INTERNALIZED SKILLS NEED QUERY-ADAPTIVE CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Skill internalization enables LLM agents to reuse procedural knowledge through compact parameter updates, reducing the context overhead of skill use. However, internalizing a skill does not determine how strongly it should be applied to a given query: the same intervention can correct errors on some queries while disrupting answers the base model would otherwise get right. To this end, we propose **Query-Adaptive Skill Gating (QASG)**, a lightweight framework that regulates skill contribution on a per-query basis. QASG couples two complementary signals: predictive uncertainty from self-reflection gauges the model's need for assistance, while adapter sensitivity quantifies the skill's response relative to the base computation. Together, these signals allow the same skill to provide different levels of support across queries, helping uncertain predictions while limiting unnecessary intervention. Extensive evaluations across five benchmarks show that QASG outperforms the best fixed-strength baseline by 8.1–8.4 points and achieves the highest average accuracy among the evaluated methods, including TextSkill, LatentSkill, and other adaptive gating methods. These results highlight the importance of query-adaptive strength control for effective use of internalized skills.

1 INTRODUCTION

External skills extend LLM agents beyond their native capabilities, enabling them to tackle tasks ranging from specialized reasoning to complex tool use (Wang et al., 2023; Xia et al., 2026; Wu et al., 2026; Ouyang et al., 2026; Pan et al., 2026; Wang et al., 2026). As skill libraries expand and their instructions grow more complex, repeatedly including full skill instructions consumes an increasing share of the model's context window. Skill internalization emerges as a compelling approach to this challenge: it encodes reusable procedures in learned parameters, allowing agents to apply specialized skills without loading the corresponding documents in full at inference time. Recent approaches have explored lightweight adapters and skill-conditioned hypernetworks to this end, yielding context-saving parameterized skills that support more accurate skill application without explicitly exposing proprietary skill content (Zhang & Qi, 2026; Zhao et al., 2026; Charakorn et al., 2026; Liu et al., 2026).

However, existing skill-internalization methods leave a critical decision unresolved: how much the model actually needs skill assistance for a given query. As Figure 1 shows, increasing the injection scale corrects a substantial fraction of base-incorrect queries but simultaneously introduces errors on queries the base model already answers correctly. The same adapter update that compensates for missing knowledge can also interfere with successful reasoning, so any fixed scale helps one group of queries at the expense of another. This query-dependent tradeoff is a central obstacle to reliable skill-based capability extension, necessitating query-adaptive control of injection strength.

But how can we anticipate when the base model needs skill assistance without knowing the correct answer? Prior work suggests that language models exhibit some awareness of what they do and do not know (Kadavath et al., 2022; Ferrando et al., 2025). Building on these findings, we measure predictive uncertainty as the model reflects on the background knowledge required by a query. This uncertainty serves as a query-adaptive proxy for the model's need for assistance, encouraging greater reliance on external skills when uncertainty is high and allowing the model to rely more on its own knowledge when uncertainty is low.

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.

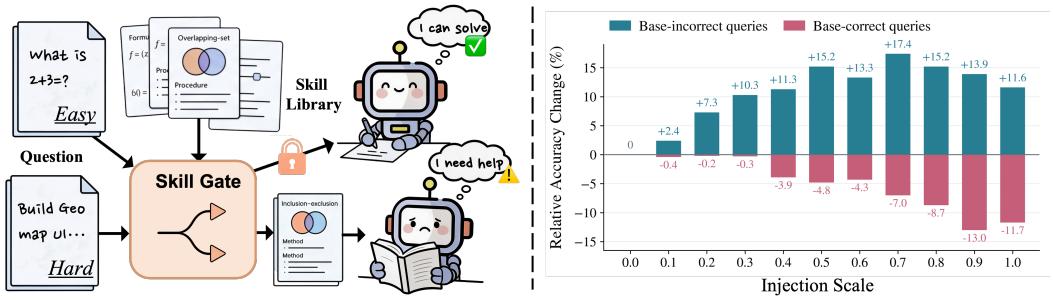


Figure 1: Query-dependent benefits and costs of skill injection. Left: queries differ in their need for skill assistance. Right: our Qwen3-4B measurements on the TheoremQA evaluation set, grouped by whether the base model answers correctly without skill injection. Bars show percentage-point changes in accuracy relative to each group's no-injection baseline as the injection scale varies. Skill injection corrects some base-incorrect queries while introducing errors on base-correct queries.

Uncertainty alone, however, overlooks how strongly the selected skill affects the model, which can vary across queries even at the same injection strength. Even when the model needs assistance, an overly strong skill response may disrupt its internal computation and impair answer accuracy. To characterize this response, we introduce adapter sensitivity, measured by the adapter branch's activation relative to the base computation in a single forward pass. We use this signal to restrain further amplification when the adapter's relative response is already strong. Combining these complementary signals, we propose **Query-Adaptive Skill Gating (QASG)**, which regulates skill contribution according to both the model's need for assistance and its response to the skill.

Across five benchmarks and two model sizes, QASG consistently delivers large gains. It improves average accuracy by 8.1–8.4 percentage points over validation-selected fixed-strength policies using the same adapters and outperforms strong textual and adaptive baselines, including TextSkill and LatentSkill. Ablation studies confirm the contributions of both signals and their coupling.

Our contributions are summarized as follows.

1. We identify a key problem in skill internalization: a learned skill adapter is conventionally injected at a fixed strength, which helps some queries and disrupts others. We argue this strength should instead be a continuous, per-query control decision.
2. We propose Query-Adaptive Skill Gating (QASG), which uses self-reflection uncertainty and adapter sensitivity as complementary, training-free signals to regulate skill contribution for each query. These signals are lightweight and expressive, making the gate broadly deployable.
3. We release a large-scale public training corpus for skill-conditioned hypernetworks. Across five diverse benchmarks, QASG outperforms a broad range of baselines, and ablations confirm that both signals and their coupling contribute.

2 Related Work

Parametric skill internalization. Low-rank adaptation provides a foundation for representing skills as compact parameter updates (Hu et al., 2021). Skill-to-LoRA learns skill-specific adapters from skill-guided demonstrations, while Parametric Skills and LatentSkill use hypernetworks to convert textual skills into LoRA updates (Zhang & Qi, 2026; Zhao et al., 2026; Yu et al., 2026). Related adapter-generation methods map task descriptions to adapters, as in Text-to-LoRA, or internalize supplied contexts, as in Generative Adapter, Doc-to-LoRA, and SHINE (Charakorn et al., 2025; Chen et al., 2025; Charakorn et al., 2026; Liu et al., 2026). Beyond adapter synthesis, SKILL0 internalizes skills by progressively withdrawing skill context during reinforcement learning, while Skill0.5 combines general skill internalization with task-specific skill use through difficulty-aware training (Lu et al., 2026; Zhu et al., 2026). QASG builds on this controllable representation by adapting the contribution of an internalized skill to each query.

Dynamic adapter control. Controlling adapter use involves selecting relevant modules and adjusting their contributions to a task or input. AdapterFusion learns to combine task-specific adapter representations, while LoraHub composes LoRA modules using a few examples from the target task (Pfeiffer et al., 2021; Huang et al., 2024). For adaptive weighting and activation, MoLE learns layer-wise weights over LoRA experts, AdaMoLE adapts expert activation thresholds to the input, and LoRA-Flow varies fusion weights across generated tokens (Wu et al., 2024; Liu & Luo, 2024; Wang et al., 2024). Arrow supports adapter routing without retraining, and LoRA on the Go selects and merges adapters using signals from a single forward pass (Ostapenko et al., 2024; Lee et al., 2026). Complementary research on knowledge awareness and query-level uncertainty provides signals for estimating the model's need for external assistance (Kadavath et al., 2022; Ferrando et al., 2025; Chen et al., 2026). QASG selects a query-dependent continuous strength for a single preselected skill adapter, without additional controller training.

3 Query-Adaptive Skill Gating

3.1 Skill Representation and Query-Level Control

Let x denote an input query, c its retrieved textual skill document, and f_θ a backbone LLM with frozen parameters θ. A skill-conditioned hypernetwork g_ϕ maps c to a collection of low-rank updates for a set of target modules $\mathcal{M}$:

$$\Delta_c = g_\phi(c) = \{\Delta_c^m\}_{m\in\mathcal{M}}, \qquad \Delta_c^m = B_c^m A_c^m. \tag{1}$$

Here, $A_c^m \in \mathbb{R}^{r\times d_{\text{in}}^m}$ and $B_c^m \in \mathbb{R}^{d_{\text{out}}^m\times r}$ are the generated LoRA factors, and r is the LoRA rank. The resulting adapter encodes the skill in parameter space, allowing the model to use it without including the skill text in the answer-generation prompt.

We instantiate g_ϕ with the Doc-to-LoRA architecture (Charakorn et al., 2026) and train it on skill-conditioned demonstrations while keeping the backbone frozen. The trained hypernetwork is then frozen before applying QASG; its training objective and configuration are given in Appendix B.

QASG controls the contribution of this adapter through a query-dependent scale $s(x) \in [0, 1]$. For each target module with base weight W^m, the effective weight and model prediction are

$$W'^m(x) = W^m + s(x)\Delta_c^m, \qquad \hat{y} = f_{\theta'(x)}(x), \tag{2}$$

where $\theta'(x)$ contains the updated module weights and leaves all other parameters unchanged. The scale $s(x)$ controls the amplitude of the generated update, with $s = 0$ recovering the base model and $s = 1$ applying the full effective adapter.

The generated LoRA factors A_c^m and B_c^m depend on the retrieved skill c and can be cached across queries that retrieve it. QASG determines $s(x)$ for each query before answer generation, using model uncertainty and adapter sensitivity without access to the ground-truth answer. This scale is shared across all adapted modules and remains fixed throughout the answer.

3.2 Uncertainty and Sensitivity Estimation

To determine query-level injection strength, QASG measures both the model's uncertainty and the selected adapter's sensitivity. As illustrated in Figure 2, the uncertainty probe runs with the adapter disabled to assess the base model's unaided state, while the sensitivity probe measures the adapter's response to the query.

Predictive uncertainty estimation. How can we assess whether the base model can handle a query before generating its answer and without access to ground truth? Prior work suggests that language models exhibit some awareness of their own knowledge gaps (Kadavath et al., 2022; Ferrando et al., 2025). Building on these findings, we elicit a short, query-specific self-assessment with the adapter disabled. The predictive distributions along this assessment provide a query-level uncertainty signal that reflects the base model's unaided state. The model is prompted to assess the query without solving it along three dimensions: the knowledge required to solve it and the model's familiarity with that knowledge, the main difficulty, and the step most likely to go wrong. The first dimension probes whether the requisite knowledge is available at all, while the other two force the model to localize where its reasoning may break rather than issue a global verdict. Constraining

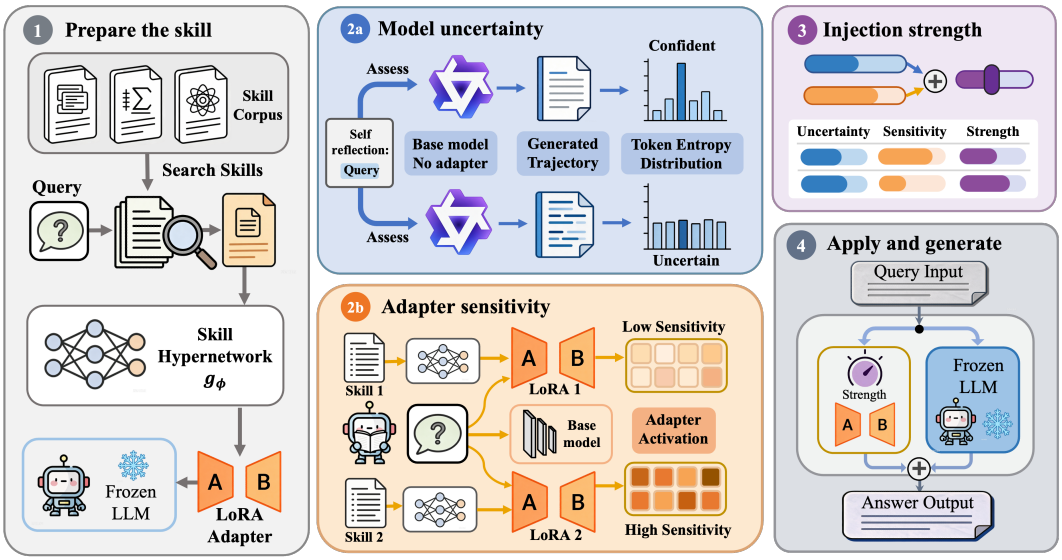


Figure 2: Overview of Query-Adaptive Skill Gating (QASG). (1) A hypernetwork converts the retrieved skill document into a LoRA adapter. (2a) Model uncertainty is estimated from the mean predictive entropy along a short self-reflection generated with the adapter disabled. (2b) Adapter sensitivity measures the adapter's activation magnitude relative to the matched base computation. (3) Uncertainty sets the baseline injection level and governs the extent of sensitivity-based refinement, with stronger adapter responses limiting additional amplification. (4) The resulting query-specific scale controls the adapter's contribution to the frozen LLM and remains fixed throughout answer generation.

the elicited text to these task-specific contents keeps the predictive distributions along the trajectory concentrated on the concepts and operations the query depends on. The template appears in Appendix A.1.

Every position of the elicited assessment carries a full next-token distribution, from which we can read the model's uncertainty directly. Let a denote the generated reflection and T_x the number of valid reflection positions. At the t-th such position, $p_t(v) = p_\theta(v \mid P(x), a_{<t})$ is the next-token distribution under the reflection prompt $P(x)$, where $a_{<t}$ denotes the full generated prefix preceding that position. We quantify how diffuse this distribution is using its predictive entropy:

$$H_t(x) = -\sum_{v \in \mathcal{V}} p_t(v) \log p_t(v), \tag{3}$$

where $\mathcal{V}$ denotes the model's vocabulary. Knowledge-related uncertainty appears at scattered concept-introducing positions (Appendix G) that recur across the three dimensions, so averaging token entropies over the whole trajectory pools these localized signals while suppressing position-specific lexical noise:

$$E(x) = \frac{1}{T_x} \sum_{t=1}^{T_x} H_t(x). \tag{4}$$

Using the mean rather than a sum normalizes for trajectory length, keeping $E(x)$ comparable across queries regardless of how long the reflection runs. A larger $E(x)$ indicates greater hesitation during self-assessment; this training-free signal directly reflects the model's need for skill assistance.

Adapter sensitivity estimation. While $E(x)$ estimates the need for assistance, it does not indicate how strongly the selected skill affects the model. Even when the model needs help, an already-strong adapter response can mean that further amplification risks overshooting and disrupting computation, whereas a weak response leaves room for more injection. We therefore characterize the adapter's response through its activation relative to the base computation, obtained in a single training-free forward pass using the adapter as generated by the hypernetwork, with no answer generation. Let

$\mathcal{I}(x)$ denote the set of valid token–module pairs at the selected readout layer $\ell^\star$, restricted to adapted modules $m \in \mathcal{M}$. For each pair i associated with module m, define $A_i = A_c^m$, $B_i = B_c^m$, and $W_i = W^m$. The adapter contribution is

$$\delta_i(x) = B_i A_i h_i(x), \tag{5}$$

where $h_i(x)$ is the input to both the adapter and its matched base projection W_i. The same adapter output can represent different degrees of intervention depending on the magnitude of the base computation it modifies. We therefore define adapter sensitivity as the mean relative activation magnitude, normalizing the magnitude of each adapter output by that of its matched base output:

$$L(x) = \frac{1}{|\mathcal{I}(x)|} \sum_{i \in \mathcal{I}(x)} \frac{\|\delta_i(x)\|_2}{\|W_i h_i(x)\|_2 + \epsilon}, \qquad \epsilon > 0. \tag{6}$$

The norm is taken over output features. Each ratio expresses intervention magnitude on the scale of the corresponding base computation, and averaging aggregates these local measurements into a query-level signal. QASG uses $L(x)$ to restrain further amplification when the adapter responds strongly and to allow more injection when it does not. Probe settings are detailed in Appendix A.

3.3 Uncertainty-Conditioned Skill Gating

The two probes inform different parts of the injection decision: $E(x)$ estimates the base model's need for assistance, while $L(x)$ measures how strongly the selected adapter already responds. We first align them using validation-set percentiles in $[0, 1]$, $u_E(x) = \hat{F}_E(E(x))$ and $u_L(x) = \hat{F}_L(L(x))$. The empirical CDFs $\hat{F}_E$ and $\hat{F}_L$ remain fixed during evaluation and deployment.

Injection strength should increase with estimated need, but a strong adapter response calls for less additional amplification. For uncertain queries, sensitivity helps determine whether further amplification is appropriate. For confident queries, intervention should remain limited, so sensitivity has less influence on the scale. We therefore use u_E as the baseline and let $1 - u_L$ add gain in proportion to u_E. QASG computes the injection scale as

$$s(x) = \tanh\big(u_E(x) + \lambda u_E(x)\big(1 - u_L(x)\big)\big), \qquad \lambda \geq 0. \tag{7}$$

The resulting scale remains fixed throughout answer generation. Section 5.3 examines the relationship between uncertainty, sensitivity, and injection strength empirically.

4 Experimental Setup

We evaluate QASG against textual skill prompting, fixed-strength adapter injection, LatentSkill, and alternative adaptive controls. We also ablate model uncertainty and adapter sensitivity, analyze how these signals relate to injection strength, and assess both backbone sizes.

Skill internalization dataset. Existing skill benchmarks such as SRA-Bench evaluate whether a model can apply a skill at test time, but they do not supply the supervised traces needed to train a skill-conditioned hypernetwork: examples that bind a reusable procedure to a worked solution demonstrating how the skill is actually applied. To our knowledge, we release the first large-scale public training corpus for skill-conditioned hypernetworks: 11,347 verified reasoning traces over 533 skills drawn from the SRA-Bench skill library (Su et al., 2026). Each record is a triple of a skill document, a newly generated question, and a complete solution demonstrating the skill in action—spanning mathematical reasoning, theorem application, clinical calculation, logical reasoning, and code generation. To make the traces trustworthy rather than self-consistent, we separate question generation, solution generation, and verification: coding solutions are checked by executing against generated unit tests, and non-coding solutions are independently solved by a second model and admitted only when answers agree, with disagreements routed through a single review round. This quality-controlled pipeline yields a common supervision source for both our hypernetwork and the LatentSkill baseline, isolating how a skill is used from how it is learned. Corpus statistics and the full construction protocol appear in Appendices C and C.1.

Benchmarks and data splits. We use the 26,262 skill documents and 3,970 benchmark questions from SRA-Bench. We evaluate five skill families: competition mathematics with CHAMP (Mao et al., 2024), theorem-grounded question answering with TheoremQA (Chen et al., 2023), clinical score calculation with MedCalc-Bench (Khandekar et al., 2024), formal reasoning with LogicBench (Parmar et al., 2024), and code generation with BigCodeBench (Zhuo et al., 2025) (abbreviated as CodeBench in tables). These tasks vary in both the base model's accuracy and the form of external procedure required.

We reserve 100 questions, 20 per benchmark, for signal calibration and hyperparameter analysis, then hold all settings fixed when evaluating on the remaining 3,870. Training questions are generated separately and kept disjoint from validation and evaluation questions. The 533 supervised skills are shared with evaluation, so the main experiment assesses generalization to new questions involving those skills.

Metrics. We report accuracy using task-specific evaluation rules: extracted-answer exact match for CHAMP and LogicBench; numerical agreement within a 2% relative tolerance for numerical answers in TheoremQA and MedCalc-Bench; normalized exact match for True/False answers in TheoremQA; and execution-based functional correctness for BigCodeBench. The *Avg* column is the unweighted mean of the five benchmark accuracies.

Baselines. We compare QASG with textual and parameter-based skill use, fixed-strength injection, and alternative adaptive controls. *No adapter* uses the base model alone, while *TextSkill* includes the retrieved skill in the input prompt. *LatentSkill* (Yu et al., 2026) is trained on the same reasoning traces as our hypernetwork, with other settings following the original work. *Uniform* applies the adapter at $s = 0.5$, while *Best fixed scale* selects the best fixed injection scale separately on each benchmark's validation questions in the main evaluation, testing whether query-level adaptation improves over per-benchmark fixed-strength tuning. For held-out evaluation, the fixed scale is selected using only validation questions from the other four benchmarks (Appendix D). For adaptive control, *Internal Confidence (IC)* uses the query-level uncertainty estimator of Chen et al. (2026), which aggregates confidence across layers and token positions without generating a response. $P(\mathrm{Yes})$ *Shift* measures the adapter-induced change in the affirmative probability under a binary self-reflection prompt. *Logit shift* measures the KL divergence between the base and adapted next-token distributions.

QASG and its scale-control baselines share the same synthesized adapter within each backbone setting. All alternative control signals are mapped to validation-set percentiles and then passed through $\tanh$, using the same calibration procedure as QASG. Detailed baseline implementations and prompt templates are provided in Appendix A.

Implementation details. We use the base versions of Qwen3-4B and Qwen3-8B as two model-size settings (Yang et al., 2025). Each setting uses a frozen backbone. QASG and its scale-control baselines use a common frozen hypernetwork; LatentSkill uses its own fine-tuned skill compiler. All skill-using methods receive the same document retrieved by BM25 from the full skill library, without access to gold answers or skill labels at inference time.

The sensitivity readout layer $\ell^\star = 20$ and refinement strength $\lambda = 0.4$ were fixed a priori. The layer-wise and coefficient sweeps provide diagnostic analyses of these settings and were not used to select the reported configuration. The gate is fixed during final answer generation and introduces no learned parameters. The self-reflection probe uses at most 200 generated tokens. Its prompt, entropy computation, probe settings, decoding configuration, and hardware are listed in Appendix A. The methods are compared at the recorded operating points; Uniform denotes the specified $s = 0.5$ baseline.

5 Results and Analysis

5.1 Overall Performance

QASG attains the highest average accuracy on both model sizes, leading on four of five benchmarks in each setting (Table 1). Uniform applies half of the generated adapter update to every query ($s = 0.5$), yet QASG improves average accuracy by 12.2 to 13.7 percentage points using the same

Table 1: Accuracy across five benchmarks and two base-model sizes, averaged over three runs that differ only in the hypernetwork training seed. Avg is the unweighted mean of the five benchmark accuracies. Bold and underlining mark the highest and second-highest reported values within each model setting. All methods share the same skill source; QASG and its scale-control baselines share the generator.

Model	Method	CHAMP	TheoremQA	MedCalc	LogicBench	CodeBench	Avg
Qwen3-4B	No adapter	0.414	0.424	0.216	0.673	0.233	0.392
	Uniform	0.452	0.490	0.252	0.710	0.268	0.434
	Best fixed scale	0.506	0.497	0.372	0.726	0.274	0.475
	Internal Confidence	0.478	0.472	0.288	0.683	0.258	0.436
	$P(\text{Yes})$ Shift	0.493	0.488	0.322	0.710	0.275	0.458
	Logit shift	0.465	0.461	0.305	0.677	0.255	0.433
	TextSkill	<u>0.571</u>	<u>0.542</u>	**0.502**	<u>0.758</u>	<u>0.293</u>	<u>0.533</u>
	LatentSkill	0.483	0.478	0.350	0.720	0.270	0.460
	QASG (ours)	**0.601**	**0.588**	<u>0.494</u>	**0.783**	**0.314**	**0.556**
Qwen3-8B	No adapter	0.463	0.470	0.256	0.708	0.283	0.436
	Uniform	0.504	0.512	0.300	0.748	0.325	0.478
	Best fixed scale	0.562	0.556	0.432	0.766	0.339	0.531
	Internal Confidence	0.532	0.525	0.338	0.718	0.313	0.485
	$P(\text{Yes})$ Shift	0.548	0.545	0.375	0.750	0.335	0.511
	Logit shift	0.517	0.514	0.355	0.713	0.310	0.482
	TextSkill	<u>0.635</u>	<u>0.602</u>	**0.559**	<u>0.800</u>	<u>0.355</u>	<u>0.590</u>
	LatentSkill	0.540	0.535	0.410	0.760	0.330	0.515
	QASG (ours)	**0.667**	**0.652**	<u>0.550</u>	**0.827**	**0.378**	**0.615**

Table 2: Component ablations on Qwen3-4B, averaged over three hypernetwork-training seeds. The study removes adapter sensitivity, model uncertainty, or their coupling. Δ is the difference from full QASG in percentage points.

Method	CHAMP	TheoremQA	MedCalc	LogicBench	CodeBench	Avg	Δ
QASG w/o L	0.535	0.521	0.403	0.750	0.291	0.500	-5.6
QASG w/o E	0.522	0.503	0.425	0.735	0.275	0.492	-6.4
QASG w/o coupling	<u>0.552</u>	<u>0.540</u>	0.448	<u>0.765</u>	<u>0.300</u>	<u>0.521</u>	-3.5
QASG (full)	**0.601**	**0.588**	**0.494**	**0.783**	**0.314**	**0.556**	—

adapter. This gap suggests that uniformly reducing injection strength is insufficient to match the benefits of query-adaptive control. QASG also exceeds the validation-selected best fixed scale by 8.1 to 8.4 points, winning all five benchmarks against both fixed-strength baselines. Together, these comparisons support adapting intervention strength to each query.

QASG exceeds the alternative adaptive baselines Internal Confidence, the affirmative-probability shift, and the logit shift by 9.8 to 10.4 points over their strongest result. Against existing state-of-the-art approaches, QASG beats LatentSkill by 9.6 to 10.0 points and textual skill prompting by 2.3 to 2.5 points. This comparison is meaningful because QASG carries the skill as a compact adapter with no context overhead, yet outperforms supplying the full skill document as prompt text. On MedCalc-Bench, TextSkill slightly leads QASG by under one point. This likely reflects that the dataset contains much fine-grained procedural detail that is difficult to compress into a compact adapter. Even on this benchmark, QASG remains the strongest adapter-based method and beats the best fixed adapter strength by 11.8 to 12.2 points. Paired-bootstrap comparisons are reported in Appendix F.

5.2 Ablation Study

Table 2 removes adapter sensitivity, model uncertainty, or their multiplicative coupling, separating the information each signal supplies from the way the two are combined. Removing either signal

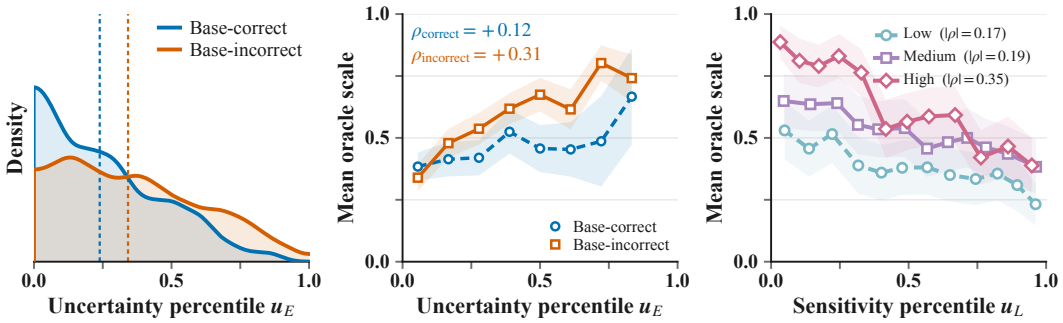


Figure 3: Complementary signals for query-adaptive skill control. (a) Uncertainty distributions for base-correct and base-incorrect queries; dashed lines mark group medians. (b) Oracle scale versus uncertainty within each correctness group. (c) Oracle scale versus sensitivity within low-, medium-, and high-uncertainty groups. Shading in (b,c) denotes 95% confidence intervals for binned means; the legend in (c) reports absolute Spearman correlations $|\rho|$.

hurts accuracy on every benchmark, with drops of 5.6 and 6.4 points respectively, and the two removals are not interchangeable. An uncertainty score alone cannot see how strongly the adapter engages, while a sensitivity score alone lacks the base model's predictive state. Keeping both signals but dropping their multiplicative coupling still leaves a gap of 3.5 points, confirming that the coupling itself contributes. Each ablation's scale parameter is independently selected to maximize accuracy on the 100-question validation set used to calibrate QASG's signals; exact formulations appear in Appendix A.

5.3 Signal Analysis

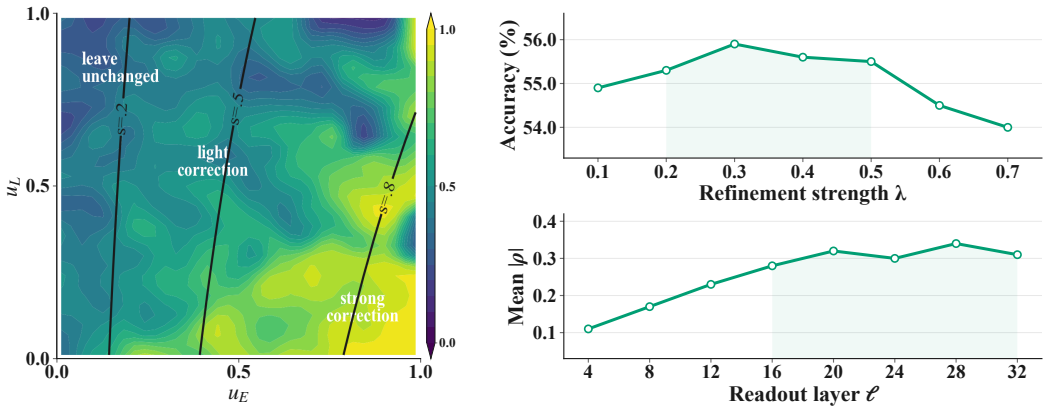
Uncertainty and assistance needs. The two signals provide complementary evidence for query-adaptive skill control. As illustrated in Figure 3(a), base-incorrect queries exhibit higher uncertainty overall than base-correct queries, indicating that self-reflection exposes differences relevant to the model's need for assistance. Within the base-incorrect group, higher uncertainty is also associated with larger oracle injection scales, as shown in Figure 3(b); the oracle scale is defined in Appendix E. Across most uncertainty levels, base-incorrect queries have higher oracle scales than base-correct queries, indicating a greater need for skill assistance. These observations support uncertainty as an informative signal for allocating skill assistance.

Sensitivity and injection strength. Sensitivity further distinguishes successful injection strengths among queries with similar uncertainty. Figure 3(c) shows that the oracle scale generally decreases as sensitivity increases, supporting more restrained amplification of strongly responding adapters. This association is strongest in the high-uncertainty group, where $|\rho| = 0.35$, compared with 0.19 and 0.17 in the medium- and low-uncertainty groups. Sensitivity therefore provides particularly useful guidance when the model is more uncertain, lending empirical support to the uncertainty-conditioned coupling in QASG.

Joint signal-space analysis. Figure 4(a) brings these relationships together. Higher oracle scales broadly concentrate in regions with high uncertainty and low sensitivity, while lower uncertainty or stronger adapter responses tend to favor smaller scales. The QASG contours follow this broad directional pattern, showing how the gate translates the complementary signals into differentiated levels of skill intervention.

5.4 Hyperparameter Analysis

Refinement strength. The upper plot in Figure 4(b) provides a post hoc sensitivity analysis of the refinement coefficient λ on Qwen3-4B. QASG maintains an average accuracy of 55.3%–55.9% for $\lambda \in [0.2, 0.5]$, indicating stable performance across neighboring configurations. The preset value



(a) (b)

Figure 4: Joint signal analysis and hyperparameter sensitivity. (a) Color encodes the oracle scale across uncertainty and sensitivity percentiles u_E and u_L; black contours show QASG injection scales from Equation 7. (b) Hyperparameter analyses. The upper plot shows average accuracy on Qwen3-4B as the refinement coefficient λ varies; the lower plot shows the effect of the sensitivity readout layer. Mean $|\rho|$ is the benchmark-averaged absolute Spearman correlation between adapter sensitivity and the per-query oracle scale on the validation set.

$\lambda = 0.4$ lies within this region, suggesting that the method does not depend on a narrowly tuned coefficient. Accuracy declines at larger values, supporting a moderate refinement strength.

Sensitivity readout layer. The lower plot in Figure 4(b) examines the association between adapter sensitivity and the oracle scale across readout layers. The mean absolute Spearman correlation remains between 0.28 and 0.34 across layers 16–32, with the preset layer $\ell^\star = 20$ lying within this region. The association therefore persists across several neighboring readout choices, supporting sensitivity estimation from upper-middle-layer responses. Appendix E provides the diagnostic definition and layer-wise values.

5.5 Computation and Deployment

QASG adds two per-query probes: a self-reflection capped at 200 generated tokens (185 on average) and a single non-decoding sensitivity forward pass. For Qwen3-4B on one NVIDIA H20 GPU, with greedy decoding, cached adapters, and a shared maximum answer length, mean end-to-end latency across the full evaluation set is 19.1 s for Uniform, 20.1 s for TextSkill, and 22.9 s for QASG. The uncertainty and sensitivity probes take 3.5 s and 0.3 s, respectively. By excluding the retrieved skill document from the final-answer prompt, QASG reduces mean input length from 1556 to 112 tokens, a 92.8% reduction. Its end-to-end latency increase over TextSkill is 2.8 s, below the combined probe time of 3.8 s. Overall, QASG improves accuracy over TextSkill by 2.3 percentage points (Table 1) at 14% higher latency while substantially reducing final-answer context use.

6 Conclusion

This work establishes query-adaptive strength control as an effective way to improve the use of internalized skills. QASG couples self-reflection uncertainty with adapter sensitivity to match assistance to the model's needs while restraining further amplification of strongly responding adapters. This lightweight design requires neither ground-truth answers at inference time nor additional controller training. We also contribute a verified skill–solution corpus for learning reusable skill adapters. Across five diverse benchmarks, QASG achieves the highest average accuracy among the evaluated methods, including TextSkill, LatentSkill, and alternative adaptive controls. Its gains over fixed-strength policies with shared adapters demonstrate the value of adapting how a skill is applied to each query. These findings point toward LLM agents that can more effectively regulate when and how strongly to draw on external skills.

AI USE STATEMENT

In this work, we used generative AI tools to assist with synthetic training-data generation, dataset cleaning and reformatting, translation, and the interpretation of results, as well as to provide feedback on research methodology and experiments.

We did not use generative AI tools to develop theoretical models or conceptual frameworks, formulate mathematical claims, provide key arguments for mathematical proofs, write proofs, propose or refine research hypotheses, implement the proposed method, or conduct qualitative or thematic data analysis.

Additionally, we used generative AI tools to assist with manuscript drafting, organization, and language editing.

We take responsibility for the final content of this work, including text, claims, data, code, and other artifacts produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

The method is specified in Equations 1–7. Appendix A collects the inference configuration, Appendix B gives the hypernetwork training objective and configuration, and Appendix C reports the dataset composition. The evaluation uses a shared skill source and task-specific answer checking. We will publicly release all code and training data upon acceptance of the paper.

REFERENCES

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-LoRA: Instant transformer adaption. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 7485–7514. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/charakorn25a.html.

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-LoRA: Learning to instantly internalize contexts, 2026. URL https://arxiv.org/abs/2602.15902.

Lihu Chen, Gerard de Melo, Fabian Suchanek, and Gael Varoquaux. Query-level uncertainty in large language models. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 34353–34372, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/3a07c3a67cfe50d3236b71fb674c7f30-Paper-Conference.pdf.

Tong Chen, Hao Fang, Patrick Xia, Xiaodong Liu, Ben Van Durme, Luke Zettlemoyer, Jianfeng Gao, and Hao Cheng. Generative adapter: Contextualizing language models in parameters with a single forward pass. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 27450–27470, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/447708674d7f30492ca5338b36b07d0c-Paper-Conference.pdf.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. TheoremQA: A theorem-driven question answering dataset. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7889–7901, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.489. URL https://aclanthology.org/2023.emnlp-main.489/.

Javier Ferrando, Oscar Obeso, Senthooran Rajamanoharan, and Neel Nanda. Do I know this entity? Knowledge awareness and hallucinations in language models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 77965–78000, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/c1c44e46358e0fb94dc94ec495a7fb1a-Paper-Conference.pdf.

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, 2024. URL https://arxiv.org/abs/2307.13269.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know, 2022. URL https://arxiv.org/abs/2207.05221.

Nikhil Khandekar, Qiao Jin, Guangzhi Xiong, Soren Dunn, Serina S Applebaum, Zain Anwar, Maame Sarfo-Gyamfi, Conrad W Safranek, Abid A Anwar, Andrew Zhang, Aidan Gilson, Maxwell B Singer, Amisha Dave, Andrew Taylor, Aidong Zhang, Qingyu Chen, and Zhiyong Lu. MedCalc-Bench: Evaluating large language models for medical calculations. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 84730–84745. Curran Associates, Inc., 2024. doi: 10.52202/079017-2690. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/99e81750f3fdfcaf9613db2dbf4bd623-Paper-Datasets_and_Benchmarks_Track.pdf.

Seungeon Lee, Soumi Das, Manish Gupta, and Krishna P. Gummadi. LoRA on the go: Instance-level dynamic LoRA selection and merging. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 39583–39601, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1837. URL https://aclanthology.org/2026.acl-long.1837/.

Yewei Liu, Xiyuan Wang, Yansheng Mao, Yoav Gelbery, Haggai Maron, and Muhan Zhang. SHINE: A scalable in-context hypernetwork for mapping context to LoRA in a single pass, 2026. URL https://arxiv.org/abs/2602.06358.

Zefang Liu and Jiahua Luo. AdaMoLE: Fine-tuning large language models with adaptive mixture of low-rank adaptation experts, 2024. URL https://arxiv.org/abs/2405.00361.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. SKILL0: In-context agentic reinforcement learning for skill internalization, 2026. URL https://arxiv.org/abs/2604.02268.

Yujun Mao, Yoon Kim, and Yilun Zhou. CHAMP: A competition-level dataset for fine-grained analyses of LLMs' mathematical reasoning capabilities. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13256–13274, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.785. URL https://aclanthology.org/2024.findings-acl.785/.

Oleksiy Ostapenko, Zhan Su, Edoardo Ponti, Laurent Charlin, Nicolas Le Roux, Lucas Caccia, and Alessandro Sordoni. Towards modular LLMs by building and reusing a library of LoRAs. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 38885–38904. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/ostapenko24a.html.

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, Maohao Shen, Vishy Tirumalashetty, George Lee, Jiawei Han, Tomas Pfister, and Chen-Yu Lee. SkillOS: Learning skill curation for self-evolving agents, 2026. URL https://arxiv.org/abs/2605.06614.

Shuai Pan, Yixiang Liu, Jiaye Gao, Te Gao, Weiwen Liu, Jianghao Lin, Zhihui Fu, Jun Wang, Weinan Zhang, and Yong Yu. SkillMAS: Skill co-evolution with LLM-based multi-agent system, 2026. URL https://arxiv.org/abs/2605.09341.

Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. LogicBench: Towards systematic evaluation of logical reasoning ability of large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13679–13707, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.739. URL https://aclanthology.org/2024.acl-long.739/.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty (eds.), *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 487–503, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.39. URL https://aclanthology.org/2021.eacl-main.39/.

Weihang Su, Jianming Long, Qingyao Ai, Qiaozhi He, Yichen Tang, Changyue Wang, Yiteng Tu, Yingbo Wang, and Yiqun Liu. Skill retrieval augmentation for agentic AI, 2026. URL https://arxiv.org/abs/2604.24594.

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, 2023. URL https://arxiv.org/abs/2305.16291.

Hanqing Wang, Bowen Ping, Shuo Wang, Xu Han, Yun Chen, Zhiyuan Liu, and Maosong Sun. LoRA-Flow: Dynamic LoRA fusion for large language models in generative tasks. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12871–12882, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.695. URL https://aclanthology.org/2024.acl-long.695/.

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, 2026. URL https://arxiv.org/abs/2605.16986.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. EvolveR: Self-evolving LLM agents through an experience-driven lifecycle, 2026. URL https://arxiv.org/abs/2510.16079.

Xun Wu, Shaohan Huang, and Furu Wei. Mixture of LoRA experts. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 47302–47318, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/ce806d8b4bf38cd92d483e5a0490d983-Paper-Conference.pdf.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin

Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Aofan Yu, Chenyu Zhou, Tianyi Xu, Zihan Guo, Rong Shan, Zhihui Fu, Jun Wang, Weiwen Liu, Yong Yu, Weinan Zhang, and Jianghao Lin. LatentSkill: From in-context textual skills to in-weight latent skills for LLM agents, 2026. URL https://arxiv.org/abs/2606.06087.

Tianyi Zhang and Zhonghao Qi. Skill-to-LoRA: From using skills to learning behaviors for token-efficient LLM agents, 2026. URL https://arxiv.org/abs/2606.16769.

Xuan Zhao, Haonan He, Qingyu Yang, Minglei Li, Jingqi Ye, Zelin Tan, Bo Wan, and Peng Ye. Parametric skills, 2026. URL https://arxiv.org/abs/2606.30015.

Jiapeng Zhu, Jianxiang Yu, Yibo Zhao, Chengcheng Han, Qi Gu, Xunliang Cai, Xiang Li, and Weining Qian. Skill0.5: Joint skill internalization and utilization for out-of-distribution generalization in agentic reinforcement learning, 2026. URL https://arxiv.org/abs/2605.28424.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen GONG, James Hoang, Armel Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, Binyuan Hui, Niklas Muennighoff, David Lo, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 66602–66656, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a6a90bcc2aa470c3871b2d39a67d26e8-Paper-Conference.pdf.

A Inference and Evaluation Configuration

Table 3 summarizes the QASG configuration shared by the 4B and 8B runs. Sensitivity estimation uses the adapter as generated by the hypernetwork; final generation uses the scale in Equation 7.

Table 3: Inference and evaluation configuration.

Setting	Value
Backbones	Qwen3-4B / Qwen3-8B base (non-Instruct)
Skill retriever / documents per query	BM25 / 1
Sensitivity readout layer $\ell^\star$	20 (one-based), `down_proj`
Sensitivity denominator stabilizer ϵ	10^{-10}
Refinement strength λ	0.4
Self-reflection maximum length / template	200 tokens / Appendix A.1
Trajectory aggregation / uncertainty signal	Mean entropy estimate
Signal normalization	Fixed-reference empirical percentiles
Probe decoding	Greedy
Final decoding / maximum length	Greedy / 2048 tokens
Adapted projections / LoRA rank	`down_proj` only / $r = 8$

Sensitivity computation. Sensitivity is computed in one forward pass over the query using the adapter as generated by the hypernetwork, with the settings in Table 3. Activations are read at layer $\ell^\star = 20$, which was fixed a priori; a diagnostic layer scan is reported in Appendix E. In Equation 6, $h_i(x)$ is obtained from this forward pass, and the numerator is the norm of the adapter branch output $B_i A_i h_i(x)$. Fixed LoRA multipliers are absorbed into the effective factors. Padding is excluded, and relative activations are averaged over the valid query-token–projection pairs at the selected layer. Adapter sensitivity measures the relative magnitude of the adapter response and does

not capture its direction or determine its effect on answer correctness. At a given uncertainty level, higher sensitivity reduces the additional amplification assigned by the gate, reflecting a conservative control choice without assuming that stronger responses are harmful or weaker responses beneficial.

Token masking. Padding is excluded using the attention mask; if no padding token is defined, EOS serves as the padding token. Special tokens are excluded from signal statistics, and generation stops at EOS or the 200-token budget (Section A.1). In the entropy expressions, $t = 1, \ldots, T_x$ indexes the retained reflection positions, so T_x counts only positions included in the average. Each predictive distribution remains conditioned on the full original generated prefix.

A.1 Self-Reflection Prompt

The uncertainty probe uses the following template with the adapter disabled; {`question`} denotes the input question.

Self-Reflection Prompt

```
Below is a question. Do not solve it. Please assess it briefly in at
    most 200 words, covering the following three dimensions:

(1) Knowledge: What domain and specific concepts, formulas, logical
    rules, or APIs are needed to solve this question? List the most
    important ones and state how familiar you are with each.

(2) Difficulty: In your judgment, where does the main difficulty of
    this question lie--the setup, a key transformation, an edge case,
    or the final answer format?

(3) Error-prone step: Which single step are you most likely to get
    wrong, and what kind of mistake would you make there?

Give your honest assessment. Question: {question}
```

Decoding stops at EOS or 200 generated tokens, independently of the prompt's word limit. Entropy is averaged over the generated trajectory, excluding prompt tokens and padding. The assessment is not graded as an answer. Appendix G provides illustrative token-level visualizations from supplementary self-reflection traces.

Self-reflection entropy captures token-level predictive uncertainty during reflection and serves as an empirical proxy for the model's need for skill assistance, without assuming a direct or calibrated correspondence to uncertainty about the final answer. This proxy may also be influenced by wording and query domain, and should therefore not be interpreted as a standalone measure of whether the base model can answer correctly.

A.2 Evaluation Details

Baseline instantiations. No adapter uses $s = 0$, Uniform uses $s = 0.5$, Best fixed scale selects the best fixed scale per benchmark in the main evaluation, chosen on that benchmark's validation questions to maximize validation accuracy, and TextSkill places the skill document at the top of the standard task prompt. For held-out evaluation, the fixed scale is selected using only validation questions from the other four benchmarks (Appendix D). Internal Confidence (IC) directly follows the training- and generation-free method of Chen et al. (2026). IC and $P(\text{Yes})$ Shift both place the complete benchmark input inside a dedicated binary self-reflection prompt: *"Can you answer the following query accurately? Respond only with Yes or No. Query:* {`query`}*"* Thus, for open-ended and coding tasks alike, the measured "Yes" is a response to this meta-question rather than a candidate continuation of the raw task query. At the position immediately before response generation, we define $P(\text{Yes}) = \exp(z_{\text{Yes}})/(\exp(z_{\text{Yes}}) + \exp(z_{\text{No}}))$, where z_{Yes} and z_{No} are the pre-softmax logits for the "Yes" and "No" tokens, respectively. IC obtains this probability from hidden states across layers and the last query-token positions in one base-model forward pass, then aggregates the values with Gaussian-decay weights centered on the final layer and final token. $P(\text{Yes})$ Shift is our

intervention-aware adaptation: we run the same meta-question once with the base model and once with the adapter injected, generate no answer tokens in either pass, and use $\Delta P(\text{Yes}) = P(\text{Yes} \mid \text{injected}) - P(\text{Yes} \mid \text{base})$ at the final prompt position. Logit shift is the KL divergence from the base to the injected next-token distribution at the query, $\text{KL}(p_{\text{base}} \,\|\, p_{\text{injected}})$. Each of these raw signals is first mapped to percentiles on the same fixed validation reference used for QASG (Section 3.3). IC, $P(\text{Yes})$ Shift, and Logit shift use $s = \tanh(\gamma u)$, where u denotes the calibrated percentile and $\gamma \geq 0$ is a method-specific scaling coefficient selected independently on the same validation set used for QASG signal calibration. These independently tuned coefficients allow each baseline to adjust its overall injection strength, rather than restricting it to the fixed mapping $s = \tanh(u)$; the coefficients remain fixed during evaluation. Because $P(\text{Yes})$ denotes confidence rather than uncertainty, IC feeds $1 - P(\text{Yes})$ into this mapping so that less confident queries receive stronger injection, matching the direction shared by all control baselines. The sensitivity-free ablation uses $s = \tanh(\alpha\, u_E)$, and the uncertainty-free ablation uses $s = \tanh(\beta(1-u_L))$; in each case the scalar is selected on the same validation set used for QASG signal calibration. The uncoupled combination uses $s = \tanh(u_E + \lambda'(1 - u_L))$ with λ' selected independently on the validation set.

LatentSkill. For the main evaluation, we fine-tune LatentSkill's skill compiler on the same 11,347 verified training trajectories over 533 skills used for our hypernetwork. The training, validation, and evaluation question splits are unchanged. We use the backbone corresponding to each model-size setting and the same skill documents and benchmark evaluation protocol. All other method-specific training and inference settings follow LatentSkill (Yu et al., 2026). Its generated adapters are applied using its original inference procedure. In the held-out experiments of Appendix D, LatentSkill's skill compiler is fine-tuned only on training traces from the other four benchmarks, excluding all training traces from the target benchmark, as for our hypernetwork.

Answer checking. For TheoremQA, evaluation depends on the answer type. Numerical answers are checked using a 2% relative tolerance, whereas True/False answers are evaluated by exact match after answer extraction and normalization. The numerical tolerance does not apply to Boolean answers. MedCalc-Bench also uses a 2% relative tolerance for numerical answers. Answer extraction, normalization, zero-valued-reference handling, and unit conversion follow the evaluation scripts. CHAMP and LogicBench use extracted-answer exact match, and BigCodeBench uses execution-based checking with the released BigCodeBench harness.

B Hypernetwork Training and Skill Resources

The hypernetwork follows the document-to-adapter architecture of Charakorn et al. (2026). Given a skill document c, it generates adapter factors that are applied to the frozen base model. Let $\mathcal{D}_c$ contain the training pairs (q, a) associated with skill c, where q is a question and a is the retained solution trace followed by its final answer. We train the hypernetwork with the autoregressive objective

$$\mathcal{L}(\phi) = -\sum_{c} \sum_{(q,a)\in\mathcal{D}_c} \sum_{t} \log p_{\theta+g_\phi(c)}(a_t \mid q, a_{<t}). \tag{8}$$

Here, $\theta + g_\phi(c)$ denotes the base model equipped with the generated adapter under the training injection configuration described below. Gradients pass through the frozen backbone to update only ϕ. The query-dependent QASG gate is not used in this training objective; it is applied after training, with both the backbone and hypernetwork frozen. The supervision contains 11,347 retained traces over 533 skills, with data construction described in Section 4 and Appendix C.1.

Shared training configuration for QASG and scale-control baselines. Training runs for 1000 steps on 8 NVIDIA H20 GPUs with a peak learning rate of 5×10^{-6}, cosine decay to a minimum of 10^{-7} after 100 warm-up steps, and batch size 2 per GPU (effective batch 16, no gradient accumulation). Optimizer defaults and numerical precision follow the released implementation.

Only `down_proj` is adapted, with rank $r = 8$. Training fixes `lora_alpha` to $2r^{3/2} = 32\sqrt{2} \approx 45.2548$; the training injection scale and checkpoint-selection rule follow the Doc-to-LoRA protocol (Charakorn et al., 2026).

DeepSeek-V4-Flash generates the training solutions, including the CoT and final answer, before verification. The accepted solution itself supplies the hypernetwork target; there is no subsequent teacher-generation stage. Claude outputs, test cases, and review messages are used for verification and are not appended to the training response. The construction protocol and prompt templates are given in Appendices C.1 and C.2, respectively.

C Per-Benchmark Data Statistics

Table 4: Benchmark questions, validation/evaluation split, supervised skills, and retained training traces by benchmark. A shared validation set of 20 questions per benchmark is held out from the question bank; Eval. reports the remaining evaluation questions. Skills count procedures with retained hypernetwork-training demonstrations; the full skill corpus contains 26,262 documents.

Benchmark	Questions	Val.	Eval.	Skills	Traces
CHAMP	223	20	203	80	1,600
TheoremQA	747	20	727	256	5,120
MedCalc-Bench	1,100	20	1,080	44	880
LogicBench	760	20	740	15	600
CodeBench	1,140	20	1,120	138	3,147
Total	3,970	100	3,870	533	11,347

The corpus contains 26,262 skill documents. The 533 skills in Table 4 are the supervised subset exercised by the evaluation questions; the remaining documents broaden the candidate set. A shared validation set of 100 questions is held out from the question bank; the reported accuracies are measured on the remaining 3,870 evaluation questions. Question-level train/validation/evaluation separation is maintained, while skills are shared. This permits evaluation of new uses of supervised skills within a larger skill library.

Evaluation questions and their provided answers and skill labels are unchanged. The question bank is split into validation and evaluation subsets; generated training questions are handled separately. The construction protocol below checks training candidates against both subsets before admission.

C.1 Training-Data Construction and Verification

The protocol uses separate calls for question generation, solution generation, and verification. No generated answer is treated as an existing benchmark gold answer. A retained sample consists of the conditioning skill, generated question, complete DeepSeek solution trace, and its final answer.

1. Generate a skill-conditioned question. DeepSeek-V4-Flash receives the skill document and a task-format specification. It produces a self-contained question without a solution. CHAMP-style questions require a competition-mathematics answer; TheoremQA-style questions specify the theorem-based task and answer type; MedCalc-Bench-style questions provide a fictional clinical case with the quantities, units, and rounding instructions required for a calculation; LogicBench-style questions state the premises, query, and permitted answer labels; BigCodeBench-style questions specify a Python interface, dependencies, and expected behavior. The format specification contains no held-out question or answer. Candidates with missing information, conflicting requirements, or no identifiable use of the target skill are rejected. Duplicate candidates and overlaps with validation or evaluation questions are excluded before retention; a corrected question is a new candidate and restarts the pipeline.

2. Generate the complete solution. In a fresh call, DeepSeek-V4-Flash receives the skill and question and produces an explanatory CoT and final answer under the shared solution prompt. For code, the final answer is a complete implementation of the required interface. Unsolvable, ambiguous, or malformed outputs are rejected. This call supplies the candidate training response; a later verification stage cannot replace just its final answer while retaining an incompatible trace.

3a. Execute coding solutions. For each generated coding question, Claude 5 Sonnet constructs a separate reference implementation and executable tests from the question specification, without seeing the candidate code. Test assertions cover specified ordinary cases, boundary cases, and error behavior where defined; expected values must be justified from the specification, rather than copied from candidate outputs. The reference implementation is first checked against these assertions. Inconsistent or incomplete test packages cause the candidate to be discarded. A valid test package is frozen before testing the DeepSeek implementation in an isolated execution environment with the declared dependencies and execution limits. A candidate is retained only when every test passes without an execution error or timeout; failed candidates are discarded. Tests for generated questions are distinct from the official benchmark tests used for evaluation. Passing this check establishes correctness on the constructed tests.

3b. Independently solve non-coding questions. For the other four benchmarks, Claude 5 Sonnet receives the skill, question, and answer-format requirements in a fresh call under the same solution prompt, without the DeepSeek trace or answer. It returns its own solution and final answer, or flags an ill-posed question. The verifier is not asked to approve a supplied candidate answer. Both valid final answers are then compared by the following rules:

- CHAMP and LogicBench use exact match after answer extraction and formatting normalization. LogicBench labels are mapped to the question's specified label set. Missing or ambiguous answers do not count as matches.
- Numerical TheoremQA and MedCalc-Bench answers are converted to the units requested by the question and compared within a 2% relative tolerance, using the independently generated Claude answer as the comparison reference. A zero reference requires an exactly zero candidate after normalization. This reference is a model-generated comparison value, not a benchmark gold answer.
- Non-numerical TheoremQA answers use the explicitly requested answer type, with normalized exact match for categorical or structured answers. Cases without a supported comparison rule are rejected rather than assigned an arbitrary numerical tolerance.

If the answers agree and neither solution flags a problem with the question, the complete DeepSeek trace is retained; agreement serves as a consistency filter before retention.

4. Review disagreement once. If two valid non-coding answers disagree, a separate Claude review call receives the question and both complete solutions. It identifies a concrete discrepancy in assumptions, calculation, units, or logic, or reports that the question is ambiguous or the disagreement unresolved. Ambiguous or unresolved cases are discarded. For a discrepancy with an actionable explanation, DeepSeek regenerates the entire solution using the review feedback. A fresh Claude call independently solves the original question again under the same solution prompt, without seeing the candidates or review. The same comparison rule is applied to these new answers. Only a valid, agreeing pair permits retention of the regenerated DeepSeek trace. A second disagreement, an invalid output, or a newly identified ambiguity causes rejection; there is no repeated retry until agreement. Agreement after review is recorded separately from agreement on the first independent check.

5. Store the accepted record. Verification records retain the question, skill identifier, candidate and verifier outputs, comparison result or test report, review history, and acceptance or rejection reason. The hypernetwork receives only the skill and accepted question–solution pair. Counts by benchmark and acceptance route are obtained from the retained records. The reported corpus size must correspond to the records actually used for training, rather than the number of generation attempts.

C.2 Construction and Verification Prompts

The placeholders below are filled with the skill text, generated question, and the corresponding task-format specification. Each independent call starts with a fresh context. JSON outputs are parsed and validated before routing to the next stage; missing fields and unparseable outputs are rejected. Here, `status` takes one of the stated literal values; alternative values shown in a template are not a

single combined output string. The solution prompt is shared by DeepSeek-V4-Flash and Claude 5 Sonnet, so that agreement reflects model agreement rather than prompt differences.

Prompt 1: Question Generation (DeepSeek-V4-Flash)

```
Create one original, self-contained training question that requires
    the procedure in the supplied skill. Follow the task specification
    exactly. Include the information, assumptions, units, answer format
    , or programming interface needed to determine a solution. Avoid
    unrelated details and do not copy an existing benchmark item. Do
    not provide a solution, a worked example revealing the answer, or
    the final answer. If the skill and task specification cannot
    support a well-posed question, return status "reject" with a reason
    .

Skill: {skill_document}. Task specification: {task_specification}.

Return JSON with keys status ("ok" or "reject"), question,
    answer_format, skill_use, and reason. The skill_use field briefly
    states why the skill is required without solving the question. For
    coding tasks, include the required function signature and allowed
    dependencies in the question. For rejected outputs, leave the
    question empty.
```

Prompt 2: Independent Solution Generation

```
Solve the supplied question using the skill. No proposed answer or
    prior solution is provided. Give a self-contained worked solution
    and a final answer consistent with it. State the assumptions needed
    ; check arithmetic, logical entailment, and units as appropriate;
    and follow the required answer format, including any units or
    rounding instructions requested by the question. For code, explain
    the approach and return a complete executable implementation as the
     final answer. Do not invent missing premises or silently change
    the question. If it is ambiguous, underspecified, or inconsistent,
    flag it instead of forcing an answer.

Skill: {skill_document}. Question: {question}. Answer format: {
    answer_format}.

Return JSON with keys status ("solved" or "invalid_question"),
    solution, final_answer, and reason. On an invalid question, leave
    the solution and final answer empty.
```

Prompt 3: Coding Reference and Test Construction

```
Construct a reference implementation and a test package for the
    supplied Python task. You have not been given any candidate
    implementation. Derive expected behavior only from the question.
    Cover ordinary inputs, boundary cases, and error cases explicitly
    required by the specification. Explain how each expected result
    follows from the specification; do not use the candidate program or
     merely compare a program with itself as an oracle. Tests must
    assert correctness, not just successful execution. Supply the entry
     point, dependencies, fixtures, reference implementation, and
    executable tests. Do not claim that tests were executed: execution
    is performed by the caller. If the specification is ambiguous or
    reliable assertions cannot be constructed, return "reject".

Question: {question}. Environment specification: {
    execution_specification}.
```

```
Return JSON with keys status ("ready" or "reject"), reference_code,
    test_code, test_rationale, dependencies, fixtures, entry_point, and
     reason.
```

Prompt 4: Disagreement Review (Claude 5 Sonnet)

```
Review two conflicting solutions to the same question. Neither answer
    is a gold answer. Diagnose the disagreement by checking the
    question, assumptions, calculations, units, and logical steps. Do
    not choose a solution merely because it is longer or more confident
    , and do not force agreement. If the question is ill-posed, return
    "invalid_question". If you can identify and explain a concrete
    correctable error, return "retry" with actionable feedback.
    Otherwise return "unresolved". Your verdict alone does not accept a
     training sample; a retry requires a newly generated complete
    solution and another independent answer comparison.

Question: {question}. Answer format: {answer_format}. Solution A: {
    deepseek_solution}. Solution B: {claude_solution}.

Return JSON with keys status ("retry", "invalid_question", or "
    unresolved"), discrepancy, and feedback.
```

Prompt 5: Complete Solution Regeneration (DeepSeek-V4-Flash)

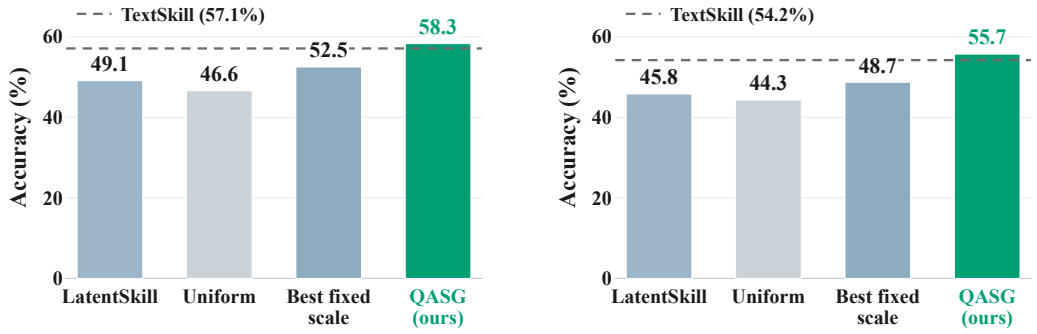
```
Re-solve the original question after considering the review feedback.
    Check the feedback against the question rather than accepting it
    automatically. Produce a new self-contained solution and its final
    answer; do not edit only the final answer of the previous trace. Do
     not change the question or add unsupported assumptions. If the
    issue cannot be resolved, return "invalid_question" with a reason.
    The regenerated answer will be compared with a fresh independent
    solution.

Skill: {skill_document}. Question: {question}. Answer format: {
    answer_format}. Previous solution: {deepseek_solution}. Review
    feedback: {review_feedback}.

Return JSON with keys status ("solved" or "invalid_question"),
    solution, final_answer, and reason. On an invalid question, leave
    the solution and final answer empty.
```

D Held-Out Generalization Evaluation

We additionally evaluate cross-dataset transfer on CHAMP and TheoremQA. For each target benchmark, both our hypernetwork and LatentSkill's skill compiler are trained only on training traces from the other four benchmarks, excluding all training traces from the target benchmark. Percentile calibration and fixed-scale selection use 80 validation questions, 20 from each of the four remaining benchmarks. Best fixed scale selects a single injection scale that maximizes the unweighted mean validation accuracy across these four benchmarks and keeps it fixed for all evaluation questions in the target benchmark. The readout layer $\ell^\star = 20$ and refinement strength $\lambda = 0.4$ retain their predefined values. Thus, the target benchmark contributes neither training traces, calibration data, nor hyperparameter choices. This tests whether the gate transfers to a benchmark that is entirely absent from training, calibration, and model selection.

Figure 5 compares QASG with LatentSkill, uniform injection, and the validation-selected best fixed scale; the dashed line reports TextSkill using the same retrieved skill document. QASG achieves the highest accuracy among all compared methods on both benchmarks, reaching 58.3% on CHAMP and 55.7% on TheoremQA. On CHAMP, LatentSkill achieves 49.1%, giving QASG an advantage of 9.2 percentage points. QASG outperforms the best fixed scale by 5.8 and 7.0 percentage points

on CHAMP and TheoremQA, respectively, and exceeds TextSkill by 1.2 and 1.5 points. These results support the generalization of query-adaptive strength control: QASG retains an advantage over both fixed-strength adapter injection and direct textual skill prompting on held-out questions. Although its absolute accuracy is lower than in the main evaluation (60.1% on CHAMP and 58.8% on TheoremQA), QASG remains the strongest method in both held-out comparisons, showing that its benefit extends to this additional evaluation.



(a) (b)

Figure 5: Cross-benchmark generalization to (a) CHAMP and (b) TheoremQA. Both the QASG hypernetwork and LatentSkill's skill compiler are trained only on traces from the other four benchmarks. Best fixed scale is selected on validation questions from those four benchmarks. QASG achieves the highest accuracy at 58.3% and 55.7%, respectively; dashed lines indicate the TextSkill reference.

E Readout-Layer Analysis

Scale diagnostic. The diagnostic figures and the readout-layer scan use a per-query oracle scale computed from a scale sweep. For each query with its skill, the frozen base model generates the answer once per candidate scale s on an 11-point grid over $[0, 1]$ $(0.0, 0.1, \ldots, 1.0)$, with the adapter merged at strength s and all other settings identical. Let $\mathcal{S}_{\text{correct}}(x)$ denote the candidate scales that yield a correct answer for query x. We define the oracle scale as their median, taking the lower of the two middle values when $|\mathcal{S}_{\text{correct}}(x)|$ is even, so that $s^\star(x) = \text{median}\,\mathcal{S}_{\text{correct}}(x)$ is the lower median, summarizing the central tendency of successful injection strengths for that query. Queries that no candidate scale answers correctly have no oracle scale and are excluded from the diagnostic plots and correlations; they remain part of the accuracy evaluation, which does not depend on the diagnostic. The median is computed separately for each query, and the reported correlations relate each signal to these per-query summaries of successful injection strength. The descriptive figures in Section 5.3 compute it on the evaluation set, while the layer scan below computes it on the validation set for diagnostic analysis; gold answers are therefore used for scoring and hyperparameter analysis only and never enter inference.

Table 5: Readout-layer scan. Values are benchmark-averaged absolute Spearman correlations with the scale diagnostic.

Layer	4	8	12	16	20	24	28	32
Avg. $\lvert\rho\rvert$	0.11	0.17	0.23	0.28	0.32	0.30	0.34	0.31

The sensitivity readout layer was fixed a priori at layer 20 (one-based). The layer-wise correlations in Table 5 are reported as a diagnostic analysis and were not used to select the readout layer; the adopted configuration therefore does not necessarily maximize this diagnostic. Sensitivity uses the token/projection aggregation in Equation 6.

The lower plot in Figure 4(b) visualizes these layer-wise associations, as discussed in Section 5.4.

F Statistical Analysis

Paired question bootstrap. For the main table and component study, all methods are evaluated on the same questions with the same skill. We first average each question's correctness over the three hypernetwork-training seeds. No adapter and TextSkill are independent of hypernetwork training; with the backbone, prompts, and greedy decoding configuration fixed, we use their per-question correctness directly without seed averaging. We then perform 10,000 bootstrap iterations, resampling questions with replacement within each benchmark and using identical sampled indices across methods. In each iteration, we recompute per-benchmark accuracies, their unweighted macro-average, and paired differences between methods. Standard errors are the standard deviations of these bootstrap statistics and quantify question-sampling uncertainty conditional on the evaluated models. For the robustness study, we report the standard deviation across question-bank re-splits.

Reported differences. For the macro-average accuracies in Table 1, the paired-bootstrap standard errors of the Avg column range from 0.87 to 0.94 percentage points, with QASG at 0.89 on Qwen3-4B and 0.87 on Qwen3-8B. The headline differences have standard errors of 0.7, 1.1, and 1.1 percentage points on Qwen3-4B and 0.8, 1.0, and 1.1 points on Qwen3-8B for QASG versus TextSkill, Uniform, and the best fixed scale, respectively; each of these differences exceeds three standard errors. At both model sizes, QASG improves accuracy on four of five benchmarks relative to TextSkill and on all five benchmarks relative to both Uniform and the best fixed scale. In the component study, the standard errors of the three Δ values in Table 2 are 1.0, 1.1, and 0.9 points for w/o L, w/o E, and w/o coupling, respectively. The magnitude of each difference exceeds three bootstrap standard errors.

Supplementary robustness analysis. On Qwen3-4B, we examine the robustness of the preset QASG configuration to different validation/evaluation splits. The sensitivity readout layer $\ell^\star = 20$ and refinement strength $\lambda = 0.4$ were fixed a priori and remain unchanged across all 20 splits; neither parameter is re-selected in this analysis. For each split, we randomly hold out 100 questions (20 per benchmark) as validation and use the remaining 3,870 questions for evaluation. Validation and evaluation are therefore disjoint within each split. For comparison, the best fixed scale is re-selected separately on each benchmark's validation questions for that split. Across the 20 splits, the macro-averaged accuracy has mean 0.554 with a standard deviation of 0.006 across splits (range 0.543–0.562). QASG exceeds the best fixed scale in all 20 splits, with gains of 7.4–9.1 percentage points. These results support the robustness of the preset QASG configuration to the evaluated question splits, with stable accuracy and a consistent advantage over the validation-selected best fixed scale.

G Token-Level Uncertainty in Self-Reflection

We visualize how the self-reflection prompt exposes uncertainty while the model identifies task-related knowledge. Figures 6–8 present three examples from 50 supplementary diagnostic traces spanning the five benchmarks. These traces were recorded with a 300-token generation cap. Each figure reproduces the complete recorded assessment, including its knowledge, difficulty, and error-prone-step sections; none of the three generations reached the cap. The examples retain the full surrounding trajectory.

Background colors encode the recorded predictive entropy at the position preceding each displayed token. All three figures share a linear color scale from 0 to 3, without smoothing, clipping, or per-example rescaling. Special control tokens are hidden, while the complete assessment text is preserved. The quoted entropy values refer to individual tokens rather than whole-phrase averages.

Uncertainty at a stated knowledge limitation. In the radiation-physics example (Figure 6), the assessment distinguishes familiar inverse-square-law reasoning from less familiar gamma-ray emission branching ratios. At the latter phrase, the entropy at *gamma* is 2.34, followed by 1.20, 1.42, and 1.31 at *ray*, *emission*, and *branching*. In contrast, the *law* token in "Familiar with inverse-square

law" has entropy 0.15. The structured assessment thus places a localized distributional signal alongside a concrete statement of the knowledge requirement, rather than reducing the entire query to a verbal confidence rating.

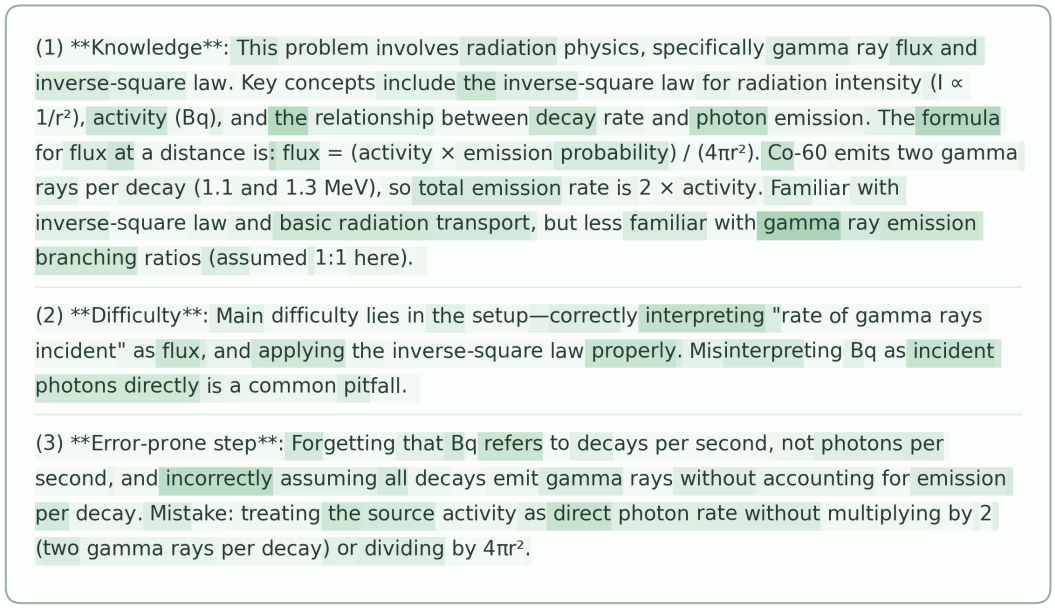


Figure 6: Token-level uncertainty in a complete TheoremQA self-reflection on radiation physics. Darker green indicates higher predictive entropy on the shared 0–3 color scale. Elevated entropy appears around gamma-ray emission branching ratios, with the surrounding knowledge, difficulty, and error-prone-step sections retained for context.

A technical term can be uncertain before its continuation becomes predictable. In the renal-function example (Figure 7), entropy reaches 2.44 at *renal* and 2.26 at *criteria*. The word *glomerular* is particularly informative: its initial token, `gl`, has entropy 2.43, whereas the following `omer` and `ular` tokens each have entropy below 0.01. This contrast shows why fluent completion of a technical term need not imply low uncertainty at every position that introduces it. It also motivates examining the assessment trajectory rather than a single final-token distribution.

Uncertainty around a task-related logical distinction. In the logical-reasoning example (Figure 8), the model lists concepts needed to evaluate an inference. Entropy is elevated at *categorical* (2.65), *statements* (1.34), and *unsupported* (1.93), placing the signal near the distinction between types of premises and justified conclusions. The prompt makes these task-specific concepts explicit, providing interpretable locations at which to inspect predictive uncertainty.

Together, these examples illustrate the intended role of self-reflection: it elicits task-related continuations whose predictive distributions can inform the uncertainty signal. These qualitative observations complement the component ablations; QASG still uses the trajectory-averaged uncertainty together with adapter sensitivity to determine the intervention.

H Case Studies

We randomly select three samples from Qwen3-4B runs and compare our QASG method with text-based skill injection. In all three cases, QASG produces the correct final answer while TextSkill does not. Both methods act on the same skill: QASG correctly completes the critical step, whereas TextSkill, although it invokes the skill, misapplies it and fails. Given the same skill knowledge, QASG puts it to markedly more effective use. For each task, we present TextSkill, Uniform, and QASG in that order, followed by a joint analysis.

(1) **Knowledge**: This question falls within clinical medicine, specifically internal medicine and acute kidney injury (AKI) evaluation. Key concepts include fluid balance, renal function, hypertension, diabetes, and differential diagnosis of oliguria. Important concepts: glomerular filtration rate (GFR), AKI criteria (e.g., KDIGO), fluid overload signs (e.g., edema, low skin turgor), and drug-induced nephrotoxicity (e.g., from new antihypertensives). Familiarity is moderate with GFR estimation and common causes of reduced urine output.

(2) **Difficulty**: The main difficulty lies in the **differential diagnosis**—distinguishing between prerenal azotemia (due to dehydration or hypovolemia), acute tubular necrosis, or intrinsic renal disease, especially given the patient's comorbidities and medication history.

(3) **Error-prone step**: Misattributing the reduced urine output to dehydration without considering medication side effects (e.g., diuretics or new antihypertensives) or underestimating the significance of subtle signs like jugular venous distension and edema. A common mistake is overlooking drug-induced causes, leading to an incorrect focus on volume depletion.

Figure 7: Token-level uncertainty in a complete MedCalc-Bench self-reflection on renal function. Darker green indicates higher predictive entropy on the shared 0–3 color scale. The initial `gl` token of *glomerular* has substantially higher entropy than its continuation, illustrating variation within a single technical term.

(1) **Knowledge**: This question involves logical reasoning, specifically deductive inference from premises. Key concepts include logical validity, categorical statements, and the identification of unsupported conclusions. No formulas or APIs are needed. Familiarity is moderate—basic logic rules (e.g., modus tollens, affirming the consequent) are known, but real-world assumptions (e.g., "friends are intelligent") are vague and context-dependent.

(2) **Difficulty**: The main difficulty lies in the setup—interpreting ambiguous or non-logical generalizations (e.g., "friends are normally intelligent") as valid premises. These are probabilistic, not deterministic, making valid deduction challenging.

(3) **Error-prone step**: Misapplying generalizations to individuals. One is likely to conclude that because Chris is not intelligent, he lacks a trait (like brown hair), or that Eric, not going to college, is therefore not intelligent—ignoring the lack of logical connection between college attendance and intelligence. A common mistake is assuming correlation implies causation or that absence of a trait in one context implies absence in another.

Figure 8: Token-level uncertainty in a complete LogicBench self-reflection. Darker green indicates higher predictive entropy on the shared 0–3 color scale. Elevated entropy appears around task-related concepts such as *categorical statements* and *unsupported conclusions*.

H.1 C1: Material Implication with a Negated Antecedent

Instance: `logicbench_00445`.

Task. Emma auditions for a choir. If she cannot sing, she cannot join it. Which conclusion follows? The options are (1) she cannot sing or will not join; (2) Emma cannot sing or will not join; (3) he can sing or will not join; and (4) she can sing or will not join. The answer is choice_4.

Skill: Material Implication (`logicbench_008`). The equivalence $P \to Q \equiv \neg P \vee Q$ converts a conditional into a disjunction by negating its antecedent and retaining its consequent. Let S denote Emma's ability to sing and J her ability to join. Any negation already present in the antecedent must be handled explicitly: $\neg S \to \neg J$ becomes $S \vee \neg J$. A candidate answer must preserve both these truth conditions and the original subject.

TextSkill: choice_2 (incorrect). We are given the conditional "If she cannot sing, she cannot join the choir." Applying the **material implication** rule, a conditional $P \to Q$ is logically equivalent to $\neg P \vee Q$.

Step 1: Identify the antecedent and consequent
Let S denote Emma's ability to sing and J her ability to join the choir. The premise is $\neg S \to \neg J$.

Step 2: Apply the material implication rule
Substituting $P = \neg S$ and $Q = \neg J$ into $P \to Q \equiv \neg P \vee Q$ gives:

$$\neg(\neg S) \vee \neg J$$

Removing the double negation, this becomes:

$$\neg S \vee \neg J$$

That is, **Emma is not able to sing or she will not join the choir**. This is exactly **choice_2**.

Step 3: Reject the remaining options
- **Choice 1** has the same logical form as choice_2 but leaves the subject implicit; choice_2 restates the premise's subject and is the more faithful conversion.
- **Choice 3** introduces "he" and is inconsistent with the subject of the premise.
- **Choice 4** ("She is able to sing or she will not join the choir") negates "cannot sing" into "can sing," reversing the antecedent stated in the premise. The disjunction must retain the negated antecedent, so choice_4 is not a valid application of the rule.

✓ **Final Answer:**
choice_2: Emma is not able to sing or she will not join the choir.

Uniform: choice_2 (incorrect). We need to convert the conditional about Emma into a statement that matches one of the available options.

Step 1: Represent the premise
Let S mean that Emma can sing and J mean that she can join the choir. The given statement is

$$\neg S \to \neg J.$$

Both propositions concern Emma, so the selected option must also refer to her.

Step 2: Identify the excluded situation
The conditional imposes a restriction on singing ability and choir membership. Interpret this restriction as excluding their joint occurrence: Emma cannot both be able to sing and join the choir. The condition can therefore be written as

$$\neg(S \wedge J).$$

Step 3: Convert the restriction into a disjunction
Applying De Morgan's law gives

$$\neg(S \wedge J) \equiv \neg S \vee \neg J.$$

Thus, at least one of the following must hold: Emma cannot sing, or she will not join the choir.

Step 4: Compare the options

Choices 1 and 2 express this disjunction. Choice 2 explicitly names Emma, so select it as the clearest match. Choice 3 refers to "he," while choice 4 uses S instead of $\neg S$ and therefore differs from the derived restriction.

Final Answer:
choice_2: Emma is not able to sing or she will not join the choir.

QASG: choice_4 (correct). Let S denote that Emma can sing and J denote that she can join the choir. The statement "If she is unable to sing, then she will not be able to join the choir" is therefore represented as $\neg S \to \neg J$.
To express this conditional as a disjunction, we apply the material implication rule $P \to Q \equiv \neg P \vee Q$. Here, the antecedent is $P = \neg S$ and the consequent is $Q = \neg J$. Negating the entire antecedent while retaining the consequent gives

$$\begin{aligned} \neg S \to \neg J &\equiv \neg(\neg S) \vee \neg J && \text{(material implication)} \\ &\equiv S \vee \neg J && \text{(double negation).} \end{aligned}$$

The double negation cancels, so the first disjunct is S, while the second remains $\neg J$. The equivalent statement is thus "She is able to sing or she will not join the choir," with "she" referring to Emma.
Choices 1 and 2 instead express $\neg S \vee \neg J$, which is not entailed by the premise: when $S = J = \text{true}$, the conditional holds but this disjunction is false. Choice 3 changes the subject to "he." Only choice_4 preserves both the equivalent logical form and the original subject.
Therefore, the answer is **choice_4: She is able to sing or she will not join the choir.**

Analysis. TextSkill applies the same material-implication rule but fails to cancel the double negation in the antecedent, retaining $\neg S$ in the conclusion. With $S = J = \text{true}$, the premise holds, but its answer, choice_2, $\neg S \vee \neg J$, is false.
Uniform reaches the same wrong answer through a different error: it replaces the conditional with mutual exclusion, $\neg(S \wedge J)$. The actual forbidden assignment is $\neg S \wedge J$, whose negation gives $S \vee \neg J$.
QASG preserves the conditional's truth conditions by resolving the double negation in the antecedent, obtaining $S \vee \neg J$. Thus, TextSkill mishandles double negation, while Uniform misidentifies the situation excluded by the premise.

H.2 C2: Exchanging Entities and Their Properties

Instance: `logicbench_00246`.

Task. Friends are usually supportive and loyal. John is not supportive, while Mary is not loyal. Does this imply that Mary is not supportive and John is not loyal? The answer is no.

Skill: Default Reasoning with Multiple Concurrent Defaults (`logicbench_018`). When two individuals violate different default properties, each exception remains attached to its stated individual and property. An exception concerning John's support does not transfer to Mary's support or John's loyalty. Under the default interpretation, unexcepted properties remain in force. In particular, the two stated negative facts cannot be exchanged between individuals to obtain the queried negative conclusion.

TextSkill: yes (incorrect). This is a case of **default reasoning**: "friends are usually supportive and loyal" states a default, and the passage supplies exceptions to it. Under default reasoning, an exception defeats the default it contradicts, so a defeated default can no longer be assumed to hold.

Step 1: Identify the default and the exceptions

> "John and Mary had been close friends for many years. We know that friends are usually supportive and normally demonstrate unwavering loyalty. However, it became apparent that **John was not living up to his usual supportive self**, while **Mary, unfortunately, seemed to be lacking in loyalty**."

- John's failure to be supportive $\to$ an exception that defeats the **supportive** default.
- Mary's lack of loyalty $\to$ an exception that defeats the **loyal** default.

Step 2: Apply the defeat
Once a default is defeated, it holds no longer. Because the supportive default is defeated, support cannot be assumed for anyone; because the loyal default is defeated, loyalty cannot be assumed for anyone.

Step 3: Draw the conclusion

With both defaults defeated, the negative properties follow for both individuals: **Mary is not supportive and John is not loyal**.

✓ **Final Answer:**
Yes, it means that Mary isn't supportive and John isn't loyal.

Uniform: yes (incorrect). We are asked whether the two stated exceptions imply that Mary is not supportive and John is not loyal.

Step 1: Represent the properties and exceptions
Let $S(p)$ mean that person p is supportive and $L(p)$ mean that person p is loyal. Friends are usually described as having both properties, while the explicit exceptions are

$$\neg S(\text{John}), \qquad \neg L(\text{Mary}).$$

These facts identify one missing property for each person.

Step 2: Relate support and loyalty
Because the description presents support and loyalty together as characteristics of friendship, treat them as two equivalent indicators of the same relationship:

$$S(p) \leftrightarrow L(p).$$

This means that being supportive implies being loyal, and being loyal implies being supportive.

Step 3: Apply the implications to each exception
Taking contrapositives yields

$$\neg S(p) \rightarrow \neg L(p), \qquad \neg L(p) \rightarrow \neg S(p).$$

John is not supportive, so the first implication gives that John is not loyal. Mary is not loyal, so the second gives that Mary is not supportive. Each stated exception therefore determines the person's other missing property.

Step 4: Combine the conclusions
The two deductions give

$$\neg S(\text{Mary}) \wedge \neg L(\text{John}),$$

which is exactly the claim in the question. Both requested negative properties have now been derived from the exceptions and the relationship between the traits.

Final Answer:
Yes, Mary is not supportive and John is not loyal.

QASG: no (correct). Let's break this down step by step.
Step 1 – Identify the given statements.
- Friends are usually supportive and demonstrate unwavering loyalty.
- John was not living up to his usual supportive self.
- Mary seemed to be lacking in loyalty.

Step 2 – Understand what "not living up to his usual supportive self" means. This specifically refers to John's *supportive* behavior, not necessarily his loyalty. So we know John is not supportive right now, but we cannot conclude anything about his loyalty from that statement.
Step 3 – Understand what "lacking in loyalty" means. This specifically refers to Mary's *loyalty*, not necessarily her support. So we know Mary is not loyal right now, but we cannot conclude anything about her supportive behavior from that statement.
Step 4 – Check the question. The question asks: "Does this mean that Mary isn't supportive and John isn't loyal?" We have:
- John's lack of support → but no info about John's loyalty.
- Mary's lack of loyalty → but no info about Mary's support.

Step 5 – Logical conclusion. We cannot infer either Mary's lack of support or John's lack of loyalty from the given information.
Therefore, the answer is no.
Therefore, the answer is no.

Analysis. TextSkill invokes the same default-reasoning rule but lets each exception defeat the default globally, transferring the exceptions between individuals. The check does not require treating "usually" as an exceptionless universal rule.
Uniform instead introduces the unsupported biconditional $S(p) \leftrightarrow L(p)$. The contraposition steps are valid only under this added assumption; two default tendencies do not establish a strict equivalence between the properties. The same assignment—John loyal but unsupportive, Mary supportive but disloyal—also refutes this argument.
QASG keeps each exception attached to its stated individual and property and therefore rejects the exchange. Mary can remain supportive and John loyal while both stated exceptions hold, falsifying the queried conclusion without contradicting the passage.

H.3 C3: Overgeneralizing from Sampled Intervals on Negative Inputs

Instance: `theoremqa_00705`.

Task. Does $2^x + 1/x = -4$ have a real solution? The answer is True.

Skill: Intermediate Value Theorem (`theoremqa_293`). If a function is continuous on a closed interval and its endpoint values lie on opposite sides of a target value, it attains that value somewhere inside the interval. The argument establishes existence, not necessarily uniqueness. For a function containing $1/x$, the interval must avoid zero; checking several inputs on only part of the domain cannot rule out a solution elsewhere.

TextSkill: False (incorrect). We are given the equation:

$$2^x + \frac{1}{x} = -4$$

We want to determine whether this equation has a real solution. Since $f(x) = 2^x + 1/x$ is continuous on its domain, we apply the **Intermediate Value Theorem**: a solution exists exactly where f crosses -4.

Step 1: Analyze the domains
- 2^x is defined for all real x, and it is always positive: $2^x > 0$ for all x.
- $\frac{1}{x}$ is defined only when $x \neq 0$.

So the domain is $x \in \mathbb{R} \setminus \{0\}$.

Step 2: Consider the left-hand side (LHS)
We have:

$$2^x + \frac{1}{x}$$

We know:
- $2^x > 0$ for all real x.
- $\frac{1}{x}$ can be positive or negative depending on the sign of x.

So the sum could be negative only if $\frac{1}{x}$ is sufficiently negative to offset the positive 2^x.

Step 3: Try negative values of x
Let $x < 0$. Then $\frac{1}{x} < 0$, so the sum could be negative.
Let's try $x = -1$:

$$2^{-1} + \frac{1}{-1} = 0.5 - 1 = -0.5 > -4$$

Too high (not equal to -4).
Try $x = -2$:

$$2^{-2} + \frac{1}{-2} = 0.25 - 0.5 = -0.25 > -4$$

Still greater than -4.
Try $x = -10$:

$$2^{-10} + \frac{1}{-10} = \frac{1}{1024} - 0.1 \approx 0.000976 - 0.1 = -0.099 > -4$$

Still greater than -4.

As $x \to -\infty$:
- $2^x \to 0^+$
- $\frac{1}{x} \to 0^-$

So the sum approaches 0 from below: $2^x + \frac{1}{x} \to 0^-$
The samples at -10, -2, and -1 span the intervals $[-10, -2]$ and $[-2, -1]$; on both, the endpoint values lie above -4, and the limit at $-\infty$ stays above -4 as well. Since f is continuous, by the Intermediate Value Theorem a crossing of -4 would require endpoint values on opposite sides of it, so the crossing condition is never met on the negative side.

Step 4: Positive x?
If $x > 0$, then:
- $2^x > 0$
- $\frac{1}{x} > 0$

So the sum is **positive**, which cannot equal -4.

Conclusion:
Since f stays above -4 on every interval we examined, the equation has **no real solution**.
Therefore, the answer is False.

Uniform: False (incorrect). We want to determine whether

$$2^x + \frac{1}{x} = -4$$

has a real solution. Define $f(x) = 2^x + 1/x$ and consider the positive and negative parts of its domain separately.

Step 1: Check the domain and positive inputs
The reciprocal requires $x \neq 0$. For $x > 0$, both 2^x and $1/x$ are positive, so

$$f(x) > 0 > -4.$$

No positive input can solve the equation. Any real solution must therefore lie on the negative side.

Step 2: Obtain a necessary inequality
For $x < 0$, rearrange the equation as

$$-\frac{1}{x} = 4 + 2^x.$$

The exponential is strictly positive for every real input. Consequently, any solution must satisfy

$$-\frac{1}{x} > 4.$$

This inequality can be used to restrict the interval in which a solution might occur.

Step 3: Restrict the candidate interval
Multiplying both sides by x gives

$$-1 > 4x,$$

and dividing by 4 yields

$$x < -\frac{1}{4}.$$

Thus, it is sufficient to check $(-\infty, -1/4)$; negative values closer to zero do not satisfy this necessary condition.

Step 4: Bound the function on this interval
Write $x = -a$, where $a > 1/4$. Then

$$\frac{1}{a} < 4, \qquad \frac{1}{x} = -\frac{1}{a} > -4.$$

Adding the positive exponential term gives

$$f(x) = 2^x + \frac{1}{x} > \frac{1}{x} > -4.$$

Therefore, throughout the candidate interval, the left-hand side stays strictly above the target value. It cannot reach -4 anywhere on this interval.

Step 5: Combine the domain checks
Positive inputs have been excluded by their sign, and zero is outside the domain. Every possible negative solution must satisfy the bound from Step 3, but Step 4 rules out that entire interval. No admissible input remains.

Final Answer:
The equation has no real solution. Therefore, the answer is **False**.

QASG: True (correct). We are asked to determine whether the equation:

$$2^x + \frac{1}{x} = -4$$

has a solution.

Step 1: Analyze the behavior of the function
Define the function:

$$f(x) = 2^x + \frac{1}{x}$$

We are solving $f(x) = -4$.

Step 2: Domain restrictions
- $x \neq 0$, because $\frac{1}{x}$ is undefined at $x = 0$.

So we consider $x \in (-\infty, 0) \cup (0, \infty)$.

Step 3: Behavior on $x > 0$
- $2^x > 0$ for all $x > 0$
- $\frac{1}{x} > 0$ for $x > 0$

So $f(x) > 0$ for all $x > 0$
Therefore, $f(x) = -4$ is **impossible** for $x > 0$

Step 4: Behavior on $x < 0$
Let $x < 0$. Then:
- $2^x > 0$ (since exponential is always positive)
- $\frac{1}{x} < 0$

So $f(x) =$ positive $+$ negative
We want to know if this sum can be -4
Let's analyze:
Let $x = -a$, where $a > 0$
Then:

$$f(-a) = 2^{-a} + \frac{1}{-a} = \frac{1}{2^a} - \frac{1}{a}$$

We want:

$$\frac{1}{2^a} - \frac{1}{a} = -4$$

Multiply both sides by a (positive):

$$\frac{a}{2^a} - 1 = -4a$$

$$\frac{a}{2^a} + 4a = 1$$

$$a\left(\frac{1}{2^a} + 4\right) = 1$$

Now, define $g(a) = a\left(\frac{1}{2^a} + 4\right)$

We want to know if $g(a) = 1$ has a solution for $a > 0$
Note:
- $\frac{1}{2^a} > 0$, so $g(a) > 4a$
- So $g(a) > 4a$

We want $g(a) = 1$, so $4a < 1 \Rightarrow a < 0.25$
Try $a = 0.1$:
- $2^{0.1} \approx 1.0718$
- $\frac{1}{2^{0.1}} \approx 0.933$
- $g(0.1) = 0.1(0.933 + 4) = 0.1 \times 4.933 = 0.4933$

Too low.
Try $a = 0.2$:
- $2^{0.2} \approx 1.1487$
- $\frac{1}{2^{0.2}} \approx 0.8706$
- $g(0.2) = 0.2(0.8706 + 4) = 0.2 \times 4.8706 = 0.9741$

Still less than 1.
Try $a = 0.21$:
- $2^{0.21} \approx 1.153$
- $\frac{1}{2^{0.21}} \approx 0.867$
- $g(0.21) = 0.21(0.867 + 4) = 0.21 \times 4.867 \approx 1.022$

Now greater than 1.
So $g(a)$ increases from below 1 to above 1 as a increases from 0.2 to 0.21.
Therefore, by Intermediate Value Theorem, there exists some $a \in (0.2, 0.21)$ such that $g(a) = 1$
Thus, there exists $x = -a$ such that:

$$2^x + \frac{1}{x} = -4$$

Final Answer:
Yes, the equation has a real solution.
Therefore, the answer is True.

Analysis. TextSkill applies the same theorem but restricts the check to intervals with $x \leq -1$; on the omitted interval $(-1, 0)$, $1/x$ diverges to $-\infty$, and indeed $f(-0.1) \approx -9.067 < -4$. The crossing the theorem would certify lies in the interval TextSkill never examined.
Uniform excludes the solution interval through an algebraic error: multiplying $-1/x > 4$ by $x < 0$ must reverse the inequality, giving $-1 < 4x$ and hence $-1/4 < x < 0$. Indeed, $f(-1/4) \approx -3.159 > -4$ and $f(-1/5) \approx -4.129 < -4$; continuity on $[-1/4, -1/5]$ guarantees a real solution.
QASG, by contrast, transforms the equation and applies the theorem on $(0.2, 0.21)$, supplying a valid existence argument. TextSkill overlooks the relevant interval, while Uniform incorrectly rules it out.

I Limitations and Future Work

QASG currently controls a single retrieved skill adapter, leaving multi-skill coordination unexplored. Future work could jointly select, compose, and regulate multiple adapters while accounting for interference between skills. The gate also uses one scale across all adapted layers and decoding steps; adapting skill contributions across layers or reasoning steps could better accommodate changing assistance needs. Our evaluation covers five benchmarks and two model sizes within the Qwen3 family. Although cross-benchmark experiments provide evidence of transfer, broader validation across model families, larger backbones, and long-horizon agent tasks remains necessary. Finally, strength control may not fully recover procedural details that an adapter has failed to encode.