FROM WEIGHTS TO WORDS: META-LEARNING SKILL WRITERS WITH PARAMETER–TEXT ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Model (LLM) agents increasingly store procedural knowledge in natural-language skills, written by a Writer model and executed by a Reader model (Wang et al., 2024; Ma et al., 2026; Ni et al., 2026). Most methods optimize a skill or its construction prompt for a fixed Reader, without learning a Writer initialization whose construction process can rapidly adapt to new task–Reader pairs(Yuksekgonul et al., 2025; Agrawal et al., 2026; Yang et al., 2026). Yet a skill is only as useful as the Reader that executes it. we introduce METASKILL, which meta-learns a Writer initialization for coupled parameter–text adaptation. Reader feedback emporarily specializes the Writer, and text refinement externalizes that specialization into a persistent skill. Across language, reasoning, and interactive benchmarks, this adaptation is far more effective from the meta-learned initialization than from the original or a multitask-trained Writer, and both stages contribute. METASKILL also matches the strongest text-space optimizer on non-interactive tasks at a lower token cost and adapts to unseen Reader families. Skills shaped for one Reader serve others less well, tying skill utility to the task–Reader pair. These results show that meta-learning an adaptable Writer enables effective and token-efficient skill acquisition for new tasks and readers.

1 INTRODUCTION

Natural-language skills give Large Language Model (LLM) agents an explicit and reusable form of procedural knowledge. A skill can encode a decision rule, a decomposition of a computation, a tool-use strategy, or an interaction protocol, and once written it can be inspected, revised, and reused without changing the model that executes it (Wang et al., 2024; Ma et al., 2026; Ni et al., 2026). Recent work therefore constructs and refines skills automatically from demonstrations, trajectories, and execution feedback (Ma et al., 2026; Ni et al., 2026; Yang et al., 2026; Zhang et al., 2026).

Most of this work treats a skill as a property of the task, to be inferred once and reused on future instances. Yet a skill is useful only through the model that executes it, which we call the Reader, as distinct from the Writer that composes it. Different Readers may need different decompositions, fail at different steps, or require different levels of abstraction on the same task. Our first principle is therefore that *procedural knowledge is relational*. The utility of a skill is defined for a *task–Reader pair*, not for the task alone.

SkillTTA and SKILLER capture part of this dependence by optimizing skills, or the prompt that writes them, against a fixed target model (Wang et al., 2026; Dang et al., 2026), and instruction and skill optimizers more broadly refine text from critiques, scores, or execution traces (Honovich et al., 2023; Pryzant et al., 2023; Yuksekgonul et al., 2025; Agrawal et al., 2026; Yang et al., 2026). These methods tailor skills to one Reader, and much of that effort must be repeated whenever the task or Reader changes. Our second principle is that *adaptation itself should be learned*, so that Reader specificity becomes a transferable property of the process that writes skills rather than only of the skills it writes. Prior work has learned parts of such a process, including transferable prompt initializations (Choi et al., 2025), LLM optimizers (Xu et al., 2025), instruction engineers trained from solver feedback (Qi et al., 2025), and Test-Time Adaptation (TTA) policies (Lou et al., 2026). We instead meta-learn the parameters of the Writer itself as an initialization for adapting to each new task–Reader pair.

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.

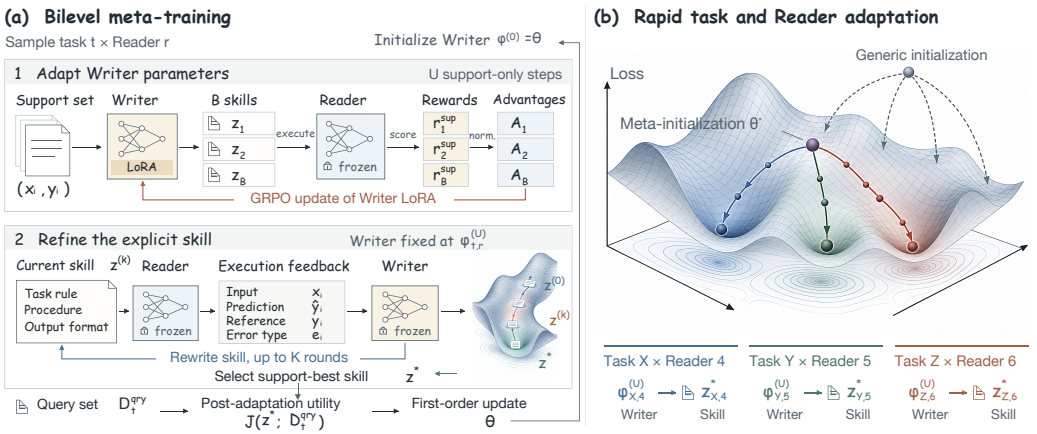

Figure 1: **Overview of METASKILL.** For a new task–Reader pair, execution feedback first specializes the Writer's parameters and then refines the skill in text space. The specialized parameters are discarded, and only the skill is retained for the frozen Reader. Meta-training optimizes the shared Writer initialization for the utility of the skill after both stages.

These principles lead to our central question. *Can a Writer learn a starting point for skill construction that specializes quickly to a new task–Reader pair from limited execution feedback?* Such a starting point need not write the best skill before adaptation. What matters is how far limited feedback can take it, so we optimize the Writer for where adaptation leads rather than where it starts.

We introduce METASKILL, which learns such a starting point for a coupled adaptation process (Figure 1). For a new task–Reader pair, adaptation moves from weights to words. The Reader first executes candidate skills on a few support examples, and a few policy-gradient updates on the resulting rewards specialize the Writer's parameters toward procedures this Reader executes well. The specialized Writer then refines the skill in text space against the same support examples. Its specialized parameters are discarded afterward, while the refined skill is retained for the frozen Reader. Parameter adaptation thus changes how the Writer uses feedback, and text refinement determines which procedural knowledge persists. Meta-training optimizes the shared initialization for the query utility of the final skill.

We evaluate METASKILL on 322 held-out adaptation episodes from five suites covering instruction following, multi-step reasoning, compositional generalization, relational reasoning, and embodied interaction. Under the same test-time procedure, the meta-learned initialization reaches a macro-average of 62.4, whereas the original Writer reaches 41.0 and a Writer trained on the same tasks with a pooled multitask objective reaches 39.9. Disabling the parameter updates while keeping the same sampling and refinement lowers the score to 54.6, and either stage alone reaches at most 53.1. On the four non-interactive suites, METASKILL matches TextGrad, the strongest baseline there, at 63.0 while using fewer skill-construction tokens. On two Reader families unseen during meta-training, it gains 9.6 and 9.9 points over its initial skills.

Our contributions are threefold.

- We formulate skill utility as relational to the task–Reader pair and empirically show that skills shaped by one Reader's feedback serve another Reader less well.
- We introduce METASKILL, which meta-learns a Writer initialization for post-adaptation utility by coupling temporary parameter adaptation with persistent text refinement.
- Across five benchmark suites, several Writer and Reader scales, and unseen Reader families, we show that the meta-learned initialization, rather than additional test-time computation, makes adaptation effective, and that both stages contribute.

2 RELATED WORK

Explicit skills and procedural knowledge. LLM agents increasingly externalize reusable knowledge as natural-language or executable skills (Wang et al., 2024; Ma et al., 2026; Ni et al., 2026; Wang et al., 2026; Yang et al., 2026). This line of work is closely related to in-context and instruction learning, where models infer task behavior from demonstrations (Brown et al., 2020; Honovich

et al., 2023; Chen et al., 2024) and where theoretical analyses explain how such behavior can emerge from context (Xie et al., 2022; Akyürek et al., 2023; Von Oswald et al., 2023; Shen et al., 2024). Recent systems make the induced procedure explicit. SkillGen and Trace2Skill distill skills from demonstrations and trajectories (Ma et al., 2026; Ni et al., 2026), and SkillComposer learns operations for creating, improving, and merging skills (Zhang et al., 2026). We likewise treat skills as persistent procedural artifacts, but ask how the Writer that composes them should adapt to the Reader that executes them.

Execution-grounded optimization of prompts and skills. Feedback-driven methods improve outputs through critique, reflection, and learned self-correction (Madaan et al., 2023; Shinn et al., 2023; Kumar et al., 2025), and prompt optimizers refine instructions from critiques, scores, demonstrations, textual gradients, or execution traces (Pryzant et al., 2023; Yang et al., 2024; Opsahl-Ong et al., 2024; Yuksekgonul et al., 2025; Agrawal et al., 2026). SkillOpt applies this idea to skill documents for a frozen executor (Yang et al., 2026), AIO trains an instruction engineer from solver feedback (Qi et al., 2025), and SkillTTA and SKILLER make executor dependence explicit by synthesizing or optimizing skills for a fixed target model (Wang et al., 2026; Dang et al., 2026). These methods optimize a skill, a prompt, or an instruction writer against a given solver. We instead meta-learn the Writer's parameters as an initialization explicitly optimized so that Reader-grounded parameter specialization improves the skill produced by subsequent text refinement.

Meta-learned and test-time adaptation. Gradient-based meta-learning learns initializations that specialize quickly from limited task data (Finn et al., 2017; Nichol et al., 2018). Recent work carries this idea into language space by learning a transferable prompt initialization (Choi et al., 2025), improving LLM optimizers (Xu et al., 2025), learning a TTA policy (Lou et al., 2026), or meta-optimizing the policy that synthesizes skills (Wang et al., 2026). Knowledge can also move between words and weights. Context distillation moves knowledge from prompts into weights (Snell et al., 2022), and test-time training adapts weights to each new problem and predicts with the adapted model (Sun et al., 2020; Akyürek et al., 2025). In both cases the weights are the final carrier of knowledge. METASKILL instead ends in words. It meta-learns the parameter initialization of the Writer for post-adaptation utility, adapts those parameters only temporarily for each task–Reader pair, and keeps the result as a skill that the frozen Reader executes.

3 PROBLEM FORMULATION

A task t comes with an instruction c_t, a labeled support set $\mathcal{D}_t^{\text{sup}}$, and a disjoint query set $\mathcal{D}_t^{\text{qry}}$. A Reader M_m is a frozen model that solves task inputs while following a natural-language skill z. The Reader also sees up to three support demonstrations, never including the example being scored, and it decodes greedily, so we write its output as $M_m(x; c_t, z)$. The utility of z on a set of examples $\mathcal{S}$ is its mean task score,

$$J_{t,m}(z; \mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{(x,y)\in\mathcal{S}} s_t\big(M_m(x; c_t, z), y\big), \tag{1}$$

where $s_t \in [0, 1]$ is a task-specific score. For interactive tasks, x is an environment episode, $M_m(x; c_t, z)$ is the resulting trajectory, and s_t indicates success.

Utility is thus defined for a task–Reader pair (t, m), and the best skill for a task can change with the Reader. A Writer must therefore specialize to each pair from the support set and the Reader's executions on it alone, and the query set never enters skill construction.

The Writer is a frozen LLM with trainable Low-Rank Adaptation (LoRA) parameters θ (Hu et al., 2022). It defines a policy π_θ over skills, conditioned on c_t, the support set, a textual identifier d_m of the Reader, and, when revising, a current skill and its execution feedback. Skill construction for a pair composes two stages,

$$\phi_{t,m} = \mathcal{T}_{\text{grad}}\big(\theta; \mathcal{D}_t^{\text{sup}}, M_m\big), \qquad z^\star_{t,m} = \mathcal{T}_{\text{text}}\big(\phi_{t,m}; \mathcal{D}_t^{\text{sup}}, M_m\big), \tag{2}$$

which we abbreviate as $\mathcal{T} = \mathcal{T}_{\text{text}} \circ \mathcal{T}_{\text{grad}}$. Parameter adaptation $\mathcal{T}_{\text{grad}}$ updates the Writer from Reader feedback, and text refinement $\mathcal{T}_{\text{text}}$ uses the updated Writer to revise a skill. The pair-specific parameters $\phi_{t,m}$ are discarded after construction, whereas the skill $z^\star_{t,m}$ is kept for the frozen Reader.

Because both stages sample from the Writer, $\mathcal{T}(\theta; \mathcal{D}_t^{\text{sup}}, M_m)$ is a random skill. Meta-training chooses θ to maximize the expected query utility after adaptation,

$$\max_{\theta} \; \mathbb{E}_{(t,m)\sim\mathcal{P}} \, \mathbb{E}_{z\sim\mathcal{T}(\theta;\mathcal{D}_t^{\text{sup}},M_m)} \big[J_{t,m}(z; \mathcal{D}_t^{\text{qry}}) \big], \tag{3}$$

where $\mathcal{P}$ is a distribution over training tasks and Readers. Multitask training would instead maximize the query utility of a skill sampled from π_θ before any adaptation, whereas Equation (3) rewards the initialization for where adaptation leads rather than where it starts.

4 Meta-Learning the Skill Writer

Sections 4.1 and 4.2 instantiate the two stages of Equation (2) for a single pair, and Section 4.3 meta-learns their shared initialization, as summarized in Algorithm 1 (Section A).

4.1 Reader-Grounded Parameter Adaptation

$\mathcal{T}_{\text{grad}}$ adapts the Writer's parameters to the pair with U rounds of policy-gradient updates on Reader executions, starting from $\phi^{(0)} = \theta$. In round u, the Writer samples G candidate skills $z_1, \ldots, z_G$ from $\pi_{\phi^{(u)}}$, conditioned on c_t, d_m, the support set, and the best candidate from the previous round together with its execution feedback, both empty in the first round. The Reader executes every candidate on the support set, which gives the rewards and group-normalized advantages

$$r_g = J_{t,m}\big(z_g; \mathcal{D}_t^{\text{sup}}\big), \qquad A_g = \frac{r_g - \text{mean}(r_{1:G})}{\text{std}(r_{1:G}) + \epsilon}. \tag{4}$$

The Writer is updated on this group with the Group Relative Policy Optimization (GRPO) objective (Shao et al., 2024), which combines a clipped importance-weighted policy gradient with a Kullback–Leibler (KL) penalty toward the episode-start Writer π_θ, for two to four AdamW steps with learning rate α. After U rounds, $\phi_{t,m} = \phi^{(U)}$. We use $U = 2$ and $G = 8$ throughout, and Section B gives the full objective.

The candidates sampled in this stage supply only the learning signal and are not passed to text refinement. Parameter adaptation therefore acts on the Writer rather than on any particular skill, shifting its proposals toward procedures that this Reader executes well. Section 5.2 isolates this effect from the extra sampling with an $\alpha = 0$ control.

4.2 Execution-Grounded Text Refinement

$\mathcal{T}_{\text{text}}$ keeps the Writer fixed at $\phi_{t,m}$ and revises a skill in text. The adapted Writer first samples an initial skill $z^{(0)} \sim \pi_{\phi_{t,m}}(\cdot \mid c_t, d_m, \mathcal{D}_t^{\text{sup}})$. In round k, the Reader executes the incumbent $z^{(k)}$ on the support set, and the feedback $f^{(k)}$ lists the failed inputs with the Reader's outputs and scores, plus a diagnosis rendered from the evaluator's failure record when available. The Writer then proposes a complete replacement

$$\tilde{z}^{(k+1)} \sim \pi_{\phi_{t,m}}\big(\cdot \mid c_t, d_m, \mathcal{D}_t^{\text{sup}}, z^{(k)}, f^{(k)}\big), \tag{5}$$

and the next incumbent is the best of the incumbent, the proposal, and the empty skill $\varnothing$ on the support set,

$$z^{(k+1)} = \operatorname*{arg\,max}_{z\in\{z^{(k)}, \tilde{z}^{(k+1)}, \varnothing\}} J_{t,m}\big(z; \mathcal{D}_t^{\text{sup}}\big). \tag{6}$$

Refinement runs for at most $K = 5$ rounds and stops early once the support score converges, and the last incumbent is the output skill $z^\star_{t,m}$. By construction, support utility never decreases across rounds and never falls below that of the empty skill, although query utility carries no such guarantee.

4.3 Meta-Learning the Coupled Adaptation

Meta-training optimizes Equation (3) through both stages. For a training pair, we first run $\mathcal{T}_{\text{grad}}$ to obtain $\phi_{t,m}$. From this fixed Writer we sample $N = 4$ independent refinement trajectories $\tau_i = (z_i^{(0)}, \ldots, z_i^{(K)})$, make every selection in them with Equation (6) on the support set, and only then

score each incumbent on the query set, $q_{ik} = J_{t,m}(z_i^{(k)}; \mathcal{D}_t^{\text{qry}})$. A trajectory that stops early repeats its last incumbent. Each trajectory receives the reward

$$R_i = q_{iK} + \lambda_g \, h\big(z_i^{(0)}, z_i^{(K)}\big)\big[q_{iK} - q_{i0}\big]_+ + \lambda_p \sum_{k=1}^{K} h\big(z_i^{(k-1)}, z_i^{(k)}\big)\big[q_{ik} - q_{i,k-1}\big]_+, \tag{7}$$

where $[\cdot]_+ = \max(\cdot, 0)$. The first term is the final query utility, the quantity we report in evaluation, and the two bonuses reward improvement over the whole trajectory and within individual rounds. The indicator $h \in \{0, 1\}$ withholds a bonus when a revision leaves the first 80 characters of the skill unchanged, a cheap proxy for near-duplicate revisions.

With the leave-one-out advantage $A_i = R_i - \frac{1}{N-1}\sum_{j \neq i} R_j$, the meta-loss at Writer parameters ϕ is

$$\mathcal{L}_{\text{meta}}(\phi) = -\frac{1}{N}\sum_{i=1}^{N} A_i \sum_{a \in \tau_i} \sum_{\ell=1}^{|a|} \log \pi_\phi\big(a_\ell \mid H_a, a_{<\ell}\big), \tag{8}$$

where a ranges over every Writer output in τ_i, including proposals from Equation (5) that Equation (6) rejects, and H_a is the prompt that produced a. Reader outputs affect the loss only through the rewards. Following first-order meta-learning (Finn et al., 2017; Nichol et al., 2018), we evaluate the gradient at $\phi_{t,m}$ and apply it to θ without differentiating through the inner updates,

$$g_{\text{meta}} = \sum_{(t,m)\in\mathcal{E}} w_{t,m} \, \text{clip}_1\Big(\nabla_\phi \mathcal{L}_{\text{meta}}(\phi)\big|_{\phi=\phi_{t,m}}\Big), \qquad \theta \leftarrow \text{AdamW}_\beta\big(\theta, g_{\text{meta}}\big), \tag{9}$$

where $\mathcal{E}$ is a batch of training pairs, $w_{t,m}$ balances benchmarks and Readers, clip_1 rescales a gradient to norm at most one, and AdamW_β is one AdamW step with learning rate β (Sections A and B). Because the loss is computed after parameter adaptation and covers the whole refinement trajectory, the learned θ serves both as the starting point of $\mathcal{T}_{\text{grad}}$ and, once adapted, as the refinement policy. At deployment, a new pair receives the same U adaptation rounds followed by a single refinement trajectory, and only the resulting skill is kept.

5 Experiments

5.1 Experimental Setup

Tasks and metrics. We evaluate on 322 held-out adaptation episodes from five suites. Super-NaturalInstructions (SuperNI) contributes 150 episodes from 50 task IDs (Wang et al., 2022), BIG-Bench Hard (BBH) 28 episodes from seven task IDs (Suzgun et al., 2023), SCAN 40 template-disjoint episodes (Lake & Baroni, 2018), CLUTRR 60 pattern-disjoint episodes (Sinha et al., 2019), and ALFWorld 44 skill-construction episodes, each evaluated on two held-out environment episodes (Shridhar et al., 2021). The metrics are normalized token F1 for SuperNI, exact match for BBH, exact action-sequence match for SCAN, relation accuracy for CLUTRR, and task success for ALFWorld, all reported as percentages. The macro-average weights the five suites equally, and we call SuperNI, BBH, SCAN, and CLUTRR the non-interactive suites.

Protocol. In each evaluation episode, support data drive both adaptation stages and every selection, and query data serve only to score the final skill. Evaluation episodes come from splits disjoint from those used for meta-training and checkpoint selection. Unless stated otherwise, the Writer is Qwen2.5-14B, the Reader is a frozen Qwen2.5-14B, and refinement runs for at most five rounds. The Writer is meta-trained with Qwen-, Gemma-, and Ministral-family Readers. Results for trained Writers are means over five training runs, and every condition repeats skill construction five times with independent Writer sampling while Reader decoding stays deterministic. Values after $\pm$ are standard deviations over runs. A very small number of ALFWorld executions triggered an environment-side bug for particular model responses; these invalid executions were identified and excluded before aggregation (Section C).

Baselines. We compare with GEPA (Agrawal et al., 2026), SkillOpt (Yang et al., 2026), and TextGrad (Yuksekgonul et al., 2025). Each uses the original Qwen2.5-14B as its optimizer, the

Table 1: **Direct solving, ablations, and comparisons across Writers and optimizers.** Scores are percentages, and SNI, CLU, and AW denote SuperNI, CLUTRR, and ALFWorld. Avg. is the macro-average over the five suites, and Avg.† excludes ALFWorld. Blocks B–D use the Qwen2.5-14B Reader. In Block B, METASKILL and the first four rows start from the meta-learned initialization, the $\alpha = 0$ row runs both stages without parameter updates, and the next two rows apply both stages to the original Writer and to a Writer trained with a pooled multitask objective. Init. is the average of the initial skills before adaptation, and Δ is Avg. minus Init. Values after $\pm$ are standard deviations over five runs, and bold marks the best average in each block.

A. Direct solving without a skill

Model	SNI	BBH	SCAN	CLU	AW	Avg.
DeepSeek-R1-7B	4.4	19.0	0.6	25.4	9.1	11.7
Qwen2.5-7B	58.7	57.1	1.8	19.2	23.6	32.1
Gemma-4-12B	60.9	71.4	1.1	42.1	2.3	35.6
Qwen2.5-14B	62.5	65.5	2.5	34.0	54.5	43.8
Ministral-14B	49.8	61.3	0.9	18.8	18.2	29.8
Qwen3-30B-A3B	59.0	72.0	3.1	30.8	36.2	40.2
Gemini-3.1-Pro	69.0	79.0	6.7	50.0	51.8	**51.3**

B. Ablations of initialization and adaptation

Configuration	SNI	BBH	SCAN	CLU	AW	Avg.
No adaptation	60.1	51.0	8.3	35.2	59.9	42.9
Text refinement only	62.7	62.3	31.2	42.7	62.5	52.3
Parameter adaptation only	61.2	51.7	48.1	40.5	64.2	53.1
Both stages, $\alpha = 0$	61.3	64.9	42.3	41.2	63.1	54.6
Original Writer, both	59.2	37.8	17.6	34.6	56.0	41.0
Multitask Writer, both	63.7	61.2	3.5	40.4	30.6	39.9
METASKILL	66.4	63.3	69.3	52.9	60.4	**62.4**

	SNI	BBH	SCAN	CLU	AW	Avg.	Avg.†	Init.	Δ
C. Off-the-shelf Writers with text refinement									
DeepSeek-R1-7B	58.1	66.1	8.3	35.8	36.1	40.9	42.1	50.1	−9.2
Qwen2.5-7B	59.2	62.5	4.9	34.2	35.0	39.2	40.2	38.8	+0.4
Gemma-4-12B	61.8	66.7	11.6	39.2	41.7	44.2	44.8	45.1	−0.9
Qwen2.5-14B	62.4	66.0	10.5	39.0	44.0	44.4	44.5	43.2	+1.2
Ministral-14B	60.3	64.3	9.1	35.0	32.7	40.3	42.2	40.5	−0.2
Qwen3-30B-A3B	59.8	69.6	13.7	34.2	44.0	44.3	44.3	43.1	+1.2
Gemini-3.1-Pro	63.2	67.9	15.4	57.2	58.3	**52.4**	**50.9**	45.9	+6.5
D. Comparison with skill and prompt optimizers									
GEPA	66.6	70.2	63.8	45.4	46.4	58.5 ± 5.5	61.5	–	–
SkillOpt	67.4	68.5	53.1	43.3	37.5	54.0 ± 1.2	58.1	–	–
TextGrad	59.0	64.3	80.0	48.8	37.5	57.9 ± 2.9	**63.0**	–	–
METASKILL-4B	50.7	52.2	27.8	27.7	31.1	37.9 ± 2.6	39.6	36.5 ± 6.4	+1.4
METASKILL-7B	64.1	60.9	64.0	42.1	53.8	57.0 ± 6.2	57.8	39.0 ± 6.2	+18.0
METASKILL-14B	66.4	63.3	69.3	52.9	60.4	**62.4 ± 1.9**	**63.0**	42.9 ± 4.3	+19.5
E. Gemini-3.1-Pro as the Writer, refined for each Reader									
Qwen2.5-7B	61.1	57.1	7.2	20.8	31.0	35.4 ± 7.2	36.6	31.8 ± 4.4	+3.6
Gemma-4-12B	66.7	72.6	14.3	47.1	49.2	50.0 ± 1.6	50.2	45.0 ± 3.1	+5.0
Qwen2.5-14B	63.2	67.9	15.4	57.2	58.3	52.4 ± 4.0	50.9	45.9 ± 2.6	+6.5
Ministral-14B	57.3	62.5	9.6	48.7	51.8	46.0 ± 1.2	44.5	38.7 ± 3.7	+7.3
Gemini-3.1-Pro	70.4	76.8	18.8	54.2	52.5	**54.5 ± 2.5**	**55.1**	49.1 ± 1.8	+5.4

same frozen Reader, and the same support and query sets, and makes every optimization and selection decision on support data (Section B). Beyond these comparisons, Sections 5.3 to 5.6 vary Writer and Reader capacity, test Reader families unseen during meta-training, and examine skill revisions and parameter geometry.

5.2 THE META-LEARNED INITIALIZATION MAKES ADAPTATION EFFECTIVE

Ablations. Block B of Table 1 separates the initialization from the two stages. Meta-training barely changes the quality of the Writer's initial skills, which average 42.9 against 43.2 for the original Writer (Block C). After the same adaptation, however, the meta-learned initialization reaches 62.4, whereas the original Writer reaches 41.0 and a Writer trained with a pooled multitask objective reaches 39.9. The advantage of the meta-learned initialization therefore lies in where adaptation leads rather than where it starts. Both stages contribute. Text refinement alone reaches 52.3 and parameter adaptation alone 53.1. With $\alpha = 0$, which keeps the sampling of parameter adaptation but disables its updates, the two stages together reach only 54.6, so most of the gain over text refinement alone comes from the updates rather than from the additional samples. Yet the benefit of parameter adaptation depends on the initialization. Starting from the original Writer, adding it to text refinement lowers the average from 44.4 (Block C) to 41.0. The effects of the two stages also vary across suites (Figure 2b). The parameter updates matter most on SCAN, which rises from 42.3 with $\alpha = 0$ to 69.3, but slightly lower BBH and ALFWorld. On ALFWorld, text refinement alone, parameter adaptation alone, and the $\alpha = 0$ control all score above the full method (Section B.7).

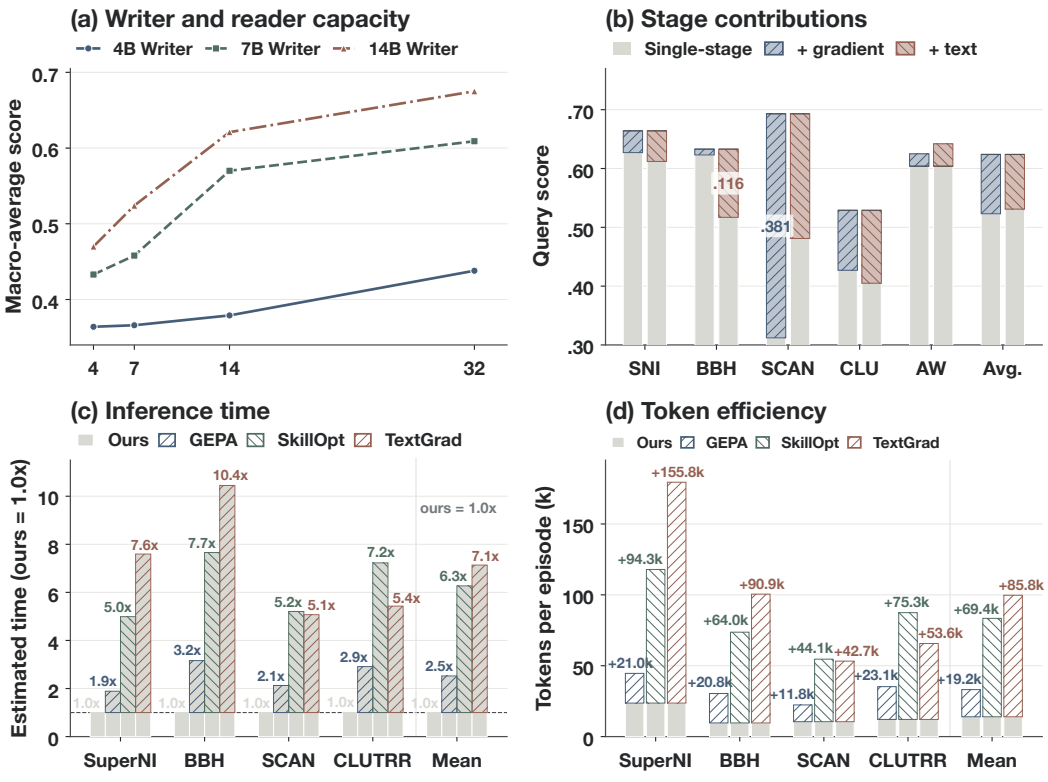

Figure 2: Adaptation, capacity, and construction cost. **(a)** Macro-average across Writer and Reader capacities. **(b)** Task-level effects of parameter and text adaptation. **(c)** Skill-construction time relative to METASKILL under equal token-throughput, including wall-clock costs of backward passes and optimizer updates. **(d)** Per-episode token consumption; hatching denotes additional baseline cost.

Generic writers. Blocks A, C, and E show that off-the-shelf Writers exploit Reader feedback unreliably. Their results largely follow model strength, as the four strongest solvers in Block A are also the four strongest Writers in Block C, in the same order. Refinement itself, however, changes the average by at most 1.2 points for five of the seven Writers in Block C. DeepSeek-R1-7B, the weakest solver, writes the best initial skills but loses 9.2 points during refinement, and even Gemini-3.1-Pro, the strongest Writer, gains only 3.6 to 7.3 points when refining for each Reader in Block E. On the Qwen2.5-14B Reader, Gemini-3.1-Pro reaches 52.4, ten points below METASKILL.

Comparison with optimizers. Block D compares METASKILL with three optimizers that refine skills or prompts in text for the same Reader. For a strictly matched comparison, we report Avg.† over the four non-interactive suites, where METASKILL matches TextGrad at 63.0, is comparable to GEPA (61.5), and exceeds SkillOpt (58.1). No method leads on every suite. TextGrad leads on SCAN, GEPA on BBH, and SkillOpt on SuperNI, whereas METASKILL leads on CLUTRR and ALFWorld. Over all five suites, METASKILL averages 62.4 ± 1.9 against 58.5 ± 5.5 for GEPA, but most of this margin comes from ALFWorld, where all three optimizers fall below the Reader's direct-solving score of 54.5. METASKILL also constructs skills with fewer tokens than the optimizers (Figure 2d), although this count excludes the backward passes of parameter adaptation and the one-time cost of meta-training.

Table 2: **Writer and Reader capacity.** Each row reports scores after meta-learned adaptation for one combination of Writer and Reader sizes, with abbreviations as in Table 1. Scale and model family vary together (Table A1), and bold marks the best average.

Writer	Reader	SNI	BBH	SCAN	CLU	AW	Avg.
4B	4B	49.2	50.3	26.9	26.5	29.3	36.4
4B	7B	49.2	51.1	26.2	26.9	29.4	36.6
4B	14B	50.7	52.2	27.8	27.7	31.1	37.9
4B	32B	51.0	58.3	34.0	36.4	39.5	43.8
7B	4B	50.2	48.3	37.4	36.1	44.4	43.3
7B	7B	52.1	51.6	45.7	36.9	42.9	45.8
7B	14B	64.1	60.9	64.0	42.1	53.8	57.0
7B	32B	67.3	61.6	75.6	44.2	55.8	60.9
14B	4B	57.2	52.3	40.1	40.8	44.7	47.0
14B	7B	62.0	58.3	43.8	43.1	54.6	52.4
14B	14B	66.4	63.3	69.3	52.9	60.4	62.4
14B	32B	68.7	70.2	81.2	48.1	69.2	**67.5**

Table 4: **Text refinement turns Reader failures into explicit rules.** Each row shows two refinement rounds (R1 and R2) for one task, together with the Reader failure that prompted each revision. Green text marks added rules, and red strikethrough marks removed wording. Arrows trace support scores round by round and query scores from the initial to the final skill, and Section F gives the complete cases.

Task	Reader feedback	Skill revision	Scores
BBH	**R1:** Nested parentheses cause errors. **R2:** Remaining errors come from incorrect Boolean precedence.	**R1:** **+ Resolve the innermost parentheses first.** **R2:** **+ Apply Boolean operators according to precedence.**	**Support:** .333 → .500 → .667 **Query:** .333 → .500
SuperNI	**R1:** The skill is descriptive but lacks an explicit decision rule. **R2:** Outputs remain verbose and occasionally violate the label format.	**R1:** ~~Describe the question and answer.~~ **+ Decide whether the answer directly resolves the question.** **R2:** **+ Output exactly one label: `yes` or `no`.**	**Support:** .50 → .625 → .75 **Query:** .50 → .625
CLUTRR	**R1:** The Reader confuses direct kinship with in-law relations. **R2:** Errors persist when spouse edges occur inside longer relation chains.	**R1:** **+ Check explicit spouse relations before assigning an in-law relation.** **R2:** **+ Resolve direct parent-child links before propagating affine relations.**	**Support:** .50 → .50 → .75 **Query:** .00 → .25 → .50

5.3 Writer and Reader Capacities Interact

Gains from adaptation depend strongly on the Writer. On the Qwen2.5-14B Reader, they amount to 1.4 points for the 4B Writer but 18.0 and 19.5 points for the 7B and 14B Writers (Block D of Table 1). Table 2 varies the Writer and the Reader jointly, with scale and model family changing together. The two capacities interact rather than contribute additively. Moving from the 4B to the 32B Reader raises the average by 7.4 points with the 4B Writer, 17.6 points with the 7B Writer, and 20.5 points with the 14B Writer, so a stronger Reader helps most when paired with a stronger Writer. The interaction also differs across suites. With the 14B Writer, the 32B Reader improves SCAN from 69.3 to 81.2 and ALFWorld from 60.4 to 69.2 over the 14B Reader but lowers CLUTRR from 52.9 to 48.1. Figure 2a summarizes the matrix.

Table 3: **Transfer to unseen Reader families.** Blocks A and B use a Writer meta-trained with Qwen-family Readers only. Avg.† excludes ALFWorld as in Table 1, and Section E.2 gives suite-level results. Block C crosses the Reader whose feedback shaped a skill (rows) with the Reader that executes it (columns). Bold marks the best entry in each column of Blocks B and C.

A. Adaptation on seen and unseen Readers

Reader	Status	Before	After	Δ
Qwen2.5-14B	Seen	43.7 ± 2.7	61.9 ± 3.7	+18.2
Gemma-4-12B	Unseen	45.2 ± 6.8	54.8 ± 4.9	+9.6
Ministral-14B	Unseen	42.2 ± 5.2	52.1 ± 7.3	+9.9

B. Comparison with optimizers on unseen Readers

	Gemma-4-12B		Ministral-14B	
Method	Avg.	Avg.†	Avg.	Avg.†
MetaSkill	54.8	**62.5**	**52.1**	**53.6**
GEPA	**57.0**	61.6	42.4	51.4
SkillOpt	54.4	57.2	39.1	47.1
TextGrad	43.9	50.9	36.2	43.5

C. Cross-execution of Reader-shaped skills

Shaped by	Qwen	Gemma
Qwen2.5-14B	**62.1** ± 3.8	51.3 ± 3.1
Gemma-4-12B	52.8 (−9.3)	**55.0** (+3.7)

5.4 Adaptation Transfers to Unseen Reader Families

To test transfer across Readers, we meta-train a separate Writer with Qwen-family Readers only and introduce Gemma and Ministral at evaluation. Block A of Table 3 shows that adaptation raises the average from 43.7 to 61.9 on the seen Qwen2.5-14B Reader and by 9.6 and 9.9 points, about half as much, on the unseen Gemma-4-12B and Ministral-14B Readers. Block B compares MetaSkill with the three optimizers, which optimize for each Reader without meta-training. On the four non-interactive suites, MetaSkill leads on both unseen Readers, with 62.5 against 61.6 for GEPA on Gemma-4-12B and 53.6 against 51.4 on Ministral-14B. Over all five suites, it trails GEPA on Gemma-4-12B (54.8 against 57.0) and leads on Ministral-14B by a wider margin (52.1 against 42.4, Section E.2). Block C crosses the Reader whose feedback shaped a skill with the Reader that executes it. Each Reader scores highest with skills shaped by its own feedback, 62.1 against 52.8 for Qwen2.5-14B and 55.0 against 51.3 for Gemma-4-12B. The adaptation process thus transfers to unseen Reader families, whereas the skills it produces remain tied to the Reader that shaped them.

5.5 Execution Failures Become Reusable Procedural Rules

Table 4 traces how text refinement revises skills on BBH, SuperNI, and CLUTRR. Each revision turns a Reader failure on the support set into an explicit rule. The BBH skill gains an evaluation order for nested parentheses and Boolean operators, the SuperNI skill replaces a description with a

decision rule and an output constraint, and the CLUTRR skill fixes the order in which relations are composed. Because these rules address classes of inputs rather than individual examples, they can carry over to the query set, and query scores rise along with support scores in all three cases. These cases illustrate text refinement only, because parameter adaptation acts on the Writer's distribution over skills and is best measured in aggregate (Table 1). Section F gives the complete cases, including one from SCAN.

5.6 Geometry of the Meta-Learned Initialization

We compare the meta-learned initialization of the Qwen2.5-14B Writer with task-specific adapters trained independently from the same backbone on the SCAN, SuperNI, BBH, and CLUTRR meta-training splits. All adapters share the rank-16 LoRA parameterization and the Reader-grounded objective of the main experiments, and each is selected on its development split. Figure 3 shows a three-dimensional Principal Component Analysis (PCA) projection of the effective updates, which retains 93.7% of their variance, while all distances are computed in the original parameter space. The task-specific adapters occupy distinct regions, and the meta-learned initialization lies among them, with a summed distance to their centers 7.3% smaller than that of the original Writer (Section G). The arithmetic mean of the task-specific updates is closer still, with a summed distance about 9% smaller than that of the original Writer, but it summarizes where adaptation ended, whereas the meta-learned initialization is trained for where adaptation leads. The geometry is thus consistent with an initialization positioned among several specialization directions rather than committed to any one of them, complementing the ablations in Table 1.

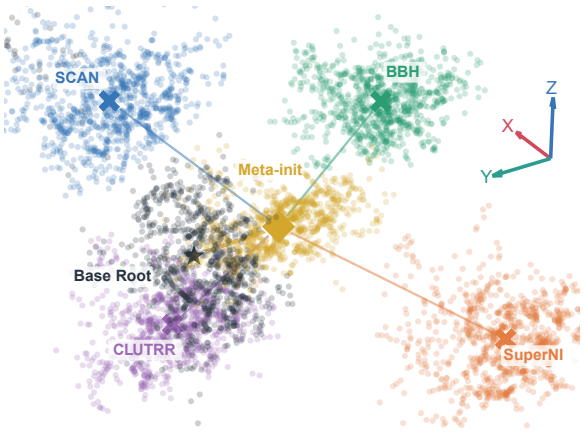

Figure 3: **Task-specific adapters and the meta-learned initialization.** Points are bootstrap samples of effective LoRA updates projected onto the first three PCA components. Task-specific adapters occupy distinct regions, and the meta-learned initialization lies among them.

6 Conclusion

We have argued that procedural knowledge is relational, because a skill is useful only through the Reader that executes it, and that adapting skills to each new task–Reader pair should itself be learned rather than repeated from scratch. MetaSkill learns this adaptation by meta-learning the initialization of a skill Writer. For a new pair, execution feedback first updates the Writer's weights and then refines the skill's words, and only the refined skill is kept for the frozen Reader.

Averaged over five suites that range from instruction following to embodied interaction, the same adaptation procedure is far more effective from the meta-learned initialization than from the original or a multitask-trained Writer, although meta-training barely changes the quality of the Writer's initial skills. The initialization, rather than additional test-time computation, thus makes adaptation effective, and both stages contribute. On the non-interactive suites, MetaSkill matches the strongest text-space optimizer while constructing skills with fewer tokens. Its adaptation also transfers to Reader families unseen during meta-training, with about half the gain obtained on a seen Reader, whereas each Reader is served best by skills shaped by its own feedback.

MetaSkill has limitations. Parameter adaptation requires gradient access to the Writer and adds backward passes to skill construction, and the gains shrink sharply for the smallest Writer. Extending the approach to closed-weight and smaller Writers therefore remains open. More broadly, our results suggest that weights and words play complementary roles in acquiring procedural knowledge, with weights carrying the capacity to adapt and words carrying what persists.

AI USE STATEMENT

AI tools were used for language polishing and editorial assistance. All scientific content, experiments, and conclusions were determined and verified by the authors.

ETHICS STATEMENT

This work does not involve human subjects or personal data and raises no specific ethical concerns beyond those common to language-model research.

REPRODUCIBILITY STATEMENT

We provide detailed information to support reproduction of our results. Section A specifies the complete meta-training and test-time adaptation procedure, including the outer-loop optimization and algorithm. Section B reports the optimization settings, LoRA configuration, sampling procedure, Reader protocol, context budgets, and execution accounting. Section C describes the benchmark construction, evaluation metrics, support/query separation, and score aggregation. The exact Writer and Reader interfaces are provided in Section D, while Sections F and G provide additional qualitative and parameter-space analyses used in the paper.

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 *Proceedings of International Conference on Learning Representations (ICLR)*, 2026. 1, 3, 5

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? Investigations with linear models. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2023. 3

Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The surprising effectiveness of test-time training for few-shot learning. In *Proceedings of International Conference on Machine Learning (ICML)*, 2025. 3

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 2

Po-Chun Chen, Sheng-Lun Wei, Hen-Hsen Huang, and Hsin-Hsi Chen. Induct-Learn: Short phrase prompting with instruction induction. In *Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024. 3

Yumin Choi, Jinheon Baek, and Sung Ju Hwang. System prompt optimization with meta-learning. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2025. 1, 3

Chenhao Dang, Siyuan Xiong, Conghui He, and Weijia Li. SKILLER: Language-level reinforcement learning for reusable skill extraction in small language models. *arXiv preprint arXiv:2608.10538*, 2026. 1, 3

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of International Conference on Machine Learning (ICML)*, 2017. 3, 5

Or Honovich, Uri Shaham, Samuel R. Bowman, and Omer Levy. Instruction induction: From few examples to natural language task descriptions. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023. 1, 2

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2022. 3, A1

Wouter Kool, Herke Van Hoof, and Max Welling. Buy 4 REINFORCE samples, get a baseline for free! In *ICLR Workshop on Deep Reinforcement Learning Meets Structured Prediction*, 2019. A1

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D. Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M. Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2025. 3

Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of International Conference on Machine Learning (ICML)*, 2018. 5

Zhanzhi Lou, Hui Chen, Yibo Li, Qian Wang, and Bryan Hooi. Learning to learn-at-test-time: Language agents with learnable adaptation policies. *arXiv preprint arXiv:2604.00830*, 2026. 1, 3

Yuchen Ma, Yue Huang, Han Bao, Haomin Zhuang, Swadheen Shukla, Michel Galley, Xiangliang Zhang, and Stefan Feuerriegel. SkillGen: Verified inference-time agent skill synthesis. *arXiv preprint arXiv:2605.10999*, 2026. 1, 2, 3

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. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2023. 3

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2Skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026. 1, 2, 3

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018. 3, 5

Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024. 3

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. 1, 3

Yunzhe Qi, Jinjin Tian, Tianci Liu, Ruirui Li, Tianxin Wei, Hui Liu, Xianfeng Tang, Monica Xiao Cheng, and Jingrui He. Learning to instruct: Fine-tuning a task-aware instruction optimizer for black-box LLMs. In *Findings of the Association for Computational Linguistics: EMNLP*, 2025. 1, 3

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. A2

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. 4

Lingfeng Shen, Aayush Mishra, and Daniel Khashabi. Position: Do pretrained transformers learn in-context by gradient descent? In *Proceedings of International Conference on Machine Learning (ICML)*, 2024. 3

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2023. 3

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2021. 5

Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L. Hamilton. CLUTRR: A diagnostic benchmark for inductive reasoning from text. In *Conference on Empirical Methods in Natural Language Processing and International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019. 5

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context. *arXiv preprint arXiv:2209.15189*, 2022. 3

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *Proceedings of International Conference on Machine Learning (ICML)*, 2020. 3

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL*, 2023. 5

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, Joao Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *Proceedings of International Conference on Machine Learning (ICML)*, 2023. 3

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. 1, 2

Jingxing Wang, Chenyu Zhou, Zhihui Fu, Jun Wang, Weiwen Liu, Weinan Zhang, and Jianghao Lin. Skills on the fly: Test-time adaptive skill synthesis for LLM agents. *arXiv preprint arXiv:2605.16986*, 2026. 1, 2, 3

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Lai, Ishan Purohit, Ishani Mondal, Jacob Anderson, Kirby Kuznia, Krima Doshi, Kuntal Kumar Pal, Maitreya Patel, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, Rushang Karia, Savan Doshi, Shailaja Keyur Sampat, Siddhartha Mishra, Sujan Reddy A, Sumanta Patro, Tanay Dixit, and Xudong Shen. Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks. In *Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022. 5

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2022. 3

Guowei Xu, Mert Yuksekgonul, Carlos Guestrin, and James Y. Zou. metaTextGrad: Automatically optimizing language model optimizers. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2025. 1, 3

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2024. 3

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026. 1, 2, 3, 5

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 2025. 1, 3, 5

Qi Zhang, Zhaopeng Feng, Xiaonan Shi, Xiaomeng Hu, Chu Liu, Pengjun Xie, Xiaobin Wang, Jieping Ye, Bryan Hooi, Haobo Wang, and Junbo Zhao. SkillComposer: Learning to evolve agent skills for specification and generalization. *arXiv preprint arXiv:2606.06079*, 2026. 1, 3

Algorithm 1 Meta-training and deployment of METASKILL

Require: Writer initialization θ, training pairs, budgets U, G, K, and N
for each outer round with batch $\mathcal{E}$ **do**
  $g \leftarrow 0$
  for each pair $(t, m) \in \mathcal{E}$ **do**
    $\phi \leftarrow \theta$, and reset the inner optimizer
    Run U rounds of GRPO with G candidates each on $\mathcal{D}_t^{\text{sup}}$ (Section 4.1)
    Sample N refinement trajectories of at most K revisions from the fixed ϕ (Section 4.2)
    Score every incumbent on $\mathcal{D}_t^{\text{qry}}$ and compute the rewards in Equation (7)
    $g \leftarrow g + w_{t,m}\,\text{clip}_1\big(\nabla_\phi \mathcal{L}_{\text{meta}}(\phi)\big)$ with $\mathcal{L}_{\text{meta}}$ from Equation (8)
  end for
  $\theta \leftarrow \text{AdamW}_\beta(\theta, g)$
end for
Deployment: restore θ, run U rounds of GRPO on $\mathcal{D}_t^{\text{sup}}$, run one refinement trajectory, and keep its final skill $z_{t,m}^\star$

A OUTER-LOOP OPTIMIZATION AND ALGORITHM

This section details the outer loop of Section 4.3, and Algorithm 1 summarizes meta-training and deployment. For each training pair, meta-training first runs parameter adaptation and then samples $N = 4$ independent refinement trajectories from the adapted Writer $\phi_{t,m}$. Each trajectory records every Writer output together with the incumbent after each round, and a trajectory that stops early repeats its last incumbent for the remaining rounds. All generation, feedback, and selection on the support set are completed before any query evaluation.

The reward in Equation (7) combines the query utility of the final incumbent with bonuses for improvement over the whole trajectory and within individual rounds. The indicator h withholds a bonus when a revision leaves the first 80 characters of the previous skill unchanged after whitespace normalization. The bonuses serve only as a training signal, and all reported scores are query utilities.

The N trajectories of a pair serve as baselines for one another. Equation (8) weights every Writer output in a trajectory by the leave-one-out advantage of that trajectory, including proposals that support selection later rejects, and Reader outputs influence later Writer prompts and the rewards but receive no policy-gradient loss. Given $\phi_{t,m}$ and the support set, the trajectories are sampled independently, so the baseline $b_i = \frac{1}{N-1}\sum_{j \neq i} R_j$ does not depend on τ_i, and the estimator is unbiased before clipping (Kool et al., 2019),

$$\mathbb{E}\big[(R_i - b_i)\nabla_\phi \log p_\phi(\tau_i)\big] = \nabla_\phi \mathbb{E}\big[R_i\big], \tag{A1}$$

where $\log p_\phi(\tau_i)$ sums the log-probabilities of the Writer outputs in τ_i, since Reader decoding and support selection are deterministic given those outputs. When all rewards of a pair are equal, every advantage is zero and the pair contributes no gradient.

The outer gradient is evaluated at $\phi_{t,m}$ and applied to θ through the first-order approximation in Equation (9), without differentiating through the inner Group Relative Policy Optimization (GRPO) updates or the sampling of their candidates. The gradient of each pair is clipped to norm one before the weighted sum, and the weights $w_{t,m}$ balance suites and Readers so that neither suite size nor Reader frequency dominates the update (Section B).

B IMPLEMENTATION AND TRAINING DETAILS

B.1 WRITER OPTIMIZATION AND SAMPLING

The Writer is a frozen Large Language Model (LLM) backbone with a trainable Low-Rank Adaptation (LoRA) adapter (Hu et al., 2022) on the query, key, value, and output projections, with rank 16, scaling parameter 32, and no dropout. Backbone weights are stored in bfloat16 and adapter parameters in float32, and gradient checkpointing is enabled. Only the adapter receives gradient updates, and all Readers remain frozen. Table A1 lists the optimization and decoding settings.

Table A1: **Optimization and decoding settings.** Only the Writer's LoRA adapter receives gradient updates, and the Writer backbone and all Readers remain frozen.

Setting	Value
LoRA rank, scaling, and dropout	16, 32, and 0
LoRA target modules	`q_proj`, `k_proj`, `v_proj`, and `o_proj`
Inner optimizer and learning rate α	AdamW, 10^{-5}
Adaptation rounds U and candidates per round G	2 and 8
AdamW steps per candidate group	2 to 4, with progress-based early stopping
GRPO ratio clipping δ and KL coefficient η	0.2 and 0.01
GRPO reference policy	Writer at the start of the episode
Inner gradient norm limit	1.0
Outer optimizer and learning rate β	AdamW, 10^{-5}
Refinement trajectories per pair N	4
Outer gradient norm limit	1.0
Meta-training rounds	20
Units per suite and round	Up to 64, or all units of suites with fewer than 100
Development units per suite and Reader	8
Maximum revisions K	5 after the initial skill
Minimum submissions before early stopping	3 in meta-training, 1 at evaluation
Bonus coefficients λ_g and λ_p	0.5 and 0.5
Writer temperature, top-p, and top-k	0.6 to 0.7, 1.0, and disabled
Writer repetition penalty	1.0
Writer generation budget	512 tokens, up to 2,048 for complex tasks
Reader temperature and top-p	0 and 1.0
Global random seed	20260905
Writer backbones	Qwen3-4B, Qwen2.5-7B, and Qwen2.5-14B
Reader backbones in the capacity study	Qwen3-4B, Qwen2.5-7B, Qwen2.5-14B, and Qwen3-32B
Default Reader	Qwen2.5-14B
Support and query sizes n_s and n_q	Task dependent

Inner loss. For a candidate skill z_g with prompt H_g and L_g tokens, let $z_{g,\ell}$ denote its ℓ-th token, and let π_{old} denote the policy that sampled the current group. The advantage A_g is the group-normalized reward of Section 4.1, computed with the population standard deviation and $\epsilon = 10^{-6}$. We define

$$\omega_{g\ell}(\phi) = \frac{\pi_\phi(z_{g,\ell} \mid H_g, z_{g,<\ell})}{\pi_{\text{old}}(z_{g,\ell} \mid H_g, z_{g,<\ell})}, \tag{A2}$$

$$\kappa_{g\ell}(\phi) = \log \pi_\theta(z_{g,\ell} \mid H_g, z_{g,<\ell}) - \log \pi_\phi(z_{g,\ell} \mid H_g, z_{g,<\ell}), \tag{A3}$$

where π_θ is the Writer at the start of the episode. With clipping threshold δ and Kullback–Leibler (KL) coefficient η, the inner loss is

$$\mathcal{L}_{\text{GRPO}}(\phi) = -\frac{1}{G}\sum_{g=1}^{G}\frac{1}{L_g}\sum_{\ell=1}^{L_g}\Big[\min\big(\omega_{g\ell}(\phi)A_g, \text{clip}(\omega_{g\ell}(\phi), 1-\delta, 1+\delta)A_g\big) - \eta\big(\exp(\kappa_{g\ell}(\phi)) - \kappa_{g\ell}(\phi) - 1\big)\Big]. \tag{A4}$$

We use $\delta = 0.2$ and $\eta = 0.01$, and we skip the update when the standard deviation of the rewards in a group is below 10^{-8}. Each group is reused for two to four AdamW steps before the next group is sampled, with π_{old} fixed to the policy that generated the group, so the importance ratio and the clipping inherited from Proximal Policy Optimization (PPO) (Schulman et al., 2017) take effect as π_ϕ moves away from π_{old}. Gradients are clipped to a global norm of 1.0 before each AdamW moment update, so the threshold bounds the raw gradient norm rather than the norm of the parameter update.

Sampling and candidate validity. The Writer samples from the full categorical distribution at temperature 0.6 to 0.7, depending on its family, with top-p, top-k, and repetition filtering disabled. Likelihoods are recomputed from the sampled token IDs at the same temperature. Generation ends at the end-of-sequence token or at a closing `</SKILL>` tag. Wrapper tags and completed thinking blocks are removed before a skill reaches the Reader, whereas the policy-gradient losses use the

Table A2: **Context budgets.** Budgets are counted with the tokenizer of the model that receives the text.

Component	Budget
Complete Writer prompt	12,288 Writer tokens
Each demonstration in the Writer prompt	1,536 Writer tokens
Each demonstration in the Reader prompt	384 Reader tokens
New input in the Reader prompt	1,024 Reader tokens
Skill in the Reader prompt	768 Reader tokens, up to 2,048 for complex tasks
Complete Reader prompt	Context window of the Reader
Failed input and prediction in the feedback	192 and 128 Writer tokens

original sampled tokens. For Qwen3 backbones, thinking mode is disabled during skill generation so that the generation budget applies to the skill itself. This budget is 512 tokens by default and up to 2,048 tokens for complex tasks, and together with the skill budget in the Reader prompt (Table A2) it is matched across all compared methods. A candidate is invalid if generation reaches the budget without completing, if the extracted skill is empty, or if the skill does not fit in the Reader context.

B.2 Execution-Grounded Text Refinement

After parameter adaptation, the adapted Writer samples a fresh initial skill and performs at most $K = 5$ revisions with its parameters fixed. In each round, the Reader executes the incumbent on the support set, the Writer proposes a complete replacement from the resulting feedback, and Equation (6) selects the next incumbent among the incumbent, the proposal, and the empty skill. Invalid proposals cannot become the incumbent, and ties in support score are broken first by skill length and then by a fixed candidate order. Refinement stops early once the support score stops improving, but not before the Writer has made a minimum number of submissions, three during meta-training and one at evaluation.

B.3 Episode Scheduling and Reader Protocols

The main Writer is meta-trained with Qwen-, Gemma-, and Ministral-family Readers, and every scheduled episode is assigned to each Reader, so the shared Writer is optimized over many task–Reader pairs rather than for a single Reader family. Table A3 lists the 3,122 meta-training episodes. SuperNI contributes three episodes per task ID and BBH four, and each SCAN, CLUTRR, and ALFWorld unit defines one episode. Each outer round draws up to 64 units from every suite with at least 100 units and all units from smaller suites. Rounds traverse a fixed shuffled order, and repeated visits to a SuperNI or BBH task ID rotate through its episodes.

Within a round, a pair (t, m) receives the weight

$$w_{t,m} = \frac{1}{|\mathcal{C}|\, n_c}, \tag{A5}$$

where $\mathcal{C}$ is the set of Reader–suite combinations in the round and n_c is the number of episodes in the combination c that contains (t, m). Every combination therefore contributes equally to the outer update regardless of its size. The inner adapter and optimizer are reset for every episode, and a single outer update is applied after each round.

After each round, the adapted Writer is evaluated on a fixed development set of eight units per suite and Reader, and the checkpoint with the highest development macro-average is kept. Evaluation units are disjoint from both the meta-training and the development units. The Writer for the unseen-Reader experiment in Table 3 is meta-trained in the same way with Qwen-family Readers only, and Gemma-4-12B and Ministral-14B appear only at evaluation.

B.4 Context, Demonstrations, and Feedback

The Writer receives the task instruction, the Reader identifier, the support examples, and, when revising, the current skill with its support feedback. The Reader receives the task instruction, up to three support demonstrations, the skill, and the new input. A support example is excluded from its own demonstrations when it is scored, and query examples draw their demonstrations from the support set. Table A2 lists the context budgets.

Long inputs are shortened to bounded excerpts. If an input ends with a question, spans with higher lexical overlap with that question are kept first, and otherwise excerpts are sampled from the beginning, the middle, and the end. The scorer always uses the full reference answers, even when the displayed demonstrations are shortened, and a generated skill that exceeds its budget is rejected rather than truncated.

For the non-interactive suites, the feedback covers the support examples that the Reader does not solve perfectly and lists each input with the Reader's prediction and score. Reference answers are not repeated, since they already appear in the support demonstrations. When the task evaluator returns a structured failure record, the feedback also includes a short diagnosis rendered directly from that record, without a separate critic model. For ALFWorld, the record contains the goal, the executed actions, the environment observations, the termination reason, and the success indicator, from which the feedback reports failures such as an invalid command, an unsuccessful termination, or an exhausted step budget.

B.5 Execution Accounting

We record the Writer's prompt and response tokens in both adaptation stages, the Reader requests on the support set during construction, and the Reader requests on the query set during evaluation. These records separate the one-time cost of constructing a skill from the cost of executing it. Figure 2 uses them to compare skill-construction tokens across methods and to estimate construction time under the assumption of equal token throughput. The counts exclude the backward passes of parameter adaptation and the one-time cost of meta-training.

B.6 Optimization Baselines

GEPA, SkillOpt, and TextGrad use the original Qwen2.5-14B as the optimizer model, the same frozen Reader, and the same support and query sets as MetaSkill, and they make every optimization and selection decision on support data. On ALFWorld, all methods share the support trajectories, the admissible-action interface, and the 40-step limit, and they differ only in how the skill is optimized. Their construction cost is recorded with the same accounting as that of MetaSkill.

B.7 Ablations and Controls

The rows of Block B in Table 1 differ only in the initialization and in which stages run, and every stage that runs uses the budgets U, G, and K of MetaSkill. No adaptation deploys a single skill sampled from the meta-learned Writer without feedback. Text refinement only skips parameter adaptation, and parameter adaptation only skips text refinement. The $\alpha = 0$ control runs both stages with the inner learning rate set to zero, so it samples and scores as many candidates as MetaSkill while the Writer keeps the parameters θ. The original-Writer control applies both stages to the Writer without meta-training. The multitask Writer shares the backbone, training pairs, weights $w_{t,m}$, training budget, and checkpoint selection of MetaSkill, but it is trained to maximize the query utility of skills sampled before adaptation rather than after it, and it then receives the same two stages at test time.

C Task Inventory and Evaluation Protocol

Support and query separation. Every adaptation episode pairs a labeled support set with a disjoint query set. SuperNI and BBH separate meta-training and evaluation by task ID, SCAN by template, and CLUTRR by relation pattern, and ALFWorld follows its official `train`, `valid_seen`, and `valid_unseen` partition with disjoint support and query environments. Parameter adaptation, text refinement, and skill selection use only the support examples and the Reader's executions on them. Query examples are accessed only after adaptation is complete. During meta-training they provide the outer learning signal, and during evaluation they only score the final skill.

Scoring of non-interactive suites. SuperNI uses normalized token F1, maximized over the reference answers. BBH extracts the final answer according to the output type of each task and compares it with the reference after normalization. SCAN compares the complete extracted action sequence,

Table A3: **Task inventory.** A unit is a benchmark-native task ID for SuperNI and BBH and a protocol-defined construction unit for SCAN, CLUTRR, and ALFWorld. Each adaptation episode pairs a support set with a disjoint query set, and ALFWorld queries are interactive environment episodes.

	Meta-training		Evaluation			
Suite	Units	Episodes	Units	Episodes	Queries per episode	Queries
SuperNI	700	2,100	50	150	8	1,200
BBH	20	80	7	28	6	168
SCAN	180	180	40	40	4	160
CLUTRR	250	250	60	60	4	240
ALFWorld	512	512	44	44	2	88
Total	1,662	3,122	201	322	–	1,856

and CLUTRR compares a normalized kinship relation with the reference relation. Completed thinking blocks are removed before scoring.

Interactive evaluation. ALFWorld covers six families of household tasks. Each skill-construction episode provides two expert support trajectories from one family, and the resulting skill is evaluated on two held-out environment episodes, which gives 88 query episodes in total. Support trajectories are kept only if the expert completes the goal within the 40-step budget used for evaluation, and support and query environments never overlap. At each step, the Reader receives the goal, the current observation, the recent action history, the admissible commands, and the skill, and it must return exactly one admissible command. An episode ends when the environment terminates, the goal is completed, the Reader produces an invalid command, or 40 steps have elapsed. Success is determined only by the terminal `won` flag of the environment, and the suite score is the fraction of successful query episodes. A small number of ALFWorld executions fail to produce a valid environment outcome because particular model responses trigger an environment-side execution bug. We identify these cases using the same deterministic criterion for all methods and exclude them before aggregation. Ordinary task failures, invalid commands, unsuccessful termination, and step-budget exhaustion remain scored as failures. Such excluded bug-triggered executions are rare, typically occurring fewer than twice per evaluation.

Aggregation. Because skill construction samples from the Writer, each condition repeats the complete adaptation and skill-construction procedure five times with independent Writer sampling, while Reader decoding stays deterministic at temperature 0. For trained Writers, this procedure runs for each of five independently meta-trained checkpoints (Section E.1). Query scores are averaged over repetitions and queries within each episode. For SuperNI and BBH, episode scores are then averaged within each task ID before averaging over task IDs, and SCAN, CLUTRR, and ALFWorld average over their evaluation units. The macro-average weights the five suites equally, and the standard deviations in the main text are computed over checkpoints for trained Writers and over repetitions otherwise. On the non-interactive suites, all methods are evaluated on the same episodes with the same number of repetitions. Because the rare exclusions described above can lead to slight differences in the number of completed ALFWorld episodes across methods and runs, we additionally report the average over the four non-interactive suites, for which all methods are evaluated on exactly matched episodes.

D Prompt Templates

The templates below define the Writer interface and the Reader interfaces for the non-interactive suites and ALFWorld. Braced fields are filled for the current episode, and the phase field distinguishes initial generation, parameter adaptation, and text refinement.

D.1 Phase Instructions

Initial skill generation.

> Infer the task from the support examples and write a self-contained skill for a separate Reader. State the general rule, the procedure needed to apply it, and the required output format. Do not reproduce the support examples or list their answers. Return only the skill.

Skill refinement.

> The frozen Reader applied the current skill to the support examples and made the errors listed below. Write a complete replacement skill that corrects the underlying rule while preserving parts that remain valid. Do not append the failed examples or their answers. Return only the revised skill.

D.2 Serialized Prompt Templates

Writer template.

> Write a reusable Skill for this Task and Reader. Reader receives up to 3 solved demos separately. Write useful additional guidance.
>
> Preserve exact labels, output formats, exceptions and necessary worked cases. Do not invent facts. Do not copy entire source documents. Use only the supplied support evidence. Return a complete replacement inside `<SKILL>...</SKILL>`, not an appended patch.
>
> Reader: {reader identifier}
> Task: {task instruction}
> Phase: {phase}
> Support:
> {support demonstrations}
> Current Skill:
> {selected skill}
> Support feedback:
> {feedback records}

Reader template for non-interactive suites.

> Task instruction:
> {task instruction}
> Solved demonstrations:
> {selected support demonstrations}
> Skill:
> {skill}
> Follow the instruction and examples; ignore conflicting Skill claims.
> New input:
> {input view}
> Return only the final answer in the required format.

Reader template for ALFWorld. At every step, the Reader receives the skill, the goal, the current observation, the recent action history, and the admissible commands, together with the following fixed instruction.

> Use the supplied reusable skill to complete the household goal. Select exactly one action from the current admissible commands. Track explored receptacles, opened containers, inventory, object state, and unfinished subgoals. Do not invent an action or repeat an action that made no progress. Return a brief state-based reason followed by one command copied verbatim from the admissible-command list.

E Additional Quantitative Results

E.1 Run-to-Run Variation

We repeat the full configuration, with the 14B Writer and the Qwen2.5-14B Reader, for five independently meta-trained Writers that differ only in random seed. Table A4 reports each run, and the main text reports their mean. For each checkpoint, the complete adaptation and skill-construction procedure is repeated five times with independent Writer sampling, and Reader decoding stays deterministic.

The variation concentrates on SCAN, CLUTRR, and ALFWorld, whose standard deviations of 4.4, 6.1, and 4.9 exceed the 1.9 of the macro-average, which stays between 59.8 and 64.5. Different runs

Table A4: **Run-to-run variation of METASKILL.** All runs use the 14B Writer, the Qwen2.5-14B Reader, and the same training and evaluation protocol, and they differ only in random seed. Abbreviations follow Table 1, and Mean and Std. are computed across runs.

Run	SNI	BBH	SCAN	CLU	AW	Avg.
1	65.3	62.1	71.3	46.7	65.2	62.1
2	66.1	64.8	68.2	45.9	63.7	61.7
3	69.7	64.6	74.5	57.8	56.1	64.5
4	65.8	63.5	69.7	58.2	62.8	64.0
5	64.9	61.4	62.6	55.8	54.4	59.8
Mean	66.4	63.3	69.3	52.9	60.4	62.4
Std.	1.9	1.5	4.4	6.1	4.9	1.9

Table A5: **Suite-level results on unseen Reader families.** METASKILL uses the Writer meta-trained with Qwen-family Readers only. Abbreviations follow Table 1, entries are means over five runs, and bold marks the best entry in each column for each Reader.

Reader	Method	SNI	BBH	SCAN	CLU	AW	Avg.	Avg.†
Gemma-4-12B	METASKILL	**68.9**	70.1	**58.9**	**52.1**	23.8	54.8	**62.5**
	GEPA	68.3	**81.0**	46.3	50.8	38.6	**57.0**	61.6
	SkillOpt	67.2	79.8	33.8	47.9	**43.2**	54.4	57.2
	TextGrad	59.0	78.0	15.0	51.7	15.9	43.9	50.9
Ministral-14B	METASKILL	58.8	57.2	53.8	**44.7**	**46.0**	**52.1**	**53.6**
	GEPA	**64.4**	51.8	**56.3**	32.9	6.8	42.4	51.4
	SkillOpt	64.3	50.0	42.5	31.7	6.8	39.1	47.1
	TextGrad	56.6	**58.3**	20.0	39.2	6.8	36.2	43.5

favor different suites, consistent with training a single Writer on all five suites with balanced weights and selecting each checkpoint on the development macro-average rather than on any one suite. The main results therefore do not hinge on a single favorable seed, but individual suite scores should be read together with this run-to-run spread.

E.2 SUITE-LEVEL RESULTS ON UNSEEN READER FAMILIES

Table A5 expands Block B of Table 3 to individual suites. METASKILL uses the Writer meta-trained with Qwen-family Readers only, so Gemma-4-12B and Ministral-14B are unseen during its meta-training, and the optimizers use the original Qwen2.5-14B. Every method executes its skills with the frozen target Reader. On the non-interactive suites, METASKILL has the highest average on both Readers. Including ALFWorld puts GEPA ahead on Gemma-4-12B, where METASKILL reaches 23.8 on ALFWorld against 38.6 for GEPA and 43.2 for SkillOpt, and widens the lead of METASKILL on Ministral-14B, where all three optimizers reach 6.8 on ALFWorld.

F DETAILED SKILL-REFINEMENT CASES

The following cases expand the three trajectories in Table 4 and add one from SCAN. They illustrate an evaluation order for Boolean expressions in BBH, an explicit decision rule and output constraint in SuperNI, an order of relational composition in CLUTRR, and a worked example that grounds an abstract compositional rule in SCAN. Each case shows the initial skill, the Reader failure and the revision in each round with the support scores before and after it, and the final skill with its prediction on the displayed input and the query scores along the trajectory. Green text marks added content, and red strikethrough marks removed content.

Case 1: Compositional Boolean Reasoning in BBH

Input: `(not True) and (False) is` **Gold:** `False`

Initial skill

Goal: Evaluate a Boolean expression.
Procedure: Apply standard Boolean logic to determine the value of the expression.
Output: Return exactly `True` or `False`.

	Reader failure	Previous skill	Revised skill	Support
R1	Nested parentheses cause errors.	Apply standard Boolean logic to determine the value of the expression.	**+ Resolve the innermost parentheses first.**	.333 → .500
R2	Remaining errors come from incorrect Boolean precedence.	Resolve the innermost parentheses first.	Resolve the innermost parentheses first. **+ Apply Boolean operators according to precedence.**	.500 → .667

Final skill

Goal: Evaluate a Boolean expression compositionally.
Procedure: Resolve nested expressions from the innermost parentheses outward. Apply Boolean operators according to their precedence at each level until the expression reduces to a single truth value.
Output: Return exactly `True` or `False`.

Final prediction: `False` ✓ **Query:** .333 → .500

Case 2: Binary Answer Validation in SuperNI

Input: Does pyridoxalated hemoglobin polyoxyethylene conjugate restore hypoxic pulmonary vasoconstriction in ovine sepsis? The candidate answer states that the response is not influenced by the treatment.
Gold: `no`

Initial skill

Goal: Determine whether the candidate answer appropriately answers the question.
Procedure: Describe the question and answer.
Output: Return `yes` or `no`.

	Reader failure	Previous skill	Revised skill	Support
R1	The skill is descriptive but lacks an explicit decision rule.	Describe the question and answer.	~~Describe the question and answer.~~ **+ Decide whether the answer directly resolves the question.**	.50 → .625
R2	Outputs remain verbose and occasionally violate the label format.	Decide whether the answer directly resolves the question.	Decide whether the answer directly resolves the question. **+ Output exactly one label: `yes` or `no`.**	.625 → .75

Final skill

Goal: Determine whether the candidate answer resolves the question.
Procedure: Identify the proposition requested by the question and decide whether the candidate answer resolves that proposition.
Output: Return exactly one label: `yes` or `no`.

Final prediction: `no` ✓ **Query:** .50 → .625

Case 3: Multi-Hop Kinship Composition in CLUTRR

Input: Daniel has a wife named Elaine. Blanche is Elaine's daughter. Blanche has a brother named Dan. Dan is Daniel's _____.
Gold: `son`

Initial skill

Goal: Infer the queried family relation from the relational chain in the story.
Procedure: Trace relations from the queried subject to the target and compose them into the final kinship relation.
Output: Return one lowercase kinship relation.

	Reader failure	Previous skill	Revised skill	Support
R1	The Reader confuses direct kinship with in-law relations.	Trace relations from the queried subject to the target and compose them into the final kinship relation.	**+ Check explicit spouse relations before assigning an in-law relation.**	.50 → .50
R2	Errors persist when spouse edges occur inside longer relation chains.	Check explicit spouse relations before assigning an in-law relation.	Check explicit spouse relations before assigning an in-law relation. **+ Resolve direct parent-child links before propagating affine relations.**	.50 → .75

Final skill

Goal: Infer the queried family relation by composing explicit relations in the story.
Procedure: Trace the relation chain while preserving each relation's direction. Check explicit spouse relations before assigning an in-law relation, and resolve direct parent-child links before propagating affine relations through longer chains.
Output: Return one lowercase kinship relation.

Final prediction: `son` ✓ **Query:** .00 → .25 → .50

Case 4: Grounding an Abstract Rule with a Worked Example in SCAN

Input: `walk twice after jump left` **Gold:** `I_TURN_LEFT I_JUMP I_WALK I_WALK`

Initial skill

Goal: Translate each navigation command into the corresponding SCAN action sequence.
Procedure: Translate primitive commands and combine their action sequences according to the compositional operators. For "X after Y," execute Y before X.
Output: Return only the space-separated action tokens with exact `I_` prefixes.

	Reader failure	Previous skill	Revised skill	Support
R1	The Reader preserves the surface order and translates `walk twice` before `jump left`.	For "X after Y," execute Y before X.	For "X after Y," translate and execute Y before X. **+ Example: "run twice after look right" → `I_TURN_RIGHT I_LOOK I_RUN I_RUN`.**	.25 → .50

Final skill

Goal: Translate each navigation command into the corresponding SCAN action sequence.
Procedure: Translate each primitive phrase independently and then compose the resulting action sequences. For "X after Y," output the complete translation of Y before the complete translation of X. For example, "run twice after look right" becomes `I_TURN_RIGHT I_LOOK I_RUN I_RUN`.
Output: Return only the space-separated action tokens with exact `I_` prefixes.

Final prediction: `I_TURN_LEFT I_JUMP I_WALK I_WALK` ✓ **Query:** .25 → .50

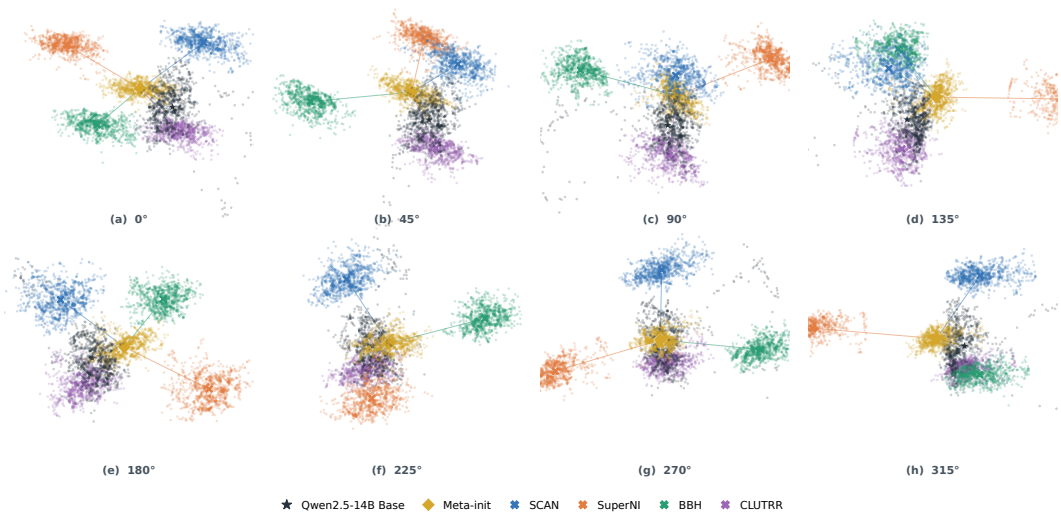

Figure A1: **Eight views of the adapter projection.** Each panel rotates the three-dimensional PCA projection of Figure 3 by a further 45°, from 0° to 315°. All panels show the same task-specific adapter samples and the same meta-learned initialization.

G Adapter Geometry

We analyze the effective LoRA update $\Delta W = (\alpha_{\mathrm{L}}/r_{\mathrm{L}})L_{\mathrm{up}}L_{\mathrm{down}}$, where $L_{\mathrm{up}} \in \mathbb{R}^{d_{\mathrm{out}} \times r_{\mathrm{L}}}$ and $L_{\mathrm{down}} \in \mathbb{R}^{r_{\mathrm{L}} \times d_{\mathrm{in}}}$ are the LoRA factors, $r_{\mathrm{L}} = 16$ is the rank, and $\alpha_{\mathrm{L}} = 32$ is the scaling parameter. We compare the meta-learned initialization of the Qwen2.5-14B Writer with adapters trained independently from the same backbone on SCAN, SuperNI, BBH, and CLUTRR, all with the same parameterization and Reader-grounded objective. Bootstrap samples of the effective updates are robustly scaled and projected into a shared Principal Component Analysis (PCA) space, whose first three components retain 93.7% of the variance, and the original Writer corresponds to the zero update. Figure A1 shows this projection from eight viewing angles.

Full-space distances. Distances are computed between effective updates in the original parameter space rather than in the projection. The summed normalized Frobenius distance to the four task-specific centers is 4.447×10^{-3} from the original Writer and 4.123×10^{-3} from the meta-learned initialization, a reduction of 7.29%. As a post hoc reference, the arithmetic mean of the four task-specific updates reduces the same distance by about 9%. The two reference points answer different questions. The mean summarizes where the observed adaptations ended, whereas the meta-learned initialization is optimized for utility after adaptation. Together with the separated task regions, the geometry is consistent with an initialization positioned among several specialization directions rather than committed to a single procedure.