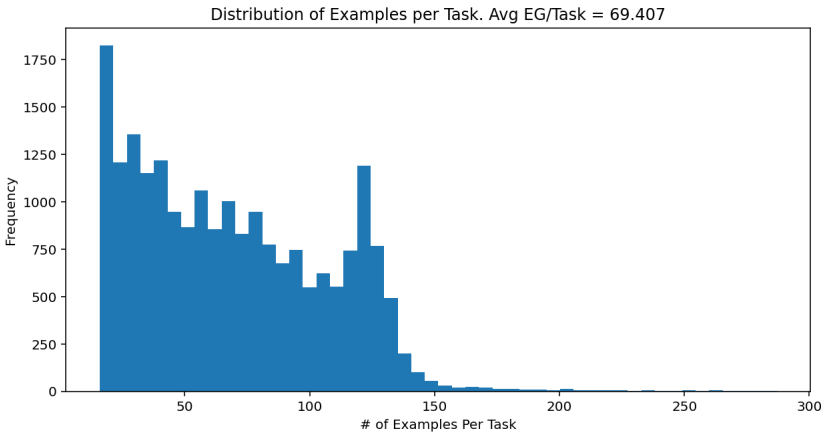

Figure 8: A few of the generated tasks have a large number of example instances (¿ 150). The average number of instances generated per task is ≈ 69

A.3 Example Composite SNI Task

Prompt 4: [Composite-SNI: 62_193] C++ Pronoun Coreference Task

Begin Task Description
Given a C++ program and a sentence describing a pronoun coreference relation within it, determine if the reasoning provided for the coreference is correct. The C++ program may contain 'for' loops. Your task is to first count the number of 'for' loops in the provided C++ code snippet. Then, analyze the sentence and the reasoning provided for the pronoun coreference. Finally, output the count of 'for' loops and whether the reasoning is 'Correct' or 'Wrong'.

Example Begin
\t[input] C++ Code:

```
int main() {
    int a = 1, b = 2, c = 3;
    for (int i = 0; i < 1; i++) {
        // He declared multiple variables on one line.
    }
    return 0;
```

}

Sentence: He declared multiple variables on one line.
Reason: The 'He' refers to the programmer because declaration style is a programmer's choice.
Question: How many 'for' loops are in the code, and is the reasoning correct or wrong?
[target] 1, Correct

Example Begin
\t[input] C++ Code:

```
int main() {
    for (int i = 0; i < 10; i++) {
        if (i == 5) {
            break; // He exited the loop when i reached 5.
        }
    }
    return 0;
}
```

Sentence: He exited the loop when i reached 5.
Reason: The 'He' refers to the 'break' statement because 'break' causes the exit.
Question: How many 'for' loops are in the code, and is the reasoning correct or wrong?
[target] 1, Wrong

Example Begin
\t[input] C++ Code:

```
int main() {
    int x = 10;
    int y = x / 2;
    // It holds half the value of x.
    return y;
}
```

Sentence: It holds half the value of x.
Reason: The 'It' refers to the variable 'y' because 'y' is assigned the result of the division.
Question: How many 'for' loops are in the code, and is the reasoning correct or wrong?
[target] 0, Correct

The above task was constructed from the following source tasks derived from the SNI dataset:

Prompt 5: [SNI Task: 62] C++ For Loop Counting Task

Begin Task Description
This task is to find the number of 'For' loops present in the given cpp program.

Example Begin
\t[input]

```
int turn(int a,int b)
{
    int i,k;
    if(a==1) k=1;
    else if(a<b) k=0;
    else
    {
        k=0;
        for(i=b;i<=a;i++)
            if(a%i==0)
                k=k+turn(a/i,i);
    }
    return k;
}
int main()
{
    int n,i,a[1000],t;
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
```

```
        t=turn(a[i],2);
        printf("%d\n",t);
    }
    return 0;
}
```

[target] 2

Example Begin

\t[input]

```
int devide(int n,int k)
{
    int s,i;
    s=1;
    for(i=k;i<=sqrt(n*1.0);i++)
    {
        if(n%i==0)
        {
            s=s+devide(n/i,i);
        }
    }
    return s;
}
int main()
{
    int n,a[200],i;
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
        if(i==0)printf("%d",devide(a[i],2));
        else printf("\n%d",devide(a[i],2));
    }
    return 0;
}
```

[target] 2

Prompt 6: [SNI Task 193] Pronoun Coreference Reasoning Task

Begin Task Description

In this task you need to indicate the plausibility of reasoning for the pronoun coreference relations. Each of the provided inputs contains a sentence with a target pronoun and a sentence that justifies which noun phrase the pronoun refers to. Correct reasons do not need to use all the knowledge from the sentence. The resolution of the pronoun coreference relations typically involve one or multiple following knowledge types about commonsense: First: 'Property', the knowledge about property of objects (e.g., ice is cold). Second: 'Object', the knowledge about objects (e.g., cats have ears). Third: 'Eventuality', the knowledge about eventuality (e.g., 'wake up' happens before 'open eyes'). Fourth: 'Spatial', the knowledge about spatial position (e.g., object at the back can be blocked). Fifth: 'Quantity', the knowledge about numbers (e.g., 2 is smaller than 10). Sixth: all other knowledge if above ones are not suitable. You should answer 'Correct' if the reasoning made sense, otherwise, you should answer 'Wrong'.

Example Begin

\t[input]Sentence: Jackson was greatly influenced by Arnold, though he lived two centuries earlier.
Reason: The 'he' refers to arnold because "Was" is past tense. He was influenced by Arnold. So you assume Arnold is dead.
Question: Is the above reasoning correct or wrong?
[target]Correct

Example Begin

\t[input]Sentence: I put the heavy book on the table and it broke.
Reason: The 'it' refers to the table because A butterfly wing would likely not be heavy enough to break a table.
Question: Is the above reasoning correct or wrong?
[target]Wrong

Example Begin
\t[input]Sentence: The sculpture rolled off the shelf because it wasn't level.
Reason: The 'it' refers to the shelf because The object not anchored is the one likely to roll.
Question: Is the above reasoning correct or wrong?
[target]Wrong

B RETRIEVAL DETAILS

Given a library $\mathcal{T}_{src}[T] = T_1, \ldots, T_N$ of N source tasks, the module retriever identifies which subset of tasks is most relevant to a novel query task T.

Task Embedding. Each source task T_i in the library is associated with a training dataset $\mathcal{D}_i$. To obtain a fixed-dimensional representation of a task, we sample K training examples from $\mathcal{D}_i$ and embed each example independently using a pretrained text embedding model. Concretely, for each example $x_j^{(i)} \in \mathcal{D}_i$, we compute $\mathbf{e}_j^{(i)} = \texttt{Embed}(x_j^{(i)}) \in \mathbb{R}^d$, yielding an embedding table $E_i = [\mathbf{e}_1^{(i)}, \ldots, \mathbf{e}_K^{(i)}] \in \mathbb{R}^{K \times d}$ for each task. In our experiments, we use K=16 training examples per task and the Gemini Embedding model (gemini-embedding-001) (Lee et al., 2025) with the classification task type.

Retriever Architecture. We frame task retrieval as an N-way classification problem over the source tasks. The retriever is a lightweight multilayer perceptron (MLP) $f_\theta : \mathbb{R}^d \to \mathbb{R}^N$ with hidden layers of sizes $(512, 256)$ and ReLU activations. Each class corresponds to one task m_i in the source-task library. The MLP is trained on the union of all per-task embedding tables:

$$\mathcal{T} = \bigcup_{i=1}^{N} \{(\mathbf{e}_j^{(i)}, i) \mid j = 1, \ldots, K\}$$

using the standard softmax cross-entropy loss where the input is the embedding $e_j \in \mathbb{R}^d$ and the label to predict is i, the unique task identifier.

The training data is split 80/20 into train and validation sets. We optimize with Adam (learning rate 10^{-3}, batch size 128) for up to 1000 epochs with early stopping.

Inference: Ensemble Scoring. At inference time, given a query task with K embedded examples $E_q = [\mathbf{e}_1^{(q)}, \ldots, \mathbf{e}_K^{(q)}]$, we compute an ensemble score for each source task in the library by aggregating the per-example log-probabilities:

$$s_i = \sum_{j=1}^{K} \log \text{softmax}(f_\theta(\mathbf{e}_j^{(q)}))_i$$

The raw scores are then normalized via z-scoring and a $\tanh$ squashing function:

$$\hat{s}_i = \tanh\left(\frac{s_i - \mu_s}{\sigma_s}\right)$$

where μ_s and σ_s are the mean and standard deviation of the raw scores across all N tasks. Source tasks are ranked by $\hat{s}_i$ in descending order, and the tasks with the top-k scores are selected for downstream use.

Heuristic Grouping: After rankings are computed, candidate pairs are constructed from the semantic relevance rankings by grouping the top-ranked tasks sequentially (e.g., the top two form the first candidate pair, the next two form the second, etc.).

C LLM PAIR SELECTION DETAILS

Given τ and $N/2$ source-task pairs as given by the retriever, we use Gemini 2.5 to compare these pairs with τ and output the most relevant task pair. Specifically, each source task T_i (and also τ) is associated with a *task description* comprising a natural-language task definition and E input-output demonstration examples drawn from the task's training data. These are formatted using delimiter tags:

```
<instruction> [task definition] </instruction>
<example> [input_1] [target_1] </example>
<example> [input_2] [target_2] </example>
...
<example> [input_E] [target_E] </example>
```

In our experiments, we use $E = 2$ examples. Next, we ask an LLM, in our case Gemini 2.5 Pro, to process the task descriptions for the candidate pairs and the target task and to output the most relevant candidate pair along with a rationale:

Prompt 7: Pair Ranking for Target Task

```
I have a target task described below:
Target Task: {task_input}
I have a list of candidate pairs of other tasks. For each pair,
consider if knowing how to solve these two tasks would help solve
the target task.
Candidate Pairs:
{candidate_pairs}
Which pair is the MOST useful for the target task? Output the index
of the pair (0-indexed) and a brief reason.
Format: Pair Index: <index>
Reason: <reason>
```

D CALCULATING ELO RATINGS

We compute Elo scores via a two-step optimization process:

Empirical Win-Rate Matrix Estimation: We construct an $N \times N$ empirical win-rate matrix W, where N represents the total number of competitive methods. For any pair of distinct methods i and j, the observed win rate W_{ij} represents the empirical fraction of evaluation tasks where method i strictly outperforms method j (achieving a lower NLL score):

$$W_{ij} = \frac{1}{|\mathcal{T}_{eval}|} \sum_{T \in \mathcal{T}_{eval}} \mathbb{I}(\mathcal{L}_i(T) < \mathcal{L}_j(T))$$

where $\mathcal{L}_i(T)$ represents the evaluation loss of method i on task T, and $\mathbb{I}(\cdot)$ denotes the indicator function. Self-comparisons (W_{ii}) are uniformly set to 0.5.

Global Rating Optimisation: We initialise each method's Elo at 1500. Under the Bradley-Terry framework, the latent expected win rate E_{ij} of method i over method j is modeled as a logistic function of their underlying latent Elo ratings R_i and R_j:

$$E_{ij} = \frac{1}{1 + 10^{(R_j - R_i)/400}}$$

To extract the optimal latent scores that minimize variance across empirical interactions, we formulate rating estimation as a non-linear objective function minimizing the sum of squared errors between the observed and expected win rates:

$$\min_{\mathbf{R}} \sum_{i=1}^{N} \sum_{j \neq i} \left(W_{ij} - \frac{1}{1 + 10^{(R_j - R_i)/400}} \right)^2$$

This objective function is optimized numerically using the L-BFGS-B algorithm (Byrd et al., 1995), where the i-th entry of the optimized vector $\mathbf{R}$ defines the final global Elo rating for the i-th method.

We provide the code to compute the ELO from the win-rate matrix below:

```
def compute_elo_from_win_rate_matrix(win_rate_matrix,
    initial_ratings=None):
    """
    Computes Elo ratings from a win rate matrix using numerical
    optimization.

    Args:
        win_rate_matrix (np.ndarray): An N x N matrix where matrix[i,
    j] is the win
                                      rate of player i against player
    j.
        initial_ratings (np.ndarray, optional): Initial guess for the
    Elo ratings.
                                                If None, all ratings
    are initialized to 1500.

    Returns:
        np.ndarray: An array of converged Elo ratings for each player.
    """
    num_players = win_rate_matrix.shape[0]

    # Initialize ratings if not provided
    if initial_ratings is None:
        initial_ratings = np.full(num_players, 1500.0)

    def elo_expected_win_rate(rating_a, rating_b):
        """Calculates the expected win rate of player A against player
    B."""
        return 1.0 / (1.0 + 10.0 ** ((rating_b - rating_a) / 400.0))

    def objective_function(ratings):
        """
        Objective function to minimize.
        It calculates the sum of squared differences between observed
    and expected win rates.
        """
        error = 0.0
        for i in range(num_players):
            for j in range(num_players):
                # Skip self-comparison and invalid entries
                if i == j:
                    continue

                observed_win_rate = win_rate_matrix[i, j]
                # Skip if there's no data for this matchup (e.g.,
    matrix[i,j] = nan)
                if not np.isfinite(observed_win_rate):
                    continue

                expected_win_rate = elo_expected_win_rate(ratings[i],
    ratings[j])
                error += (observed_win_rate - expected_win_rate) ** 2
        return error

    # Solve the optimization problem
    result = scipy.optimize.minimize(objective_function,
    initial_ratings, method='L-BFGS-B')

    if result.success:
        return result.x
```

```
        else:
            raise RuntimeError("Optimization failed: " + result.message)
```

Along with the reported Elo Scores in Section 5, we present the unaggregated raw NLL scores for every baseline and task in Appendix L for transparency and verification alongside the relative rankings.

E SNI AND MMLU-PROX SETUP DETAILS

Super-natural Instructions (SNI) (Wang et al., 2022): A benchmark of 1,616 diverse natural language processing tasks across 76 semantic categories, i.e. text classification, information extraction, sequence tagging, text writing and composition. We use a subset of 875 tasks, each containing 1024 instances for our experiments.

F SOURCE TASK TRAINING HYPERPARAMETERS

For the initial source task library generation, prefix-cache sequence lengths are randomly sampled from $\{32, 64, 128\}$. The optimization grid sweeps across steps $\in \{200, 500, 1000\}$ and learning rates spanning $1e^{-2}$ to $1e^{-4}$. For all prefix K-Vs we train, we only train for the global layers, as prefix K-Vs with local layers eventually falls out of context.

MMLU-ProX (Xuan et al., 2025): To stress-test cross-lingual and cross-domain generalization, we leverage this multilingual dataset spanning 14 subject categories and 29 languages across five geographic regions. To construct a challenging evaluation split, we identified the lowest-performing languages for Gemma 3 4B within each region (per Table 6 in the Appendix of (Xuan et al., 2025)) and held out the three lowest-performing languages overall. Additionally, we randomly excluded three subject categories from training. Hence, the meta-training set comprises 26 languages and 11 categories, while the evaluation pool retains all 29 languages and 14 categories. This yields a two-pronged held-out breakdown: (i) 3 completely unseen languages evaluated across all 14 categories, and (ii) 26 seen languages evaluated across the 3 unseen categories. We evaluate on a final held-out space of 6 randomly sampled language-category pairs: `wolof_math`, `wolof_health`, `zulu_physics`, `spanish_law`, `indonesian_law`, and `afrikaans_history`.

G BASELINE DETAILS

Given a target task T_{new} paired with source tasks T_1 and T_2 and their corresponding task bundles $b_{T_1} = (w_{T_1}, m_{T_1})$ and $b_{T_2} = (w_{T_2}, m_{T_2})$, we compare SkillSmith against standard approaches across the following uni-modal baselines. Note that for all these methods (except ICL), we investigate both zero-shot and continued training on T_{new}-only data.

G.1 WEIGHT-SPACE ONLY

These baselines represent standard parameter-space model merging paradigms that combine learned behaviors efficiently but lack the capacity to process semantic instructions or relational task metadata.

LERP (Linear Interpolation): Performs simple parameter-space task arithmetic by averaging the weights of the two parent prefix caches element-wise:

$$m_{T_{\text{new}}} = \frac{m_{T_1} + m_{T_2}}{2}$$

This is conceptually equivalent to task arithmetic (Ilharco et al., 2023).

Concat: Directly concatenates both parent prefix caches along their sequence dimension, yielding a target prefix sequence length of 64 tokens. This method results in prefix-caches that have strictly more capacity than all other methods.

$$m_{T_{\text{new}}} = m_{T_1} \circ m_{T_2}$$

Source-Task Transfer: Computes the average zero-shot performance of the parent KV-caches on the target task. We also evaluate a prefix-tuning approach that initializes the target KV-cache using the parent KV-caches prior to continuation training. We report the averaged performance of this method, which is functionally equivalent to SPoT (Vu et al., 2022).

Singular Value Decomposition (SVD): Inspired by Stoica et al. (2024), we perform SVD across the sequence (`SVD_SEQ_AXIS`), head (`SVD_HEAD_AXIS`), or embedding (`SVD_EMBED_AXIS`) dimensions of the prefix caches. We stack N trained KV-caches along a new terminal dimension and permute the tensor to isolate the target axis d and the checkpoint axis N as the final two dimensions. SVD is applied independently to each $d \times N$ matrix slice ($M = U\Sigma V^T$). To prevent feature collapse and preserve the energy distribution of the parameters, we scale the first left singular vector ($U_{:,1}$) by the root mean square of all singular values:

$$m_{T_{\text{new}}} = U_{:,1}\sqrt{\frac{1}{N}\sum_{i=1}^{N}\sigma_i^2}$$

The output is subsequently permuted and squeezed back to the original tensor dimensions.

G.2 Text-Space Only

This approach leverages the base language model's default text comprehension capabilities without adapting its parameter space.

In-Context Learning (ICL): Evaluates the performance of k-shot ICL, where k is selected from $\{16, 32, 64\}$ based on validation set NLL. For tasks with constrained sample numbers, k is capped at the closest power of 2.

G.3 Transfer-less Adaptation

Direct Prefix Tuning: Optimizes a fresh target task prefix-cache from scratch on T_{new} under standard initialization, or initialized using the first 32 tokens of ICL examples from the target task. As detailed in Section 6.2, we also compare against a text-augmented direct baseline where all textual metadata available to SkillSmith (source texts and combo-text) is directly prepended to the input examples before prefix-tuning.

H Does SkillSmith actually generalize?

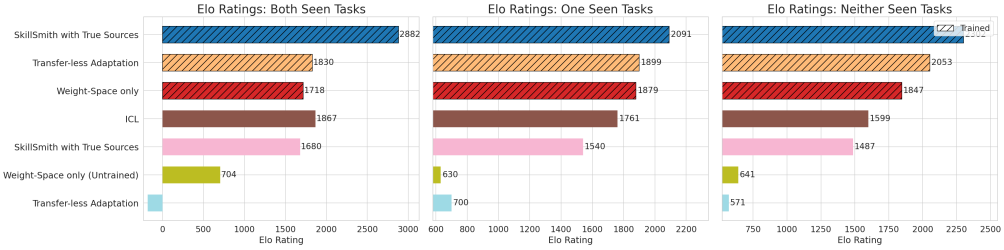

Figure 9: Results over different stratifications of CSNI.

To evaluate the ability of SkillSmith to generalize beyond the $\mathcal{T}_{src}$ encountered during training (c.f. Section H), we further stratify the meta-eval set into three subsets based on whether their constituent source tasks appear in the meta-train set: (i) **Both-Seen**: Both source tasks appear in at least one meta-train composite task; (ii) **One-Seen**: Only one source task appears in the meta-train set; and (iii) **Neither-Seen**: Neither source task appears in the meta-train set.

The results of Figure 3 are aggregated over 15 tasks, 5 each from the original CSNI sub-splits (**Neither-seen**, **One-seen**, **Both-seen**). To ensure if SkillSmith actually generalizes beyond the tasks seen during meta-training, we re-compute the evaluation per sub-split.

Across all splits, SkillSmith is the clear winner. Even when faced with composite tasks where none of their parent tasks were used to construct the tasks in the meta-train set, SkillSmith achieves the

best Elo rating by a large margin. This attests to SkillSmith's ability to generalize beyond its training distribution.

I Source Task Sensitivity Results

Table 1: Task Performance Comparison (Rounded).

Task	True Source Tasks	Retrieval + LLM Selection	Random + LLM Selection	Direct Training
100_870	0.5649	0.6332	0.6219	0.7053
101_269	1.0197	1.0841	1.0682	1.3763
1_450	0.1834	0.2428	0.2320	0.3793
31_91	0.4099	0.4964	0.5707	0.7620
330_736	0.0999	0.0000	0.0000	0.0106
41_174	0.0999	0.3438	0.4096	0.6084
43_122	1.1463	0.9278	1.0524	1.6496
583_850	0.3461	0.4149	0.4055	0.5242
625_640	0.0000	0.1328	0.2653	0.1368
667_794	0.0507	0.0789	0.1176	0.0920
79_411	0.2708	0.0792	0.1140	0.0882
869_873	0.6440	0.6455	0.6450	0.7729
91_595	0.9210	0.9208	0.9221	1.0107
96_721	0.0030	0.0007	0.0000	0.0274
96_827	1.0050	1.0216	1.0210	1.1932

I.1 Ablation: Sensitivity to Source Task Selection

Having established the heuristic task selection pipeline in Section 4.4.2, which we refer to **Retrieval + LLM Selection**, we now investigate SkillSmith's sensitivity to the quality of these retrieved source sets by investigating **Random + LLM Selection** – a setting where we replace the initial retrieval phase with a random sampling of task pairs. We compare these two heuristic approaches against both an upper bound (using the true source tasks from the base SNI dataset) and a standard direct prefix-tuning baseline.

Our findings (Figure 10) demonstrate that SkillSmith maintains a substantial performance advantage over the direct training baseline, even when operating without ground-truth knowledge. While utilizing the true source tasks predictably yields the strongest overall performance, synthesizing tasks selected via heuristic pipelines still confers a significant empirical advantage over learning from scratch.

Though the Elo scores show that *Retrieval + LLM Selection* is only *slightly* better than its random counterpart, a deeper dive into the per-task performance (see above Table 1) reveals that whilst *Random + LLM Selection* can result in performance that is significantly worse than Direct Training (3/15 tasks), the performance of *Retrieval + LLM Selection* is superior to the baseline across all 15 tasks. This suggests future work on improving the retrieval mechanism.

J Rationale Prompt

Prompt 8: Rationale Prompt

```
Target Task: {task_input}
Selection: {selected_pair}
Please write a detailed rationale for why solving the selected
pair of tasks (independently or jointly) might result in useful
skills/knowledge transfer to the target task. If any or none of the
pair of tasks has transfer value for the target, then indicate so.
Do not mention the task IDs of the selected pair number in the final
rationale, just the task details. The rationale should start with
'Rationale: '.
```

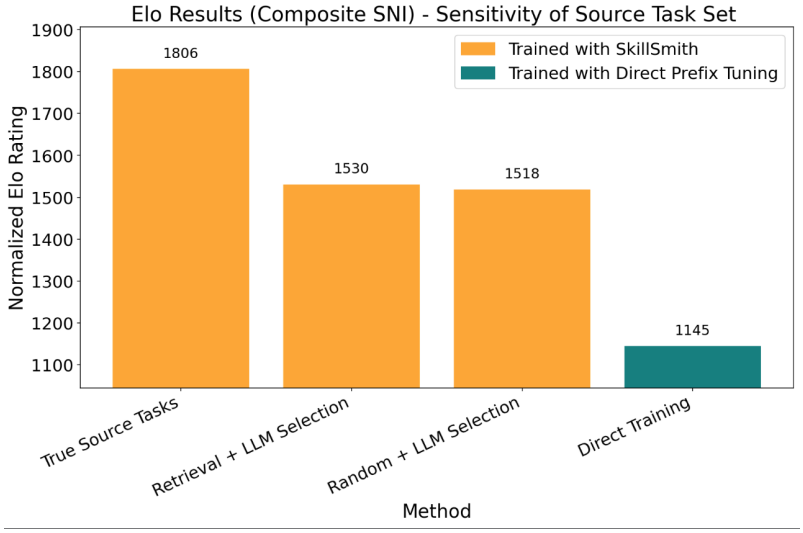

Figure 10: Our results demonstrate that even in the absence of ground-truth, synthesizing an intelligently selected set of source tasks can boost performance on target tasks. (Elo calculated over performance on 15 tasks)

K SkillSmith Training Details

For training SkillSmith, we use the official pre-trained Gemma 3-4B (Google, 2025b) as the augmented co-processor model. We use the Adam optimizer (Kingma & Ba, 2014) with learning rates $\{1e^{-3}, 1e^{-4}\}$ and batch sizes $\{32, 64, 128\}$. For experiments with Composite-SNI, we train for 10000 steps whilst we use 2000 steps when training for the base SNI tasks. To reduce overfitting, we introduce data augmentations during training where parts of the model input are occasionally dropped out. We drop-out KV-caches by setting the whole cache to zero. We explore KV dropout probabilities in the set $\{0.1, 0.293, 0.5\}$. During training, we allow SkillSmith to randomly sample the output length of the generated cache from the set $\{16, 32, 64, 128\}$, but during evaluation, we always output caches of length 32.

K.1 SkillSmith Training data example

Below is the full sequence of a single training example from the SNI Meta-Training set.

Prompt 9: Preamble Text

```
Your role is Cognitive Grafting.  Take the established root systems
(K-V prefixes) from the varied source tasks and graft them together
to support the growth of the new, novel downstream task.  Next is
information about the source tasks
```

Prompt 10: Source Text 1

```
</instruction> You are given a question on professional accounting.
You are also given 4 answer options (associated with ''A'', ''B'',
''C'', ''D''), out of which only one is correct.  You need to answer
the question by selecting the correct option.  You should only answer
with the choice letter, not the whole answer.  </instruction>"
```

Prompt 11: Source Text 2

</instruction> You are provided with a user review of a restaurant. Your task is to classify the given review into two categories: 1) positive, and 2) negative based on its content. </instruction>

Prompt 12: Combination Text

The preceding source tasks were combined using the following rationale:

<rationale> The target task involves analyzing a product review to determine its sentiment (positive or negative) and comparing that sentiment to a given numerical rating. The core required skill is text sentiment analysis to determine if the review implies a positive or negative experience.
The selected pair includes a task (Task 298) that requires classifying restaurant reviews into 'positive' or 'negative' categories. This directly aligns with the primary skill needed for the target task: interpreting subjective text to extract a binary sentiment. The model trained on this task learns to identify keywords, phrases, and overall tone that indicate satisfaction or dissatisfaction. This skill is crucial for evaluating whether the sentiment expressed in the review matches the numerical rating provided in the target task input.
The other task in the pair (Task 828) is an unrelated accounting multiple-choice question task, which does not offer much relevant skill transfer. However, Task 298 provides a strong, direct match for the core sentiment analysis component, making this pair a suitable choice for skill transfer to the target task. </rationale>

to create a new task, which you will be asked to solve.

Here are a few examples of the task you will be solving:

\n<example> In this task, you're given a review from Amazon and rating for the product on a scale of 1-5 based on the review. The rating means 1: extremely poor, 2: poor, 3: neutral, 4: good, 5: extremely good. Your task is to generate whether the rating matches the review or not. Answer with ``True'' if the sentence belongs to that section, otherwise answer with ``False''. \t [input]Review: As soon as I opened it, I didn't like it because the product looks old and used and rusted. I decided to give it a try but it took like half an hour to heat up. In addition to it, I don't like the curling iron. It doesn't curl at all. Straightener is good though. I am returning the item. \n Rating: 1 \n [target]True \n </example>

\n<example> In this task, you're given a review from Amazon and rating for the product on a scale of 1-5 based on the review. The rating means 1: extremely poor, 2: poor, 3: neutral, 4: good, 5: extremely good. Your task is to generate whether the rating matches the review or not. Answer with ``True'' if the sentence belongs to that section, otherwise answer with ``False''. \t [input]Review: I bought this awhile back. This mattress is quite firm so if you aren't looking for that, I would probably avoid this mattress. It expanded just as expected and was definitely a great mattress considering how cheap it was. \n Rating: 2 \n [target]False \n </example>

\n<example> In this task, you're given a review from Amazon and rating for the product on a scale of 1-5 based on the review. The rating means 1: extremely poor, 2: poor, 3: neutral, 4: good, 5: extremely good. Your task is to generate whether the rating matches the review or not. Answer with ``True'' if the sentence belongs to that section, otherwise answer with ``False''. \t [input]Review: I

```
purchased 2 bookmarks and received them yesterday.  Both screens don't
work.  I was about to return one, but now the other one said it is
ineligible for a return/replacement.  Not impressed.  \n Rating:  1 \n
[target]True \n </example>

Now please generate the K-V cache that can be used to solve this given
task when given new unseen test examples.
```

L RAW NLL SCORES

Table 2: Raw Negative Log-Likelihood (NLL) on Composite-SNI: Zero-Shot, ICL, and Weight-Space Initializations. (Lower is better).

Task ID	Parent Avg	Concat	LERP	SVD-Embed	SVD-Head	SVD-Seq	ICL-Init ZS	ICL	SkillSmith (ZS)
1_450	2.5800	2.6200	2.6284	2.5910	2.6128	2.6070	2.6480	0.4416	0.8708
31_91	2.1100	2.0188	1.9513	2.0885	2.1634	2.1673	2.1990	0.8295	0.6362
41_174	6.3605	6.4461	6.5336	9.0098	8.2121	8.0883	5.2100	0.8580	1.5269
43_122	2.1207	2.1464	2.1074	2.1104	2.0985	2.1047	2.1400	1.0430	1.7436
79_411	11.5230	11.2963	10.6155	13.8411	11.6429	11.9085	11.6800	1.9610	0.1968
91_595	1.8326	1.7886	1.7988	1.8573	1.7957	1.7943	1.8130	0.9901	1.0572
96_721	2.9530	3.0372	2.9455	3.3154	3.3144	3.3138	3.2960	0.1460	0.9202
96_827	2.6139	2.6399	2.6034	2.5904	2.5990	2.5983	2.6360	1.0010	1.6001
100_870	3.1702	3.1399	2.9852	3.2037	3.1432	3.1193	2.6080	0.7800	1.3528
101_269	3.0830	3.0477	3.0547	3.1226	3.1057	3.1053	3.1970	1.1936	1.8002
330_736	8.1049	7.6368	8.4033	9.1675	9.1498	9.0877	8.6360	1.5540	0.4654
583_850	4.0360	4.0434	3.6181	3.5950	3.5448	3.5423	3.3190	0.4490	3.0320
667_794	6.4471	6.3629	6.1754	6.3289	6.3341	6.3613	2.7670	0.2770	0.8779
869_873	1.2255	1.1658	0.9661	0.9933	0.9443	0.9409	0.9510	0.6847	0.6881
625_640	11.0340	10.9995	11.0538	11.1835	11.0508	10.9446	10.6950	1.6160	0.3733

Table 3: Raw Negative Log-Likelihood (NLL) on Composite-SNI: Performance Following Fine-Tuning. (Lower is better).

Task ID	Direct Tr.	Parent Avg	Concat	LERP	SVD-Embed	SVD-Head	SVD-Seq	ICL-Init FT	SkillSmith (True Sources)
1_450	0.3793	0.2950	0.2974	0.3356	0.2620	0.3073	0.3243	0.2920	0.1834
31_91	0.7620	0.8750	0.8139	0.7226	0.7982	0.8116	0.8391	0.7060	0.4099
41_174	0.6084	0.5411	0.7213	0.5339	0.4223	0.3242	0.3538	0.1760	0.0999
43_122	1.6496	1.6659	1.8626	1.6357	1.6257	1.6440	1.5945	1.2620	1.1463
79_411	0.0882	0.3195	0.6856	0.5381	0.3677	0.2070	0.3175	1.3250	0.2708
91_595	1.0107	1.0608	1.0162	1.0195	1.0456	1.0549	1.0707	1.0230	0.9210
96_721	0.0274	0.0309	0.0253	0.0235	0.0218	0.0334	0.0182	0.0180	0.0030
96_827	1.1932	1.2045	1.2621	1.2105	1.1520	1.2037	1.0914	1.1090	1.0050
100_870	0.7053	0.7109	0.7358	0.7552	0.7235	0.6411	0.6550	0.5860	0.5649
101_269	1.3763	1.3869	1.3669	1.3572	1.3612	1.3496	1.3547	1.2080	1.0197
330_736	0.0106	0.3231	0.2034	0.4816	0.2841	0.4134	0.3136	0.0470	0.0999
583_850	0.5242	0.3973	0.4993	0.4285	0.4285	0.4052	0.4366	0.4870	0.3461
667_794	0.0920	0.1029	0.1543	0.1376	0.1732	0.0821	0.1928	0.0820	0.0507
869_873	0.7729	0.7936	0.8243	0.7930	0.7996	0.8296	0.8101	0.7490	0.6440
625_640	0.1368	0.0887	0.0570	0.1229	0.1493	0.2155	0.0438	0.0450	0.0000

Table 4: Raw Negative Log-Likelihood (NLL) on Super-Natural Instructions (SNI): Zero-Shot, ICL, and Initializations. (Lower is better).

Task ID	Parent Avg	Concat	LERP	SVD Embed	SVD Head	SVD Seq	ICL-Init ZS	ICL	SkillSmith (Retrieved)	SkillSmith (CSNI-Pre)
57	7.5625	7.6265	8.0553	8.1522	8.2454	8.1967	8.0772	2.0198	0.6252	0.3985
133	3.5278	3.5995	3.5273	3.5164	3.4602	3.4947	3.4990	1.0906	0.3078	0.1992
171	2.0432	1.8293	1.8325	2.5682	1.9435	1.9337	1.9817	0.9786	1.2097	1.0847
204	11.3479	11.3024	11.1291	11.6804	11.6542	11.5744	7.2891	2.2348	0.3235	0.3519
393	4.2804	4.4789	3.4612	3.7875	3.5375	3.5528	3.8527	1.1926	1.9743	1.3607
549	4.6227	4.5129	4.7049	5.7759	5.1111	5.0962	5.1828	1.3415	1.6529	0.3667
691	8.1169	7.8365	6.7845	7.0051	7.5992	7.5915	7.5849	2.1524	5.5604	3.3689
731	10.6090	11.2829	10.2385	11.3719	11.8686	11.9030	9.3789	1.3625	0.5422	0.5549
780	6.8185	6.6995	5.5661	5.8363	6.4860	6.4141	6.4399	1.3462	3.4754	3.5575
843	10.0737	9.7736	10.1783	10.2499	10.2811	10.1875	10.2819	3.1265	3.6243	0.9736

Table 5: Raw Negative Log-Likelihood (NLL) on Super-Natural Instructions (SNI): Performance Following Downstream Fine-Tuning. (Lower is better).

Task ID	Direct Tr.	Parent Avg	Concat	LERP	SVD Embed	SVD Head	SVD Seq	ICL-Init FT	SkillSmith (Retrieved)	SkillSmith (CSNI-Pre)	SkillSmith (CSNI-Pre-Ret)
57	0.3478	0.3060	0.3401	0.2574	0.2994	0.2762	0.1753	0.3324	0.3349	0.1585	0.1869
133	0.0226	0.0078	0.0060	0.0122	0.0044	0.0089	0.0079	0.0027	0.0036	0.0114	0.0038
171	0.7604	0.6776	0.6442	0.6489	0.6899	0.6857	0.6999	0.6296	0.6875	0.6631	0.6366
204	0.1134	0.0350	0.0268	0.1236	0.0430	0.0249	0.0121	0.0449	0.0395	0.0300	0.0435
393	0.2134	0.1525	0.1725	0.1396	0.1526	0.1646	0.1606	0.1401	0.1441	0.1382	0.1696
549	0.2405	0.1750	0.1533	0.1945	0.1280	0.1508	0.1967	0.1079	0.1726	0.1931	0.1419
691	0.0658	0.0364	0.0193	0.0063	0.0245	0.0014	0.0292	0.0284	0.0239	0.0208	0.0244
731	0.0459	0.0412	0.0159	0.0517	0.0778	0.0386	0.0167	0.0005	0.0135	0.0151	0.0398
780	0.5251	0.2547	0.2646	0.2866	0.3437	0.2345	0.2600	0.2881	0.2966	0.3003	0.2838
843	0.0161	0.1100	0.1541	0.1849	0.1344	0.1106	0.0465	0.0468	0.0312	0.0040	0.0251

Table 6: Raw Negative Log-Likelihood (NLL) on MMLU-ProX: Zero-Shot, ICL, and Initializations. (Lower is better).

Language-Subject Pair	Parent Avg	Concat	LERP	SVD Embed	SVD Head	SVD Seq	ICL-Init ZS	ICL	MMLU SkillSmith	CSNI MMLU SkillSmith
afrikaans_history	1.1155	1.0945	1.1137	5.2073	3.0015	2.9739	2.9060	2.6226	1.3311	1.0582
indonesian_law	1.2312	1.2188	1.2161	1.7624	2.9021	2.8879	2.8020	2.9910	1.2113	1.1263
spanish_law	1.2199	1.1979	1.2045	2.0386	3.1152	3.1073	3.0310	3.2074	1.1729	1.1182
wolof_health	1.1030	1.0880	1.0946	5.0859	3.3385	3.2990	3.2010	3.1473	1.2284	1.1037
wolof_math	1.0863	1.0645	1.0809	1.7081	3.8409	3.8136	3.7020	3.3445	1.0032	1.0157
zulu_physics	1.1585	1.1928	1.1272	8.1525	3.5368	3.4717	3.3830	3.4179	1.0423	1.0699

Table 7: Raw Negative Log-Likelihood (NLL) on MMLU-ProX: Performance Following Downstream Fine-Tuning. (Lower is better).

Language-Subject Pair	Direct Tr.	Parent Avg	Concat	LERP	SVD Embed	SVD Head	SVD Seq	ICL-Init FT	MMLU SkillSmith	CSNI MMLU SkillSmith
afrikaans_history	1.1040	1.0895	1.0793	1.0790	1.0698	1.0878	1.1410	1.1050	1.0815	0.9883
indonesian_law	1.0886	1.1589	1.1520	1.1514	1.1887	1.1394	1.1475	1.1650	1.0376	1.0208
spanish_law	1.1243	1.1327	1.1010	1.1060	1.1275	1.1168	1.1267	1.1200	1.0433	1.0244
wolof_health	1.1524	1.0989	1.1016	1.1033	1.1142	1.0924	1.1066	1.1320	1.0980	1.0693
wolof_math	1.1428	1.0725	1.0489	1.0790	1.1485	1.1432	1.1409	1.1310	0.9859	0.9976
zulu_physics	1.1410	1.0889	1.0937	1.0754	1.1509	1.1278	1.1619	1.1340	1.0213	1.0405