

TT-SI: Self-Improving LLM Agents with Test-Time Training

Anonymous ACL submission

Abstract

One paradigm of language model (LM) fine-tuning relies on creating large training datasets, under the assumption that high quantity and diversity will enable models to generalize to novel tasks after post-training. In practice, gathering large sets of data is inefficient, and training on them is prohibitively expensive; worse, there is no guarantee that the resulting model will handle complex scenarios or generalize better. Moreover, existing techniques rarely assess whether a training sample provides novel information, resulting in unnecessary costs. In this work, we explore a new Test-Time Self-Improvement (TT-SI) algorithm to create more effective and generalizable agentic LMs *on-the-fly*. TT-SI can be summarized in three steps: (i) first it identifies the samples that model struggles with (self-awareness), (ii) then generates similar examples from detected uncertain samples (self-data augmentation), and (iii) uses these newly generated samples at test-time training (self-improvement). We further explore *Test-Time Distillation* (TT-D), which leverages a stronger supervisor for targeted data generation. Empirical evaluations across different agent benchmarks demonstrate that TT-SI improves the performance with +5.48% absolute accuracy gain on average across all benchmarks and surpasses other standard learning methods more efficiently. Our findings highlight the promise of TT-SI, demonstrating the potential of self-improvement algorithms at test-time as a new paradigm for building more capable agents toward self-evolution.

1 Introduction

Recent progress in language model (LM) post-training has shown promising results across a wide range of tasks (Kumar et al., 2025) by equipping these models with explicit knowledge (Grattafiori et al., 2024; Yang et al., 2025), reasoning (Zelikman et al., 2022; Guo et al., 2025), and agentic capabilities (Zeng et al., 2024; Chen et al.,

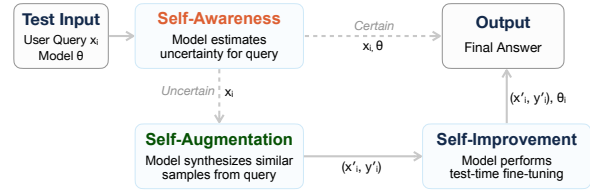


Figure 1: **TT-SI Framework.** TT-SI enables *on-the-fly* adaptation by targeting uncertain test instances during inference, following three steps: (1) **Self-Awareness:** An **Uncertainty Estimator (H)** identifies challenging samples. (2) **Self-Data Augmentation:** For each identified uncertain sample, generate one similar variant using **Data Synthesis Function (G)**. (3) **Self-Improvement: Test-Time Training (T)** applies a lightweight update using only *one generated training instance per case*.

2024b). These systems are typically trained to approximate an unknown mapping $\mathcal{F}_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ from large-scale collections of input-output pairs (x_i, y_i) , where \mathcal{X} denotes the inputs and \mathcal{Y} denotes their corresponding desired targets. In this approach, a single function F_θ attempts to cover all the relevant knowledge and generalization capability from a single dataset X , implicitly assuming that the dataset has sufficient quality, diversity, and scale to effectively learn diverse tasks. However, this learning paradigm can remain narrow and inefficient compared to actual human learning (Mitchell et al., 2018).

In contrast, humans take advantage of their background experience (similar to the pretraining stage of LMs) and exhibit remarkable efficiency during learning, often guided by self-regulated learning principles (Zimmerman, 2002) where individuals actively seek and learn from informative demonstrations (Nelson, 1990). For example, consider a student who is preparing for a college entrance exam after years of coursework. Engaging in metacognitive reflection (Flavell, 1979), the student can either broadly practice questions on various topics (e.g., algebra, history, chemistry) or strategically identify gaps in their knowledge (**self-awareness**),

collect targeted questions addressing these specific deficiencies (**self-data augmentation**), and practice them repeatedly to learn (**self-improvement**). Clearly, the second strategy is more effective and explicitly improves the required knowledge (see Appendix B for other examples).

The same inefficiency is evident in the standard LM agent fine-tuning paradigms, which train models to inductively learn general rules from training data to be applied to new, unseen test instances such as tool use or other complex agentic tasks. It involves collecting large-scale training datasets (Ouyang et al., 2022; Wang et al., 2023; Zeng et al., 2024) (either human-curated or LLM-synthesized) and fine-tuning models on them (Grattafiori et al., 2024; Zeng et al., 2024; Acikgoz et al., 2025). However, constructing these datasets is costly, often requiring days to weeks of computation and manual labor, and still provides no guarantee of effective performance and generalization after fine-tuning. Moreover, this approach treats all samples as equally necessary, overlooking redundancy and the model’s existing knowledge. Based on these deficiencies, a key open question is *whether models can be trained to acquire new skills more efficiently and self-aware, without relying on exhaustive datasets or processing large amounts of redundant information*.

Motivated by local and transductive learning (Bottou and Vapnik, 1992; Joachims, 1999) with recent advances in test-time training (TTT) (Akyurek et al., 2025), we investigate a simple, yet powerful, instance-specific self-improvement algorithm that adapts agents *on-the-fly* to each downstream task at test-time (Figure 1). TTT enables parametric models to update their weights temporarily during inference (Sun, 2023) and has recently been extended to LLMs for abstract reasoning (Akyurek et al., 2025), but remains largely under-explored for agentic tasks. Our proposed algorithm first identifies the most informative and challenging samples while discarding mastered or redundant ones, guided by the designed **Uncertainty Estimator (H)**, which reflects *self-awareness*. For each retained “necessary” test instance, the model synthesizes a set of distributionally similar samples with **Data Synthesis Function (G)** as *self-data augmentation* and performs temporary gradient updates with **Test-Time Training (T)** through *self-improvement* on these instances. We explore two different variants of our approach: Test-Time Self-Improvement (TT-SI), where the model

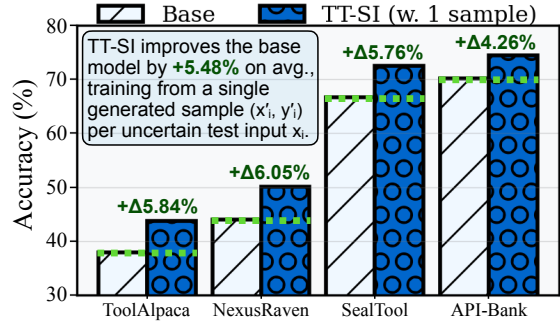


Figure 2: Δ -accuracy gains of TT-SI during test-time. TT-SI improves the baseline by +5.48% on average across ToolAlpaca (+5.84%), NexusRaven (+6.05%), SealTool (+5.76%), and API-Bank (+4.26%).

trains on self-generated samples using parameter efficient fine-tuning techniques (PEFT) (Hu et al., 2022), and Test-Time Distillation (TT-D) where adaptation is guided by supervision from samples synthesized by a more capable teacher model.

In this work, we make the first exploration of TTT with LLM agents. We demonstrate that TT-SI enables agents to adapt on-the-fly by leveraging their own uncertain predictions. With only a *single targeted training instance* per test case, TT-SI shows consistent absolute accuracy gains across four challenging agent benchmarks: +5.84% on ToolAlpaca +6.05% on NexusRaven, +5.76% on SealTool, and +4.26% on API-Bank (Figure 2). These improvements highlight that even minimal, uncertainty-guided adaptation can substantially boost performance during inference. We also find that, TT-D further extends these gains in complex and context-heavy scenarios. Compared to inductive supervised fine-tuning (SFT), TT-SI achieves higher accuracy with $68\times$ fewer samples, underscoring its efficiency without compromising effectiveness. Moreover, when training is infeasible, TT-SI with in-context learning (ICL) offers a fast and training-free alternative, outperforming other standard learning methods in similar conditions. Specifically, our main findings and contributions are:

- **(Algorithm)** We propose a three-stage algorithm for *test-time self-improvement* (TT-SI), motivated by human learning theories: (i) identify uncertain samples via a novel uncertainty estimator, (ii) generate new training instances similar to these samples, and (iii) update the model online.
- **(Analysis)** We conduct a systematic empirical study of two variants, TT-SI and TT-D, analyzing key components such as the impact of uncertain samples, learning method at test time, scaling of generated samples, and other parameter effects.

- **(Performance)** We validate that agentic LMs can self-improve during inference, even from a single training instance, and show that our framework outperforms standard inductive learning approaches under distribution-shift, achieving significant gains with orders-of-magnitude less compute through both test-time ICL and fine-tuning.

2 Preliminaries

2.1 Challenges in Inductive Fine-Tuning

The standard post-training paradigm separates training and testing: models are trained by *inductively* extracting generalizable patterns from data and subsequently evaluated on new, possibly unseen examples (Vapnik, 1999; LeCun et al., 2015; Zhang et al., 2024). Current approaches for training LMs largely follow this paradigm, relying on large-scale post-training datasets. Formally, these datasets are denoted as $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^N$, consist of N samples assumed to be independently and identically distributed (i.i.d.) according to a *training* distribution $\mathcal{P}_{\text{train}}(\mathcal{X}, \mathcal{Y})$.

Here, $x_i \in \mathcal{X}$ represents an input (e.g., a task query) and $y_i \in \mathcal{Y}$ is its corresponding desired output (e.g., a sequence of actions for an agent). The objective is to find the parameters θ of a mapping function $\mathcal{F}_\theta : \mathcal{X} \rightarrow \mathcal{Y}$, representing the agent, that minimize the empirical risk on the training data: $\hat{\mathcal{L}}_{\text{train}}(\theta) = \frac{1}{N} \sum_{i=1}^N \ell(\mathcal{F}_\theta(x_i), y_i)$, where ℓ is a predefined loss function. The foundational assumption is that if N is sufficiently large and $\mathcal{D}_{\text{train}}$ is diverse enough, the learned model \mathcal{F}_θ will generalize effectively to new, unseen inputs x drawn from a *test* distribution $\mathcal{P}_{\text{test}}(\mathcal{X})$. However, this prevailing paradigm is beset by several fundamental issues:

- **Distributional Shift:** Test distributions $\mathcal{P}_{\text{test}}$ often differ from the training distribution $\mathcal{P}_{\text{train}}$ (i.e., $\mathcal{P}_{\text{test}} \neq \mathcal{P}_{\text{train}}$). This means the empirical risk $\hat{\mathcal{L}}_{\text{train}}(\theta)$ provides a misleading picture of the true test risk $\mathcal{L}_{\text{test}}(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{P}_{\text{test}}}[\ell(\mathcal{F}_\theta(x), y)]$, which in turn impairs the model’s generalization to novel or complex scenarios (Liu et al., 2021).
- **Computation Cost:** The reliance on extremely large training datasets $\mathcal{D}_{\text{train}}$ ($N \gg 10^4$) leads to substantial annotation and computational costs, both scaling with N , rendering agent development prohibitively expensive (Mirzasoiman et al., 2020; Mindermann et al., 2022).
- **Redundancy and Inefficient Use of Information:** Treating all N training samples (x_i, y_i) in

$\mathcal{D}_{\text{train}}$ as equally informative is inefficient, as the number of truly effective samples N_{eff} is often much smaller than N (i.e., $N_{\text{eff}} \ll N$) (Zhou et al., 2023). Processing redundant or already mastered examples wastes computation and can hinder generalization, especially on adversarial inputs that remain challenging for current agents (Settles, 2009; Sorscher et al., 2022).

- **Catastrophic Forgetting and Model Churn:** Standard fine-tuning for LMs often suffer from catastrophic forgetting (Luo et al., 2025), where fine-tuning a model on a new task inadvertently degrades its performance on previously acquired skills. Moreover, the rapid release of new and more capable LLMs (Grattafiori et al., 2024; Yang et al., 2025) necessitates a continuous and costly re-training cycle, as the entire fine-tuning process must be repeated on $\mathcal{D}_{\text{train}}$ to leverage the increased knowledge and reasoning abilities of each new base model on the downstream task.

These limitations motivate a new post-training paradigm grounded in *transductive* and *local learning*. Accordingly, we investigate TTT for LLM agents, where inference-time updates can help mitigate the aforementioned issues in agentic settings.

2.2 Self-Improvement and How it Works

Recent studies suggest that LLMs can *self-improve* their capabilities (Huang et al., 2023; Wang et al., 2023; Chen et al., 2024a; Pang et al., 2024; Yuan et al., 2024; Shafayat et al., 2025; Huang et al., 2025), i.e., they can refine their own output distribution using internal signals derived from their own parameters, without relying on external supervision (Xie et al., 2020; He et al., 2020; Huang et al., 2023, 2025). At first glance, this appears paradoxical: how can a model improve its performance if no new information is introduced? The key insight lies in the hypothesis that LLMs contain *hidden knowledge* (Hinton et al., 2015), latent representations within their weights that are not fully accessible through standard inference. Huang et al. (2025) suggests self-improvement emerges through a *sharpening mechanism*, where the model iteratively refines its output distribution to favor high-confidence predictions that align with internal self-evaluation criteria, effectively surfacing this hidden knowledge. This process can be framed as **distribution sharpening**.

Formally, let θ_0 denote the parameters of a base model \mathcal{M} . For a test input $x_i \in \mathcal{D}_{\text{test}}$, the model

induces a conditional distribution $\mathcal{M}_{\theta_0}(y|x_i)$ over possible responses y . Self-improvement methods aim to adapt θ_0 such that the updated parameters θ_i favor responses that maximize an internally defined self-reward r_{self} :

$$\theta_i \approx \arg \max_{\theta} r_{\text{self}}(y|x_i, \theta), y \sim \mathcal{M}_{\theta}(\cdot|x_i) \quad (1)$$

Here, r_{self} is not explicitly optimized but acts as an *implicit intrinsic reward*, induced by the model’s own objective and activated during adaptation (Agarwal et al., 2025; Shafayat et al., 2025; Zuo et al., 2025). Overall, this process tilts the distribution toward more certain, high-reward outputs, amplifying the model’s inherent strengths. Self-improvement, therefore, is not about creating knowledge ex nihilo, but rather about designing algorithms to elicit and amplify this hidden latent knowledge. Building on this foundation, our test-time self-improvement (TT-SI) algorithm operationalizes the sharpening mechanism by learning sample-specific temporary parameters θ_i during inference. Further details on prior work in TTT, general agent post-training, and how our work differs are provided in Appendix C.

3 Method

We introduce a test-time self-improvement framework designed to enable agents to learn from challenging instances on-the-fly by integrating three key components, as shown in Algorithm 1:

- **Self-Awareness: Uncertainty Estimator (H)** identifies inputs x_i at inference-time which the agent is uncertain on, ensuring adaptation focuses only on challenging cases (Section 3.1).
- **Self-Augmentation: Data Synthesis Function (G)** generates a set of K new samples (\mathcal{D}'_i) that are closely related synthetic examples, generated based on the uncertain input x_i (Section 3.2).
- **Self-Improvement: Test-Time Training (T)** temporarily updates the agent’s parameters (θ) on the targeted synthetic data (\mathcal{D}'_i) (Section 3.3).

In the following subsections, we detail each component one by one, first by providing formal definitions followed by their algorithmic specifics.

3.1 Self-Aware Sample Selection at Test-Time

Definition Given a task with inputs x_i , we define **Uncertainty Estimator (H)** that estimates the model’s confidence score (\mathcal{C}) for each candidate action $a_1, \dots, a_n \in \mathcal{A}$ available to the model \mathcal{M} in

its environment (e.g., available API calls). For each input x_i and candidate action a_n , the confidence is computed as:

$$\mathcal{C}_i = \mathbf{H}(x_i, a_n, \mathcal{M}) \quad (2)$$

This estimation is performed without access to ground-truth labels y_i , ensuring fairness and applicability during inference. A sample x_i is deemed **uncertain** if $\mathcal{C}_i < \tau$ for a user-defined confidence threshold τ . By filtering out high-confidence (i.e., certain) instances, this uncertainty estimation step focuses computational and learning resources for the most informative and challenging questions, thereby enhancing both efficiency and quality.

Selecting Uncertain Samples To systematically identify uncertain samples, we implement a *margin-based confidence estimator* using the likelihood distribution generated by the model \mathcal{M} for a given input x_i . Given a set of available actions a_1, a_2, \dots, a_N , we first compute the negative log-likelihood (NLL) for each action as:

$$\text{NLL}(a_n|x_i) = -\log P_{\mathcal{M}}(a_n|x_i), \forall n \in 1, \dots, N \quad (3)$$

However, raw NLL scores are not directly interpretable due to their unbounded nature, limiting their utility in precisely quantifying uncertainty. To address this issue, we apply a *Relative Softmax Scoring (RSS)* mechanism, which transforms these scores into a normalized and interpretable confidence distribution:

$$p^n = \frac{\exp(\ell_n - \max_j \ell_j)}{\sum_{k=1}^N \exp(\ell_k - \max_j \ell_j)} \quad (4)$$

where $\ell_n = -\text{NLL}(a_n | x_i)$. Here, p^n is the RSS confidence score for action a_n , and ℓ_n denotes the negative log-likelihood score corresponding to a_n . The $\max_j \ell_j$ term represents the maximum NLL score among all candidate actions, serving as a numerical stabilizer. To quantify prediction uncertainty, we compute the difference between the highest and second-highest RSS scores, termed the *softmax-difference*. Formally, uncertainty for input x_i is defined as:

$$u(x_i) = p^{(1)} - p^{(2)}, \quad (5)$$

where $p^{(1)}$ and $p^{(2)}$ denote the highest and second-highest RSS scores, respectively. Finally, using a user-defined threshold τ , we select samples exhibiting high uncertainty ($u(x_i) < \tau$), which ensures that subsequent adaptation or analysis efforts are focused on the most ambiguous instances, where the model is likely to benefit most from further information or refinement.

3.2 Data Synthesis Method

Definition When an input sample x_i (without ground-truth labels) is processed during inference and flagged as uncertain by **H**, we trigger the synthesis of K new training examples together with the corresponding labels. **Data Synthesis Function (G)** is invoked for this specific uncertain input x_i , producing a set of K new input-output pairs following the provided instructions (Figure 7). These synthetic examples, $(x'_{ij}, y'_{ij})_{j=1}^K$, are aimed to be semantically similar to the original uncertain sample x_i while introducing slight variations.

The **G** is invoked for this specific uncertain input x_i , producing a set of K novel input-output pairs following the provided prompt of instructions (**P**):

$$\mathbf{G} : x_i \rightarrow \{(x'_{ij}, y'_{ij})\}_{j=1}^K \quad (6)$$

Here, K is a user-defined hyperparameter dictating the volume of synthetic data generated for the current uncertain instance x_i . Each generated pair (x'_{ij}, y'_{ij}) aims to be a plausible instance from the underlying data distribution $P(x, y)$ relevant to x_i , specifically targeting the model’s region of uncertainty around this input. These K generated pairs immediately form a temporary, query-specific dataset \mathcal{D}_i :

$$\mathcal{D}_i = \{(x'_{ij}, y'_{ij})\}_{j=1}^K \quad (7)$$

This dataset \mathcal{D}_i is then used for a localized adaptation of the model parameters θ before processing subsequent input samples with **Test-Time Training (T)** as the next step described in the next section (Section 3.3). This iterative process of detection, synthesis, and adaptation is performed for each sample identified as uncertain.

Data Generation and Validation For each uncertain input x_i , we invoke the synthesis function **G** (Equation (6)) to perform on-the-fly self-augmentation using the model itself as a generator \mathcal{L}_{gen} . Given a hand-crafted prompt (See Figure 7), the unlabeled seed x_i , and a sampling budget K , the model produces K candidate pairs $\{(x'_{ij}, y'_{ij})\}_{j=1}^K$, yielding semantically consistent yet surface-diverse variants similar to Wang et al. (2023). To reduce confirmation bias and error amplification during test-time adaptation, we validate candidates by functional correctness rather than surface semantics: outputs must be parseable, instruction-consistent, and schema-complete with valid types, filtering non-executable or hallucinated tool calls. Among valid candidates, we adopt the majority-agreed function as a pseudo-label under

self-consistency (Zuo et al., 2025; Huang et al., 2023; Yuan et al., 2024), since agreement across independent samples is less likely to arise from stochastic or formatting errors. By constructing \mathcal{D}_i using only executable and majority-agreed tool-calls as labels, this procedure reduces self-labeling bias and curtails the propagation of spurious or non-executable behaviors during adaptation.

3.3 Self-Aware Test-Time Training

Definition Building on our previous steps, uncertainty detection (Section 3.1) and targeted data synthesis (Section 3.2), we now use **Test-Time Training (T)** to adapt model \mathcal{M} during inference, using the generated samples \mathcal{D}_i for each uncertain test query x_i . Once we got \mathcal{D}_i with Equation (6), we optimize initial parameters (θ_0) to minimize the loss function $\mathcal{L}(\mathcal{D}_i; \theta_0)$, producing temporarily updated parameters θ_i for the target task prediction. Importantly, after generating predictions, the model is restored to the original parameters θ_0 for the next iteration using sample x_{i+1} , thereby creating a specialized prediction model for each *out-of-distribution* sample without permanently altering the base model.

Test-Time Training The primary goal of inference-time adaptation is to temporarily adjust the model \mathcal{M} ’s parameters (θ) to better handle the current uncertain sample x_i . This is achieved by fine-tuning the model on the newly synthesized dataset $\mathcal{D}_i = \{(x'_{ij}, y'_{ij})\}_{j=1}^K$. The adaptation involves minimizing a task-specific loss function $\mathcal{L}_{\text{task}}$ over the samples in \mathcal{D}_i . For a given self-generated sample $(x'_{ij}, y'_{ij}) \in \mathcal{D}_i$, the loss is computed as $\ell(\mathcal{M}(x'_{ij}; \theta), y'_{ij})$, and the objective for adapting parameters θ using dataset \mathcal{D}_i is:

$$\theta_i^* = \arg \min_{\theta'} \sum_{(x'_{ij}, y'_{ij}) \in \mathcal{D}_i} \ell(\mathcal{M}(x'_{ij}; \theta'), y'_{ij}) \quad (8)$$

where θ_i^* represents the adapted parameters for the context of x_i .

4 Results

Experimental Protocol We evaluate our approach on four complementary agent benchmarks: NexusRaven (Srinivasan et al., 2023), SealTool (Wu et al., 2024), API-Bank (Li et al., 2023), and ToolAlpaca (Tang et al., 2023). We use Qwen2.5-1.5B-Instruct for main experiments, as its strong performance and small size allow efficient use of limited hardware and demonstrate the potential of compact agentic models (Belcak et al.,

Inference	Method	NexusRaven	SealTool	API-Bank	ToolAlpaca	Avg.	$\Delta\%$
Input/Output	w/o TT-SI (Base)	44.03 \pm 1.42	66.67 \pm 2.39	70.08 \pm 0.82	37.86 \pm 0.97	54.66	–
	w. TT-SI	50.08 \pm 0.47	72.43 \pm 0.75	74.34 \pm 1.89	43.70 \pm 0.68	60.14	\uparrow 3.48
	w. TT-D	52.52 \pm 0.65	73.92 \pm 0.79	75.56 \pm 0.57	42.31 \pm 0.87	61.08	\uparrow 6.42
Majority Vote	w/o TT-SI (Base)	46.56 \pm 1.61	69.73 \pm 1.21	73.96 \pm 0.75	41.94 \pm 0.81	58.05	–
	w. TT-SI	52.20 \pm 1.34	72.93 \pm 0.59	75.68 \pm 0.74	46.79 \pm 1.06	61.90	\uparrow 3.85
	w. TT-D	54.91 \pm 0.57	75.28 \pm 0.86	78.12 \pm 0.71	46.02 \pm 0.53	63.58	\uparrow 5.53
Pass@5	w/o TT-SI (Base)	59.69 \pm 0.51	78.16 \pm 0.92	78.67 \pm 0.69	46.99 \pm 0.53	65.88	–
	w. TT-SI	63.40 \pm 0.26	82.32 \pm 0.67	81.72 \pm 0.58	49.90 \pm 0.87	69.34	\uparrow 3.46
	w. TT-D	65.98 \pm 0.57	84.78 \pm 0.53	84.97 \pm 0.89	52.24 \pm 0.82	71.99	\uparrow 6.11

Table 1: **Main Results of TT-SI.** Accuracy results of baseline prompting (w/o TT-SI), TT-SI, and TT-D across four agentic benchmarks under three inference settings: *Input/Output* (direct prediction) and *Majority Vote* (5-sample self-consistency), and *Pass@5* (correct if any of 5). Δ denotes the average absolute improvement over base model without TT-SI and \uparrow indicates performance increase.

2025), and further include Qwen2.5-7B-Instruct as scaling ablations. All experiments with TT-SI are trained with SFT using LoRA (Hu et al., 2022) on a single NVIDIA A40 GPU, unless otherwise mentioned. Our method often follows a small-sample regime (e.g., single-sample training), higher deviation is expected; thus, all experiments, including baselines, are repeated five times with different sample trainings, seeds, and reported as averages. Additional details on experimental setup are provided in Appendix G.

4.1 Main Results

Insight 1: Agents can self-improve themselves at test-time even when training on just one sample. For our main results, we evaluate TT-SI using three inference-time scaling methods: (i) zero-shot prompting, majority vote over 5 generations, and pass@5 (success if any of 5 generations is correct), as shown in Table 1. Here we check the effect of TT-SI by only using one sample generated by G. TT-SI improves the baseline (w/o TT-SI) across all benchmarks, achieving an average absolute gain of 5.48% for direct inference (54.66% \rightarrow 60.14%), 3.85% for majority voting (58.05% \rightarrow 61.90%), and 3.46% for pass@5 (65.88% \rightarrow 69.34%), which shows TT-SI enables agents to self-improve with only one training instance per uncertain case during inference. Here, TT-SI acts a test-time *sharpening* step where one synthetic sample from an uncertain input re-weights the model’s probability distribution to resolve that uncertainty. We also examine a variant of TT-SI as test-time distillation (TT-D), where where G’s self-generated data is replaced with gpt-5-mini outputs. TT-D further improves over TT-SI by 0.94% for direct inference, 1.68% for majority voting, and 2.65% for pass@5, indicating that higher quality training signals provide

modest but consistent additional gains.

Insight 2: TT-SI outperforms inductive fine-tuning with orders of magnitude less data. We compare TT-SI against inductive SFT on SealTool, which provides an official training split of \sim 13k samples (Figure 3, Left). TT-SI with SFT (72.43%) surpasses all three baselines and exceeds standard inductive SFT (70.20%) by 2.23% accuracy. Notably, TT-SI achieves this improvement using only 190 uncertain cases (each paired with one synthetic example) rather than the full 13k training set. This corresponds to \sim 68 \times fewer samples yet yields higher accuracy, underscoring the effectiveness TT-SI in addressing train–test distribution mismatch compared to inductive fine-tuning.

Insight 3: When training is infeasible, test-time self-improvement with ICL offers a fast alternative. We extend TT-SI to an ICL setting (Figure 3, left), where generated examples are inserted directly into the context of the prompt rather than used for fine-tuning. TT-SI with ICL achieves a slight improvement over the base model (66.37% \rightarrow 68.36%) and even outperforms the standard ICL baseline (67.74%) that leverages SealTool’s training split. This highlights ICL as a training-free, low-overhead alternative to inductive methods. This improvement is likely to be from enhanced model certainty: TT-SI generates demonstrations that boost the model’s confidence in the correct output format and reasoning process, increasing the likelihood of accurate predictions without relying on external training data.

Insight 4: Uncertainty filtering balances accuracy and efficiency. Because TT-SI operates at inference time, efficiency is critical. In our design, H identifies uncertain samples for targeted adapta-

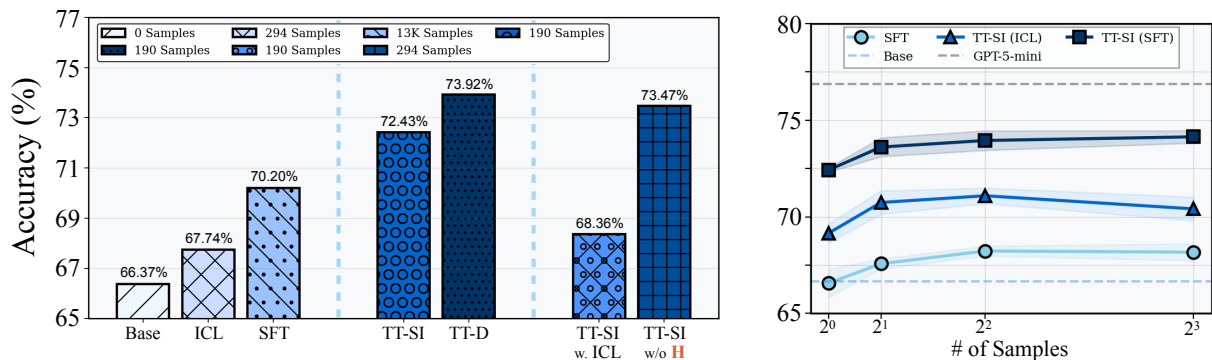


Figure 3: **Experimental Results on SealTool.** **Left:** Accuracy comparison of TT-SI against standard baselines (left-most) and its variants (middle), including ablations (right-most). **Right:** Scaling behavior under different adaptation strategies with varying numbers of samples.

tion, while certain ones are passed directly to the base model. To assess its effect, we also evaluate a variant that treats all test inputs as uncertain (TT-SI w/o **H**) in Figure 3 (left). This achieves only a marginal +1.04% gain, but requires adapting to all 294 test samples rather than 190 samples (an additional 104 LoRA weights to learn) resulting in higher cost. Thus, the slight accuracy gain is outweighed by the efficiency loss, underscoring the importance of uncertainty filtering for practical test-time adaptation. The estimator’s accuracy is further detailed in Appendix D.

4.2 Ablation Studies and Analysis

Insight 5: Uncertainty-guided test-time adaptation is more effective than inductive data scaling under distribution shift. We study how performance scales with increasing training data for both standard SFT and TT-SI under an out-of-distribution (OOD) setting Figure 3, right). Here, we use the xLAM dataset (Zhang et al., 2025) as an OOD training source for SFT, where training subsets (1, 2, 4, 8) are randomly sampled without access to test-time relevance signals. For each scale, we repeat sampling five times and report mean performance with standard deviations. While SFT exhibits limited and quickly saturating gains as more data is added, TT-SI consistently outperforms SFT across all scales, with improvements increasing as additional uncertain test instances are incorporated. This highlights that performance gains stem not from data volume alone, but from uncertainty-guided, instance-specific adaptation at test-time. Notably, the training-free variant of TT-SI using ICL also surpasses standard SFT under identical data budgets, demonstrating that targeted test-time adaptation can outperform inductive fine-tuning even when both methods operate on the same OOD data. These results suggest that under

distribution shift, the ability to identify and adapt to informative test instances is more critical than scaling indiscriminately collected training data.

Insight 6: Optimal τ improves efficiency with minimal accuracy loss. We investigate the impact of the uncertainty threshold τ on TT-SI performance and efficiency in Figure 4 (left). First, TT-SI improves accuracy regardless of τ , surpassing the base of 66.37% in all cases. A high τ (approaching 1) selects all samples, yielding the highest accuracy (73.47%) but requiring updates for all 294 instances, resulting in substantial computational overhead for marginal gains. For example, $\tau = 0.95$ achieves 72.43% accuracy with only 190 updates (35% fewer), preserving near-optimal performance. In contrast, a low $\tau = 0.35$ minimizes false positives (FPR=0.09) but misses many errors (TPR=0.42), lowering accuracy to 68.10%. Thus, $\tau = 0.95$ offers an effective balance, capturing most errors while avoiding redundant updates and optimizing the cost-performance trade-off, akin to human learning focused on uncertain cases (See Appendix D for uncertainty calculation details).

Insight 7: TT-SI scales robustly with larger relative gains for smaller models. To assess whether TT-SI scales across different architectures, we conduct experiments with Qwen2.5-1.5B-Instruct and Qwen2.5-7B-Instruct in Figure 4 (middle). On the smaller model, TT-SI yields a substantial +5.76% absolute gain (66.67→72.43), while on the larger model it delivers a +3.02% gain (80.95→83.97). These improvements indicate that TT-SI can enhance performance irrespective of model size, supporting its architectural generality. Interestingly, the relative boost is more pronounced for smaller models, underscoring potential of small agents as an efficiency-oriented strategy for practical deployments (Belcak et al., 2025).

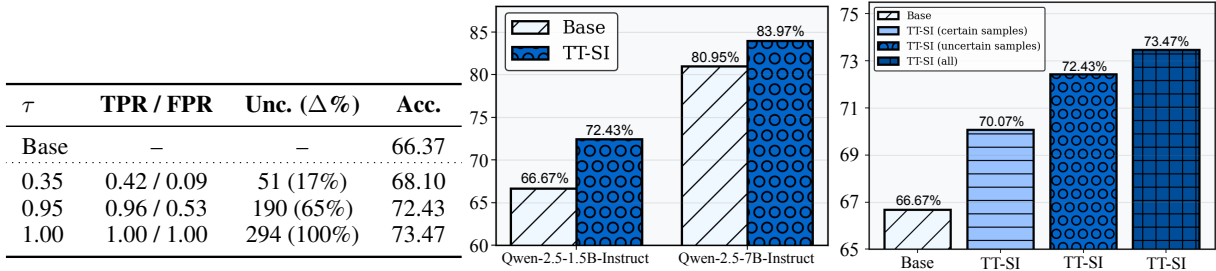


Figure 4: **Left:** Effect of the uncertainty threshold (τ) on uncertain sample selection, efficiency, and accuracy. **Middle:** Model scaling and architectural generalization experiments. **Right:** Impact of self-aware sample selection on TT-SI performance when trained on certain samples (w. H'), uncertain samples (main), and all samples (w/o H).

Insight 8: Targeting uncertain samples is crucial for effective and efficient sharpening. To assess the impact of uncertain samples, we compared three variants of TT-SI on SealTool with Qwen2.5-1.5B-Instruct: (i) TT-SI applied only on uncertain samples detected by H (our main setting), (ii) TT-SI only on certain samples (w. H'), and (iii) TT-SI applied to all test samples regardless of uncertainty (w/o H). Results in Figure 4 (right) show that focusing on uncertain samples (72.43%) yields clear gains over using only certain samples (70.07%). This finding validates our core hypothesis that adaptation is most impactful on challenging instances where the model’s latent knowledge is underutilized, thereby maximizing the “sharpening” effect (Equation (1)). Training on all samples yields only marginal gains while incurring prohibitive cost from 104 additional updates, undermining test-time efficiency.

We provide additional analyses of more detailed H results (Appendix D), qualitative analysis of G (Appendix E), explore oracle effects through controlled “cheating” experiments (Appendix F), and discussions on runtime complexity (Appendix H).

5 Discussions

Conclusion In this work, we investigate *test-time self-improvement* (TT-SI) for language-based agents that (i) measures uncertainty with **Uncertainty Estimator** (H) to decide whether to act directly or adapt, (ii) when uncertain, synthesizes targeted training instances with **Data Synthesis Function** (G), and (iii) performs lightweight updates via **Test-Time Training** (T). We demonstrate that TT-SI improve agents performance during inference, even training with one instance. Across different benchmarks, TT-SI consistently improves test-time performance with strong accuracy and efficiency. We further analyze the variants of TT-SI, the impact of each component, and key takeaways. Our results reveal the potential of TT-SI, suggest-

ing the promise of efficient test-time learning for achieving self-evolving agents.

Impact Statement Overall, our goal is not to propose a specific uncertainty metric or data generator, but rather a novel algorithm that integrates TTT with self-awareness, self-augmentation, and self-improvement for agentic NLP tasks. TT-SI is a modular algorithm: stronger update rules can replace SFT within T , improved uncertainty quantification can plug into H , and better data generation can substitute for G . We believe that, equipped with a perfect **Uncertainty Estimator** (H) that renders the model self-aware of its knowledge and skills, a precise **Data Synthesis Function** (G) capable of generating diverse yet distributionally aligned samples from uncertain subspaces, and an effective update mechanism **Test-Time Training** (T) any scenario becomes learnable in a manner akin to human learning, thereby guiding us toward the realization of a *master algorithm*.

Future Work TT-SI prioritizes self-improvement with TTT, aiming to elicit the model’s optimal performance within its existing knowledge boundary. Beyond self-improvement, a key direction is enabling *self-evolution* where our proposed algorithm can serve as a foundational step towards such self-evolving agents. Another promising direction is more adaptive data generation, where the model itself determines how many synthetic examples are needed for a given uncertain case rather than relying on fixed hyperparameters (Zweiger et al., 2025). Also, our current framework optimizes only the agent, not the data generator; a co-evolutionary setup like *dual-learning*, where both the agent and generator adapt to each other, could further enhance performance. Finally, extending TT-SI to domains such as mathematics (reasoning) or medicine (knowledge) presents an opportunity to explore how domain-specific uncertainty and knowledge interact with self-improvement (Zhao et al., 2025).

683 Limitations

684 While TT-SI demonstrates promising results, it has
685 limitations. First, identifying uncertain samples
686 requires a threshold τ . Although our ablations
687 show that performance gains are consistent across
688 different values of τ (Section 4.2), the best per-
689 formance is sensitive to this choice. Principled
690 methods for learning this threshold autonomously
691 in uncertainty calibration domain remain an open
692 challenge (Bakman et al., 2025). Finally, TT-SI
693 is inherently bounded by the capacity of the base
694 model parameters θ . If the knowledge required to
695 solve a task is absent from the pretrained model
696 (e.g., a newly introduced medical concept), self-
697 improvement alone cannot recover it; in such cases,
698 external knowledge integration through retrieval or
699 search mechanisms can be necessary.

700 References

701 Emre Can Acikgoz, Jeremiah Greer, Akul Datta,
702 Ze Yang, William Zeng, Oussama Elachqar, Em-
703 manouil Koukoumidis, Dilek Hakkani-Tür, and
704 Gokhan Tur. 2025. [Can a single model master both
705 multi-turn conversations and tool use? CoALM: A
706 unified conversational agentic language model](#). In
707 *Proceedings of the 63rd Annual Meeting of the As-
708 sociation for Computational Linguistics (Volume 1:
709 Long Papers)*, pages 12370–12390, Vienna, Austria.
710 Association for Computational Linguistics.

711 Shivam Agarwal, Zimin Zhang, Lifan Yuan, Jiawei Han,
712 and Hao Peng. 2025. The unreasonable effectiveness
713 of entropy minimization in llm reasoning. *arXiv
714 preprint arXiv:2505.15134*.

715 Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu
716 Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob
717 Andreas. 2025. [The surprising effectiveness of test-
718 time training for few-shot learning](#). In *Forty-second
719 International Conference on Machine Learning*.

720 Yavuz Faruk Bakman, Duygu Nur Yaldiz, Sungmin
721 Kang, Tuo Zhang, Baturalp Buyukates, Salman Aves-
722 timehr, and Sai Praneeth Karimireddy. 2025. [Recon-
723 sidering LLM uncertainty estimation methods in the
724 wild](#). In *Proceedings of the 63rd Annual Meeting of
725 the Association for Computational Linguistics (Vol-
726 ume 1: Long Papers)*, pages 29531–29556, Vienna,
727 Austria. Association for Computational Linguistics.

728 Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan
729 Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine
730 Lin, and Pavlo Molchanov. 2025. Small language
731 models are the future of agentic ai. *arXiv preprint
732 arXiv:2506.02153*.

733 Léon Bottou and Vladimir Vapnik. 1992. [Local learning
734 algorithms](#). *Neural Computation*, 4(6):888–900.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and
735 Denny Zhou. 2024a. [Teaching large language mod-
736 els to self-debug](#). In *The Twelfth International Con-
737 ference on Learning Representations*. 738

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei
739 Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and
740 Feng Zhao. 2024b. [Agent-FLAN: Designing data
741 and methods of effective agent tuning for large lan-
742 guage models](#). In *Findings of the Association for
743 Computational Linguistics: ACL 2024*, pages 9354–
744 9366, Bangkok, Thailand. Association for Computa-
745 tional Linguistics. 746

John Flavell. 1979. [Metacognition and cognitive mon-
747 itoring: A new area of cognitive-developmental in-
748 quiry](#). *American Psychologist*, 34:906–911. 749

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu,
750 Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao
751 Qiu, Xuan Qi, Yiran Wu, and 1 others. 2025. A
752 survey of self-evolving agents: On path to artificial
753 super intelligence. *arXiv preprint arXiv:2507.21046*.
754

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri,
755 Abhinav Pandey, Abhishek Kadian, Ahmad Al-
756 Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten,
757 Alex Vaughan, and 1 others. 2024. The llama 3 herd
758 of models. *arXiv preprint arXiv:2407.21783*. 759

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao
760 Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shi-
761 rong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025.
762 Deepseek-r1: Incentivizing reasoning capability in
763 llms via reinforcement learning. *arXiv preprint
764 arXiv:2501.12948*. 765

Moritz Hardt and Yu Sun. 2024. [Test-time training
766 on nearest neighbors for large language models](#). In
767 *The Twelfth International Conference on Learning
768 Representations*. 769

Junxian He, Jiatao Gu, Jiajun Shen, and Marc’Aurelio
770 Ranzato. 2020. [Revisiting self-training for neural
771 sequence generation](#). In *International Conference on
772 Learning Representations*. 773

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015.
774 Distilling the knowledge in a neural network. *arXiv
775 preprint arXiv:1503.02531*. 776

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-
777 Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu
778 Chen. 2022. [LoRA: Low-rank adaptation of large
779 language models](#). In *International Conference on
780 Learning Representations*. 781

Audrey Huang, Adam Block, Dylan J Foster, Dhruv Ro-
782 hatgi, Cyril Zhang, Max Simchowitz, Jordan T. Ash,
783 and Akshay Krishnamurthy. 2025. [Self-improvement
784 in language models: The sharpening mechanism](#). In
785 *The Thirteenth International Conference on Learning
786 Representations*. 787

788	Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2023. Large language models can self-improve . In <i>Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing</i> , pages 1051–1068, Singapore. Association for Computational Linguistics.	844
789		845
790		846
791		847
792		848
793		849
794	Jonas Hübötter, Sascha Bongni, Ido Hakimi, and Andreas Krause. 2025. Efficiently learning at test-time: Active fine-tuning of LLMs . In <i>The Thirteenth International Conference on Learning Representations</i> .	850
795		851
796		852
797		853
798	Thorsten Joachims. 1999. Transductive inference for text classification using support vector machines. In <i>Proceedings of the Sixteenth International Conference on Machine Learning</i> , ICML '99, page 200–209, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.	854
799		855
800		856
801		857
802		858
803		859
804	Komal Kumar, Tajamul Ashraf, Omkar Thawakar, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, Phillip HS Torr, Fahad Shahbaz Khan, and Salman Khan. 2025. Llm post-training: A deep dive into reasoning large language models. <i>arXiv preprint arXiv:2502.21321</i> .	860
805		861
806		862
807		863
808		864
809		865
810	Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In <i>Proceedings of the 29th symposium on operating systems principles</i> , pages 611–626.	866
811		867
812		868
813		869
814		870
815		871
816		872
817	Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. <i>nature</i> , 521(7553):436–444.	873
818		874
819	Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. <i>arXiv preprint arXiv:2304.08244</i> .	875
820		876
821		877
822		878
823		879
824	Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, Jun Wang, and Weinan Zhang. 2025. Robust function-calling for on-device language model via function masking . In <i>The Thirteenth International Conference on Learning Representations</i> .	880
825		881
826		882
827		883
828		884
829		885
830		886
831	Jiashuo Liu, Zheyang Shen, Yue He, Xingxuan Zhang, Renzhe Xu, Han Yu, and Peng Cui. 2021. Towards out-of-distribution generalization: A survey. <i>arXiv preprint arXiv:2108.13624</i> .	887
832		888
833		889
834		890
835	Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2025. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. <i>IEEE Transactions on Audio, Speech and Language Processing</i> .	891
836		892
837		893
838		894
839		895
840	Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. 2018. Umap: Uniform manifold approximation and projection. <i>Journal of Open Source Software</i> , 3(29).	896
841		897
842		898
843		899
	Sören Mindermann, Jan M Brauner, Muhammed T Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltingen, Aidan N Gomez, Adrien Morisot, Sebastian Farquhar, and 1 others. 2022. Prioritized training on points that are learnable, worth learning, and not yet learnt. In <i>International Conference on Machine Learning</i> , pages 15630–15649. PMLR.	844
		845
		846
		847
		848
		849
		850
	Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. 2020. Coresets for data-efficient training of machine learning models. In <i>International Conference on Machine Learning</i> , pages 6950–6960. PMLR.	851
		852
		853
		854
	T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, B. Yang, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, and 7 others. 2018. Never-ending learning . <i>Commun. ACM</i> , 61(5):103–115.	855
		856
		857
		858
		859
		860
	Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei-ge Chen, Olga Vrousos, Corby Rosset, and 1 others. 2024. Agentinstruct: Toward generative teaching with agentic flows. <i>arXiv preprint arXiv:2407.03502</i> .	861
		862
		863
		864
		865
		866
	Thomas O. Nelson. 1990. Metamemory: A theoretical framework and new findings . volume 26 of <i>Psychology of Learning and Motivation</i> , pages 125–173. Academic Press.	867
		868
		869
		870
	Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. <i>Advances in neural information processing systems</i> , 35:27730–27744.	871
		872
		873
		874
		875
		876
	Jing-Cheng Pang, Pengyuan Wang, Kaiyuan Li, Xiong-Hui Chen, Jiacheng Xu, Zongzhang Zhang, and Yang Yu. 2024. Language model self-improvement by reinforcement learning contemplation . In <i>The Twelfth International Conference on Learning Representations</i> .	877
		878
		879
		880
		881
		882
	Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. <i>Advances in Neural Information Processing Systems</i> , 37:126544–126565.	883
		884
		885
		886
	Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks . In <i>Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)</i> , pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.	887
		888
		889
		890
		891
		892
		893
		894
	Marcus Ruopp, Neil Perkins, Brian Whitcomb, and Enrique Schisterman. 2008. Youden index and optimal cut-point estimated from observations affected by a lower limit of detection . <i>Biometrical journal. Biometrische Zeitschrift</i> , 50:419–30.	895
		896
		897
		898
		899

900	Burr Settles. 2009. Active learning literature survey .	State-of-the-art natural language processing. <i>arXiv preprint arXiv:1910.03771</i> .	954 955
901	Sheikh Shafayat, Fahim Tajwar, Ruslan Salakhutdinov, Jeff Schneider, and Andrea Zanette. 2025. Can large reasoning models self-train? <i>arXiv preprint arXiv:2505.21444</i> .	Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. 2024. Seal-tools: Self-instruct tool learning dataset for agent tuning and detailed benchmark. In <i>CCF International Conference on Natural Language Processing and Chinese Computing</i> , pages 372–384. Springer.	956 957 958 959 960 961
905	Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari Morcos. 2022. Beyond neural scaling laws: beating power law scaling via data pruning. <i>Advances in Neural Information Processing Systems</i> , 35:19523–19536.	Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. 2020. Self-training with noisy student improves imagenet classification. In <i>Proceedings of the IEEE/CVF conference on computer vision and pattern recognition</i> , pages 10687–10698.	962 963 964 965 966
910	Venkat Krishna Srinivasan, Zhen Dong, Banghua Zhu, Brian Yu, Hanzi Mao, Damon Mosk-Aoyama, Kurt Keutzer, Jiantao Jiao, and Jian Zhang. 2023. Nexus-raven: a commercially-permissive language model for function calling . In <i>NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following</i> .	An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. <i>arXiv preprint arXiv:2505.09388</i> .	967 968 969 970 971
916	Yu Sun. 2023. Test-Time Training . Ph.D. thesis, EECS Department, University of California, Berkeley.	Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models . In <i>The Eleventh International Conference on Learning Representations</i> .	972 973 974 975 976
918	Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. 2020. Test-time training with self-supervision for generalization under distribution shifts . In <i>Proceedings of the 37th International Conference on Machine Learning</i> , volume 119 of <i>Proceedings of Machine Learning Research</i> , pages 9229–9248. PMLR.	Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E Weston. 2024. Self-rewarding language models . In <i>Proceedings of the 41st International Conference on Machine Learning</i> , volume 235 of <i>Proceedings of Machine Learning Research</i> , pages 57905–57923. PMLR.	977 978 979 980 981 982 983
925	Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. <i>arXiv preprint arXiv:2306.05301</i> .	Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. <i>Advances in Neural Information Processing Systems</i> , 35:15476–15488.	984 985 986 987
930	Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca .	Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. AgentTuning: Enabling generalized agent abilities for LLMs . In <i>Findings of the Association for Computational Linguistics: ACL 2024</i> , pages 3053–3077, Bangkok, Thailand. Association for Computational Linguistics.	988 989 990 991 992 993
935	Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne . <i>Journal of Machine Learning Research</i> , 9(86):2579–2605.	Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Quoc Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Manoj Awalgaoonkar, Rithesh R N, Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, and 2 others. 2025. xLAM: A family of large action models to empower AI agent systems . In <i>Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)</i> , pages 11583–11597, Albuquerque, New Mexico. Association for Computational Linguistics.	994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006
938	Vladimir Vapnik. 1999. <i>The Nature of Statistical Learning Theory</i> . Springer: New York.	Shilong Zhang, Peize Sun, Shoufa Chen, Min Xiao, Wenqi Shao, Wenwei Zhang, Yu Liu, Kai Chen, and Ping Luo. 2024. Gpt4roi: Instruction tuning large language model on region-of-interest. In <i>European conference on computer vision</i> , pages 52–70. Springer.	1007 1008 1009 1010 1011
940	Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions . In <i>Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)</i> , pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.		
948	Philip Winne and Allyson Hadwin. 1998. <i>Studying as Self-Regulated Learning</i> , volume 93, pages 277–304.		
950	Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and 1 others. 2019. Huggingface’s transformers:		

- 1012 Wanjia Zhao, Mert Yuksekgonul, Shirley Wu, and James
1013 Zou. 2025. [Sirius: Self-improving multi-agent systems via bootstrapped reasoning](#). In *Workshop on Reasoning and Planning for Large Language Models*.
1014
1015
1016
- 1017 Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan
1018 Ye, and Zheyang Luo. 2024. [LlamaFactory: Unified efficient fine-tuning of 100+ language models](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 400–410, Bangkok, Thailand. Association for Computational Linguistics.
1019
1020
1021
1022
1023
- 1024 Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer,
1025 Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping
1026 Yu, Lili Yu, and 1 others. 2023. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36:55006–55021.
1027
1028
- 1029 Barry J. Zimmerman. 2002. [Becoming a self-regulated learner: An overview](#). *Theory Into Practice*, 41(2):64–70.
1030
1031
- 1032 Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu
1033 Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, and 1 others. 2025. [Ttrl: Test-time reinforcement learning](#). *arXiv preprint arXiv:2504.16084*.
1034
1035
1036
- 1037 Adam Zweiger, Jyothish Pari, Han Guo, Ekin Akyürek,
1038 Yoon Kim, and Pulkit Agrawal. 2025. [Self-adapting language models](#). *arXiv preprint arXiv:2506.10943*.
1039

1040	Appendix	
1041	A Terminology: Self-Improving and Self-Evolving Agents	13
1042		
1043	B Other Examples from Self-Regulated Learning	13
1044		
1045	C Background on Test-Time Training and LLM Agents	14
1046	C.1 Test-Time Training with LLMs . . .	14
1047	C.2 Previous Work on Large Language Models (LLMs) and Agent Fine-tuning	14
1048		
1049		
1050		
1051	D Uncertainty Estimation Results	15
1052	E Data Generation Details	16
1053	E.1 Experimental Results	17
1054	F Cheating Experiments and TT-SI Comparison	17
1055		
1056	G Implementation Details of TT-SI	18
1057	G.1 Uncertainty Estimation	18
1058	G.2 Data Generation	18
1059	G.3 Training	19
1060	G.4 Evaluation	19
1061	H Additional Run-time Overhead and Resource Usage Analysis	20
1062		
1063	I Use of LLMs	20
1064	A Terminology: Self-Improving and Self-Evolving Agents	
1065		
1066	In the context of agentic systems powered by LLMs, distinguishing between <i>self-improving</i> and <i>self-evolving</i> agents is crucial for understanding their capabilities, limitations, and directing more clear path for future work. We try to provide some terminology with definitions, key differences, and technical insights drawn from recent literature.	
1067		
1068		
1069		
1070		
1071		
1072		
1073	• Self-Improving Agents: Self-improving agents refer to systems that autonomously enhance their own performance on specific tasks through iterative self-refinement mechanisms, without requiring any external intervention. The agent iteratively generates candidate actions, evaluates them against an internal scoring signal (e.g., can be a self-reward function), and updates its subsequent steps to align with	
1074		
1075		
1076		
1077		
1078		
1079		
1080		
1081		

these evaluations. Thus, self-improvement in language models is achieved by reshaping their output probability distribution to preferentially weight higher-quality responses, without incorporating external knowledge beyond what is already encoded in the model parameters (Huang et al., 2025). The scope is generally task-specific, emphasizing efficiency gains within bounded domains, such as data analysis or coding, without altering the underlying system structure.	1082
	1083
	1084
	1085
	1086
	1087
	1088
	1089
	1090
	1091
	1092
• Self-Evolving Agents: Self-evolving agents are designed for broader, continuous adaptation across dynamic environments and sequential tasks, enabling lifelong learning and generalization. These agents evolve not only parametric components (e.g., model weights) but also non-parametric elements like memory, tools, prompts, and architecture (Gao et al., 2025). This allows agents to handle open-ended scenarios, such as real-world feedback loops in interactive environments.	1093
	1094
	1095
	1096
	1097
	1098
	1099
	1100
	1101
	1102
	1103
Overall, self-improving language models focus on optimizing specific task performance by iteratively refining their output distribution toward higher-quality responses using mechanisms, without adding new information. In contrast, self-evolving language models prioritize holistic adaptation, dynamically restructuring their knowledge or architecture to enhance generalization and handle novel environments over time.	1104
	1105
	1106
	1107
	1108
	1109
	1110
	1111
	1112
B Other Examples from Self-Regulated Learning	1113
	1114
Student Homework. Our paradigm of test-time self-improvement (TT-SI) also draws inspiration from how students engage in self-learning (Zimmerman, 2002). When faced with uncertainty, such as being unsure how to solve a homework problem (<i>self-awareness</i>), students often seek out related examples from textbooks and online resources (<i>self-augmentation</i>) to resolve their knowledge gap and build confidence in solving similar tasks (Winne and Hadwin, 1998) (<i>self-improvement</i>). This process is closely aligned with theories of metacognition and self-regulated learning, where learners actively identify their knowledge gaps and pursue targeted resources to close them (Nelson, 1990; Winne and Hadwin, 1998; Zimmerman, 2002). Unlike classical active learning in machine learning (Settles, 2009), which deliberately queries an	1115
	1116
	1117
	1118
	1119
	1120
	1121
	1122
	1123
	1124
	1125
	1126
	1127
	1128
	1129
	1130
	1131

1132	oracle or annotator for novel and informative ex-	and Sun (2024) fine-tune on retrieved nearest	1182
1133	amples, our method generates additional data au-	neighbors to reduce perplexity, and SIFT (Hübotter	1183
1134	tomatically without external supervision. More-	et al., 2025) actively selects diverse, informative	1184
1135	over, while student learning often benefits from	neighbors to limit redundancy. Closest to our	1185
1136	diverse perspectives and human explanations, our	setting, Akyürek et al. (2025) apply rule-based	1186
1137	approach focuses on generating semantically simi-	linear transformations to in-context test examples	1187
1138	lar but slightly varied problem instances to refine	in ARC to get additional test-time training data.	1188
1139	the model’s performance. This analogy highlights	However, these approaches either target perplexity	1189
1140	the natural intuition behind TT-SI while under-	rather than general reasoning tasks, assume access	1190
1141	scoring its distinct contribution as an automated,	to high-quality neighbors, or in-context exemplars.	1191
1142	uncertainty-driven, and cost-efficient alternative to	Our work instead selects informative test instances,	1192
1143	data collection.	generates and filters training signals on-the-fly,	1193
1144	Sport Analogy. Lets also consider a running	yielding improvements on challenging agent	1194
1145	back (RB) in American football honing skills for	benchmarks. To the best of our knowledge, this	1195
1146	the NFL Combine, whose performance relies heav-	is the first language generation-based test-time	1196
1147	ily on lower-body strength and the rate of force	fine-tuning method applied to LLM-based agents.	1197
1148	development for explosive acceleration. If an ath-		
1149	lete’s primary weakness lies in short-burst speed		
1150	and rapid change-of-direction, a targeted regimen		
1151	emphasizing plyometric drills, resisted sprints, and	C.2 Previous Work on Large Language	1198
1152	eccentric–concentric coupling work can directly	Models (LLMs) and Agent Fine-tuning	1199
1153	address this limitation. In contrast, allocating sig-		
1154	nificant training time to non-specific full-body hy-	The de-facto approach to equip LLMs with new	1200
1155	pertrophy (e.g., frequent bench pressing or iso-	capabilities is to collect task-specific corpora and	1201
1156	lated arm work) not only increases recovery de-	fine-tune on them (Kumar et al., 2025), with	1202
1157	mands and neuromuscular fatigue but can also	such datasets either curated through human an-	1203
1158	lead to excess non-functional muscle mass, which	notation (Ouyang et al., 2022) or synthesized by	1204
1159	may reduce stride frequency and overall sprint ve-	LLMs (Wang et al., 2023). Following these ad-	1205
1160	locity. By diagnosing the limiting factor (<i>self-</i>	advancements, LLM-based agents have emerged (Yao	1206
1161	<i>awareness</i>), incorporating performance-specific	et al., 2023), where models interact with external	1207
1162	drills (<i>self-augmentation</i>), and progressively re-	tools and APIs rather than producing text alone,	1208
1163	fining execution through repeated exposure (<i>self-</i>	which require learning tool-use skills and handling	1209
1164	<i>improvement</i>), the athlete can achieve more mean-	structured inputs and outputs (Patil et al., 2024).	1210
1165	ingful outputs without the performance trade-offs	Training LLMs for such agentic skills has led to	1211
1166	of untargeted training.	the exploration of effective dataset design and tun-	1212
		ing strategies aimed at improving generalizabil-	1213
1167	C Background on Test-Time Training	ity (Zeng et al., 2024; Mitra et al., 2024; Chen	1214
1168	and LLM Agents	et al., 2024b). However, this inductive approach is	1215
		prone to catastrophic forgetting when transferred	1216
1169	C.1 Test-Time Training with LLMs	across different environments, requires costly data	1217
		generation pipelines, and does not guarantee con-	1218
1170	Test-time training (TTT) performs small,	sistent gains over strong base models. In contrast,	1219
1171	ephemeral parameter updates during inference,	to the best of our knowledge, our work introduces	1220
1172	conditioning the model on the current input and	the first application of TTT to LLM-based agents	1221
1173	thus partially collapsing the train–test boundary	by enabling temporary parameter updates during in-	1222
1174	(Sun, 2023). The idea traces to local and trans-	ference and therefore avoids catastrophic forgetting	1223
1175	ductive learning, where hypotheses are adapted	and reduces dependence on large offline datasets.	1224
1176	after observing test inputs (Bottou and Vapnik,	Furthermore, considering training efficiency, we	1225
1177	1992; Joachims, 1999). In deep learning, Sun	incorporate selective data usage by using only the	1226
1178	et al. (2020) showed that a simple self-supervised	most informative samples for the model, rather than	1227
1179	TTT objective can improve the robustness of	redundantly training on already well-understood in-	1228
1180	image classifiers under distribution shift. In	stances.	1229
1181	LLMs, TTT is comparatively nascent: Hardt		

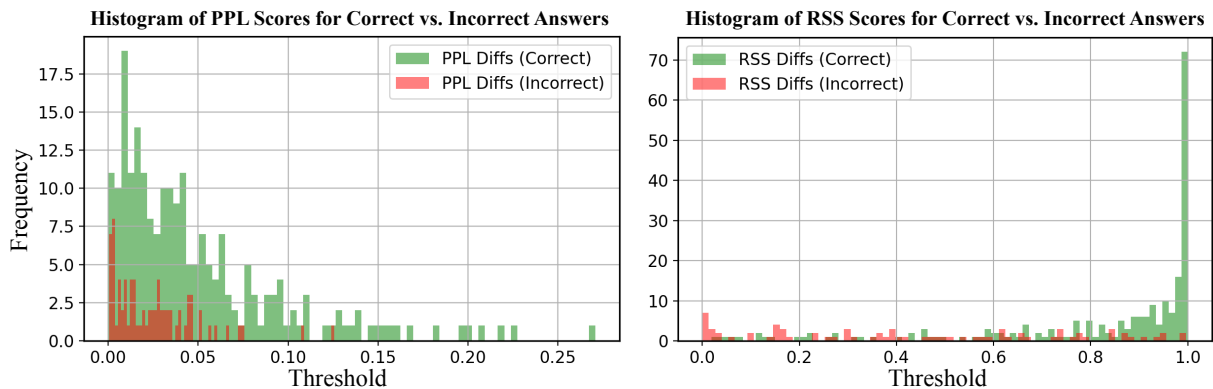


Figure 5: **Comparison of PPL and RSS (ours) as Uncertainty Estimator (H) on SealTool.** Histograms show score differences between the top-1 and top-2 predictions. The **green** bars denote correct predictions, and **red** bars denote incorrect ones. PPL-based uncertainty (left) shows strong overlap between correct and incorrect cases, whereas our RSS-based estimator (right) yields a clearer separation, enabling more reliable uncertainty filtering.

D Uncertainty Estimation Results

To evaluate the effectiveness of our proposed **Uncertainty Estimator (H)** in Section 3.1, we compare our RSS-based method defined in Equation (4) against a standard perplexity (PPL)-based uncertainty signal using Qwen-2.5-1.5B-Instruct on SealTool (Wu et al., 2024). We visualize the distributions of Equation (5) as histograms in Figure 5. The x-axis shows the difference between the most probable and the second most probable function call, while the y-axis denotes frequency. Green bars correspond to correctly predicted test samples, and red bars to incorrect ones.

Ideally, correct predictions (green) should cluster separately from incorrect ones (red), enabling a clear threshold τ for filtering uncertain cases. As shown in the left histogram, PPL differences for correct and incorrect answers are heavily intertwined, making it difficult to separate them with any heuristic. In contrast, our RSS-based estimator (right) exhibits a clear separation: correct predictions concentrate on the right side with larger score differences (indicating higher confidence in the top prediction), while incorrect predictions are scattered on the left side with smaller differences (reflecting model uncertainty). This separation highlights both the interpretability and effectiveness of our proposed **Uncertainty Estimator (H)**, especially compared to the baseline PPL-based uncertainty measure.

We further check the performance of **H** with Qwen-2.5-1.5B-Instruct and Qwen-2.5-7B-Instruct on SealTool (Wu et al., 2024) in Figure 6 Left and Right, respectively. For every test input, we (i) obtain RSS confidence scores p_n via Equation (4), (ii) compute

the softmax-difference $u(x_i) = p^{(1)} - p^{(2)}$, and (iii) mark the prediction as *uncertain* (i.e., select it for adaptation) when $u(x_i) < \tau$. We study how the choice of threshold τ affects performance in Figure 6 by reporting true positive rate (TPR, i.e., correctly flagging the model’s wrong predictions as uncertain) and false positive rate (FPR, i.e., incorrectly flagging the model’s correct predictions as uncertain). As the threshold τ for the softmax-difference $u(x_i)$ increases (0.35 \rightarrow 0.99), the condition $u(x_i) < \tau$ for being uncertain becomes less stringent, leading to more samples being identified as uncertain and routed for downstream adaptation for both model. Raising τ monotonically increases both the TPR and FPR. Ideally, the goal is to **maximize TPR while keeping FPR as low as possible**. For Qwen-2.5-1.5B-Instruct, when we increase τ , TPR rises steadily from 42% at $\tau = 0.35$ to 96% at $\tau = 0.95$ and FPR remains low across all thresholds, only increasing from 9% to 53% as τ . The overall discrimination ability of the estimator, quantified by Youden’s J statistic (Ruopp et al., 2008) (TPR – FPR), reaches its highest value of 46.0% when the threshold τ is set to 0.95. On the other hand, a similar trend is observed with Qwen-2.5-7B-Instruct: as τ increases, TPR rises from 39% at $\tau = 0.35$ to 99% at $\tau = 0.95$, while FPR increases from 15% to 49%, yielding one of the group’s highest Youden’s index values at 50.

At one of its optimal threshold, the estimator captures nearly 99% of model errors as uncertain, while only miss classifying 49% of correct answers. This reflects a strong balance between *sensitivity* and *specificity*, demonstrating the effectiveness of

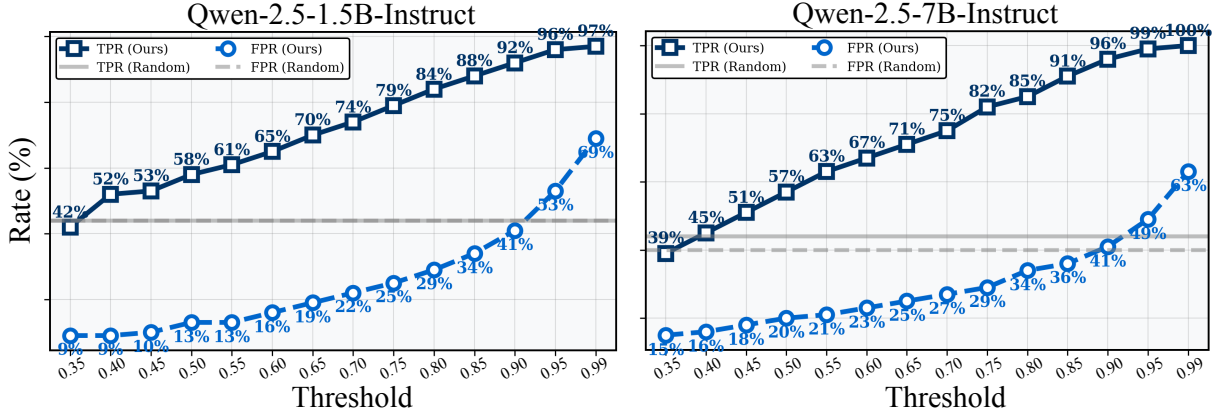


Figure 6: **Threshold (τ) Experiments with Uncertainty Estimator (H) on SealTool.** We investigate the effect of the softmax-difference threshold τ , which controls the sensitivity of H in flagging uncertain cases. The left plot shows the true positive rate (TPR, proportion of correctly identified uncertain cases), while the right plot reports the false positive rate (FPR, proportion of certain cases incorrectly flagged as uncertain). As τ increases, TPR rises, but so does FPR, illustrating the trade-off between coverage and reliability.

our (H). For all experiments, we adopt $\tau = 0.95$ as the default threshold, as it consistently achieves the best trade-off between identifying the majority of erroneous outputs and minimizing unnecessary intervention on correct predictions. In Section 4.2, we also analyze the impact of τ on TT-SI and find that the framework consistently improves base accuracy across all settings. Nevertheless, τ substantially influences efficiency, underscoring an inherent trade-off between accuracy and computational cost. Moreover, τ is a hyperparameter and by tuning τ , one can flexibly adjust the stringency of uncertainty filtering to match the requirements of specific downstream tasks or adaptation budgets, ensuring both effective error coverage and efficient resource allocation. Future work should establish more effective methods to automatically determine the optimal τ , as this remains a central challenge in the domain of uncertainty estimation.

Finally, we compare our H on Qwen-2.5-1.5B-Instruct with several baselines on SealTool. The baselines include: Random, which labels predictions as uncertain uniformly at random; Trivial, which marks all predictions as uncertain; and Perplexity (PPL), which uses negative log-likelihood scores as an uncertainty signal based on Equation (3). To evaluate, we apply each method to the ground-truth test set of SealTool and measure the resulting true positive rate (TPR), false positive rate (FPR), F1 score, and Youden’s J statistic. Results show that our method achieves a TPR of 96.10%, effectively capturing almost all misclassified samples, while maintaining a moderate FPR of 53.46%. More importantly, our approach achieves the highest J score (42.64%),

Method	TPR (\uparrow)	FPR (\downarrow)	F1 (\uparrow)	J (\uparrow)
Random	44.16	43.78	33.01	0.38
Trivial	100.00	100.00	41.51	0.00
Perplexity	57.14	38.25	43.14	18.89
Ours	96.10	53.46	55.43	42.64

Table 2: **Comparison of Uncertainty Estimators on SealTool.** Our method achieves the highest balance (J) compared to Random, Trivial, and Perplexity baselines.

more than double that of Perplexity (18.89%), and also yields the best F1 score. These results quantitatively support our earlier visualization claims in Figure 5, highlighting that H provides a strong balance between sensitivity and specificity compared to naive baselines.

E Data Generation Details

The implementation of **Data Synthesis Function (G)**, which is triggered for each uncertain sample x_i , employs the agent itself for data synthesis (\mathcal{L}_{gen}) as *self-augmentation*. For each generation instance, \mathcal{L}_{gen} is provided with a carefully hand-crafted prompt \mathcal{P} (See Figure 7), the uncertain input x_i serving as the direct seed (critically, without its corresponding label y_i), and a specified number of samples K to generate. The model then produces K new input-output pairs, denoted as $\{(x'_{ij}, y'_{ij})\}_{j=1}^K$. This seed-based generation process, inspired by self-instruction methodologies (Wang et al., 2023), guides \mathcal{L}_{gen} to produce variants that maintain the core semantic meaning and task relevance of x_i while introducing controlled surface-level variations. By *synthesizing data in this on-the-fly* manner for each uncertain

Data Generation Prompt

You are given an instruction, input, and example sample as seed, but not labeled output. You must generate new synthetic examples that closely match the original uncertain scenario.

1. Create distinct variants of the seed by altering names, context, or wording but no variant may duplicate the original.
2. Your response format must be: { "instruction": "<instruction>", "input": "<input>", "output": "<output>" }
3. The "output" field must be a function calling JSON object with the following structure: [{"name": "Tool name", "parameters": {"Parameter name": "Value",...}},...]

Number of Examples

Generate <number> examples.

Seed Example

<seed_example>

Generated Examples

Your Response:

Figure 7: Data generation prompt for uncertain samples with LLM.

instance, we facilitate targeted and timely model adaptation, aiming to improve performance on precisely the types of queries the model struggles with, as they are encountered.

E.1 Experimental Results

For each uncertain input x_i detected by the procedure in **H** we synthesize exactly one new example ($K = 1$) using the same LLM (i.e., Qwen-2.5-1.5B-Instruct). The generator receives only the instruction and query of x_i —never the gold label—and produces both a revised input and its answer, thereby creating a temporary, query-specific dataset \mathcal{D}_i that is used immediately for inference-time adaptation. Interpreting and understanding how our **Data Synthesis Function (G)** operates is essential for understanding the effectiveness of our generations. To this end, we embed (Reimers and Gurevych, 2019) all SealTool test samples, an uncertain example x_i from this set, and ten self-generated queries for x_i produced by Qwen-2.5-1.5B-Instruct into a two-dimensional semantic space using UMAP (McInnes et al., 2018). As visualized in Figure 8, the generated samples form a compact cluster in the embedding space, closely aligned with both in each other and the corresponding uncertain input. This spatial proximity suggests that our data synthesis function **G** can yields mutually consistent and semantically faithful examples, effectively bridging the gap for adaptation to challenging queries.

Detailed Qualitative Analysis of Synthetic Query Generation. To provide a more comprehensive qualitative evaluation of the quality and diversity of our self-generated synthetic queries, fo-

cus on the semantic embedding space derived from the SealTool dataset (Wu et al., 2024). We begin by encoding textual data into dense vectors. Each sample is represented as a concatenation of the system prompt, user query, and output response (or equivalent instruction-input-output triples for generated data). These are embedded using the Sentence-BERT model with all-mpnet-base-v2 (Reimers and Gurevych, 2019), which produces 768-dimensional vectors optimized for semantic similarity in natural language tasks. The high-dimensional embeddings are then projected into a two-dimensional latent space using Uniform Manifold Approximation and Projection (UMAP) (McInnes et al., 2018), a nonlinear dimensionality reduction algorithm that preserves both local and global topological structures more effectively than alternatives like t-SNE (van der Maaten and Hinton, 2008). We configure UMAP with 15 neighbors to balance local clustering and global layout, and set minimum distance as 0.1 to allow moderate spread in low-density regions, facilitating the identification of outliers such as uncertain samples.

F Cheating Experiments and TT-SI Comparison

To better contextualize our proposed method, we conducted a *cheating experiment* in which baseline models were explicitly trained on the test set. This unrealistic setting serves as an upper bound on performance for common adaptation strategies, including in-context learning (ICL), supervised fine-tuning (SFT), and test-time training (TTT). Using Qwen-2.5-1.5B-Instruct, we report results on

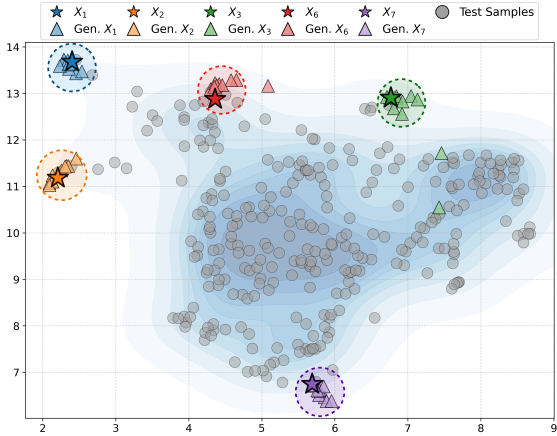


Figure 8: **Self-Generated Data Visualization.** All test samples (circles) are projected into a two-dimensional semantic space via UMAP, and shown with the density contour distributions. The star denotes the uncertain input x_i , and triangles indicate 10 randomly sampled, self-generated synthetic queries from uncertain sample x_i . Generated samples are tightly clustered and situated near x_i , demonstrating distributional alignment of **G**.

the SealTool benchmark in Figure 9, where the left four bars show the cheating baselines. For comparison, we include our proposed TT-SI framework and its TT-D variant (right bars), which were not trained on the test set but instead evaluated under their standard configuration.

When comparing cheating TTT (78.89%) with our TT-SI (72.43%), the scores are remarkably close, suggesting that providing highly similar samples during test-time training (rather than exact ground truth answers) is sufficient to shift the model toward the uncertain sample distribution. This process increases confidence for samples previously overlooked by the base model’s parameters, leading to improved performance after temporary updates. If the gap between cheating TTT and TT-SI were larger, it would indicate that exact ground truth answers are critical and that better data generation methods beyond self-improvement are needed. Interestingly, TT-SI achieves scores comparable to SFT trained on the test set. More surprisingly, neither update-based method (SFT, TTT) reaches 100% accuracy after one epoch of training on actual samples, suggesting issues with the update rules themselves. Also, it is important to mention that SFT reaches 96.60% after 10 epochs of training on this 294 sample test set. In contrast, ICL achieves 100% accuracy directly when these actual test samples are added to the prompt during inference.

G Implementation Details of TT-SI

We believe that transparency and detailed reporting are essential for both understand the approach in-depth and advancing future research. Accordingly, we do our best to provide complete descriptions of each component of the proposed TT-SI framework: **Uncertainty Estimator (H)** (Section 3.1), **Data Synthesis Function (G)** (Section 3.2), and **Test-Time Training (T)** (Section 3.3). In all steps, we use Qwen2.5-1.5B-Instruct¹ from HuggingFace, running on a single NVIDIA A40 GPU.

G.1 Uncertainty Estimation

For implementing **H**, we use the HuggingFace Transformers library (Wolf et al., 2019), as it provides straightforward access to token logits and confidence estimates through the AutoModelForCausalLM class, unlike vLLM. We do not apply temperature scaling at this step. To estimate uncertainty, we directly input the test sample instruction as a query and merge all available function names extracted via regex operations. The confidence scores for each candidate function are computed using Equation (3) and normalized with RSS as formulated in Equation (4). On SealTool, labeling a sample as uncertain requires on average 0.87 seconds.

G.2 Data Generation

Once an uncertain sample is identified with **H**, we generate K similar samples using **G**. This is done with the prompt shown in Figure 7, where the model is asked to create slight variations of the sample (but not the exact same sample) along with corresponding labels. The uncertain sample is inserted into the prompt as a seed (<seed>) (Wang et al., 2023), and K is set as a hyperparameter (<number>) by replacing special tokens. We then extract the generated samples. We study two variants: TT-SI and TT-D. In TT-SI, the same Qwen2.5-1.5B-Instruct model generates its own synthetic samples, while in TT-D, sample generation is performed by GPT-5-mini². For TT-SI, we use vLLM with temperature 0.7 and maximum length set to 32768. Because of the model’s small scale, sometimes parsing errors occur, where the model may omit some JSON strings. We allow

¹<https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct>

²<https://platform.openai.com/docs/models/gpt-5-mini>

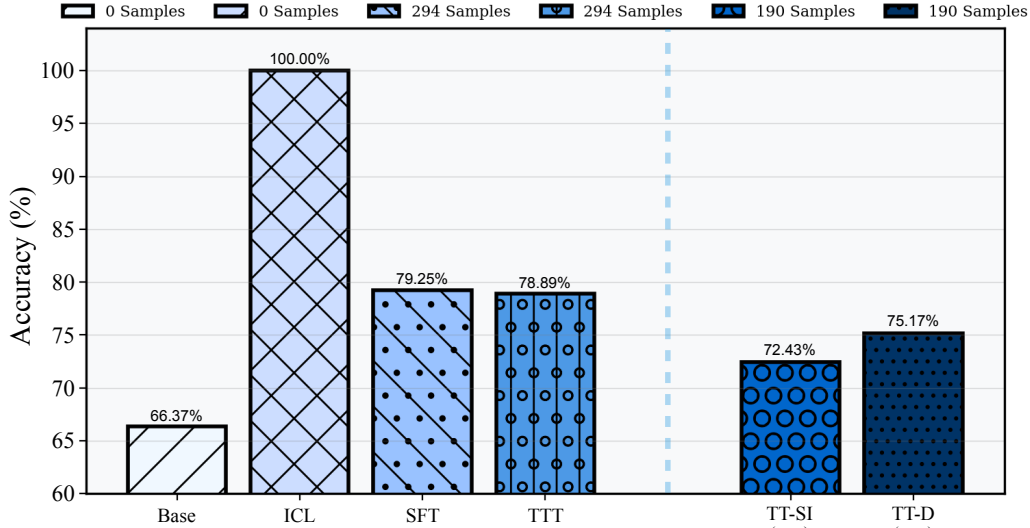


Figure 9: **Cheating Experiment on SealTool.** Comparison of baseline methods—Base, in-context learning (ICL), supervised fine-tuning (SFT), and test-time training (TTT)—when explicitly trained on the test set (left four bars) using Qwen-2.5-1.5B-Instruct. We also report actual (non-cheating) scores for our TT-SI algorithm and its TT-D variant (right two bars).

up to 5 retries; in practice, errors are usually resolved within the second or third attempt. For gpt-5-mini, we use the standard OpenAI API without temperature adjustments or additional decoding strategies. The computational cost of using gpt-5-mini API is negligible, effectively zero for a single experiment. On average, generating one sample with TT-SI takes approximately 3.45 seconds. A qualitative analysis of the data generation process is provided in Appendix E.

G.3 Training

After obtaining the K generated samples from **G**, we directly perform test-time fine-tuning with **T**. For training, we use LLaMA-Factory (Zheng et al., 2024), chosen for its optimized implementation and user-friendly CLI. All fine-tuning is conducted with Parameter-Efficient Fine-Tuning (PEFT) via LoRA (Hu et al., 2022). We set the LoRA parameters to $rank = 8$ and $\alpha = 16$, applied to all linear layers. Training runs for 5 epochs with a fixed learning rate of 1.0×10^{-4} , a warm-up ratio of 0.03, and batch size of 1. Despite the short training, we use a cosine scheduler by default. Otherwise, we keep the default configuration parameters of LLaMA-Factory and HuggingFace without further modifications. For inference on the fine-tuned models, we adopt the same vLLM decoding settings as in data generation with temperature is set to 0.7. All trainings follow the Alpaca-style data format (Taori et al., 2023), where the instruction and input fields are zero-padded, and the loss is computed only on the output field. Training a single sample with

LLaMA-Factory takes on average 2.05 seconds. On the other hand, our reported timings exclude software initializations, I/O operation overheads from checkpoint loading, merging, and saving, as these are highly implementation/tool-dependent, which is discussed in Appendix H.

G.4 Evaluation

We evaluate our method on three established agent benchmarks: NexusRaven (Srinivasan et al., 2023), SealTool (Wu et al., 2024), API-Bank (Li et al., 2023), and ToolAlpaca (Tang et al., 2023). **NexusRaven** focuses on realistic software operation tasks, particularly in domains such as cybersecurity and enterprise applications. It is designed to test high-fidelity function execution in business scenarios, featuring long and diverse tool invocations across 65 distinct APIs with a total of 318 samples (see Figure 10 for an example). **SealTool** is one of the most extensive and recent benchmarks, comprising 4,076 APIs spanning diverse domains. Its latest version is designed to minimize potential data leakage, making it a robust benchmark for tool-use evaluation. In our experiments, we use the curated test set of 294 samples (see Figure 11 for an example). **API-Bank** contains 314 multi-turn conversations with 753 distinct API calls. It evaluates an LLM’s ability to select appropriate functions and arguments in realistic dialogue settings. Following prior work, we focus on 316 samples from Levels 1 and 2, which balance task complexity and data availability (see Figure 12 for an example). **ToolAlpaca** employs a synthetic data generation

framework, featuring 3,938 tool-use instances in 50 categories, designed to assess generalized tool-use capabilities across diverse APIs (see Figure 13 for an example).

Following prior work (Lin et al., 2025), we use 318, 294, 361, and 103 test samples for each benchmark, respectively, consistent with previous studies, except for slight modifications on evaluation metrics to ensure a more accurate and reliable evaluation setup. Across all benchmarks, we evaluate whether models produce correct function names, arguments, and their corresponding values/types. A key challenge involves string arguments, where models often produce superficial variations of gold-standard values—differing in case, tense, or plurality (e.g., "fatigued" vs. "fatigue" or "fatigous"). We argue these discrepancies reflect artifacts of current benchmarks rather than genuine errors, yet they are difficult to evaluate fairly: exact-match metrics are overly strict, while LLM-based judges introduce unreliability. We therefore adopt a soft-matching metric that ignores case and minor morphological variations. This adjustment changes performance by only 2–3%, but provides a more accurate estimate of functional correctness. Thus, our evaluation framework prioritizes semantic equivalence over superficial string matching, better reflecting real-world tool-calling capability.

H Additional Run-time Overhead and Resource Usage Analysis

We analyze the latency of each algorithmic step of TT-SI on the SealTool dataset, excluding model merging and other I/O overheads, which we discuss separately below. For consistency, we employ HuggingFace (Wolf et al., 2019) for confidence estimation with **H**, vLLM (Kwon et al., 2023) for data synthesis with **G** and inference, and LLaMA-Factory (Zheng et al., 2024) for trainings with **T**.

Table 3 reports total latency, per-sample averages, and variation statistics. On average, **H** requires 0.87s per sample to estimate uncertainty. For uncertain inputs, **G** synthesizes additional variants in 3.45s, which are subsequently used for **T** training updates that take 2.05s each. Finally, inference via vLLM adds only 0.89s per sample. These steps together amount to an average of $\sim 7.3s$ per uncertain sample, while non-uncertain samples require only 0.87s; amounting to ~ 36 minutes for 190 updates and 104 direct inferences. In contrast, SFT requires 7,966.6s ($\sim 2h12m$) to train on SealTool’s 13K-sample split. Despite training on $\sim 68\times$

fewer samples, TT-SI delivers a $3.7\times$ wall-clock speed-up.

Step	Avg (s)	Std (s)	Range (s)
Uncertainty (H)	0.87	0.13	0.53–1.20
Data Generation (G)	3.45	1.23	2.04–9.63
Training (T)	2.05	0.42	1.67–3.49
Inference	0.89	0.14	0.49–1.34
Total per-sample: 7.26s (uncertain)			1.76s (certain)

Table 3: **Latency Analysis.** Step-wise latency of TT-SI on SealTool (I/O and merge overhead excluded).

However, we note that most of the additional latency stems from model merging, file-saving operations, and vLLM model loading. While our main algorithmic steps: **Uncertainty Estimator**, **Data Synthesis Function**, and **Test-Time Training** introduce only minimal computational overhead as discussed above, I/O operations can substantially increase end-to-end latency. Since efficient file handling lies outside the scope of our main contribution, we do not focus on these issues in our work. For these reason, we exclude such I/O overheads from the reported clock-time analysis. On the other hand, third-party libraries offer options to directly use merged weights without writing them to disk, and more efficient configurations can be implemented to manage these steps. Thus, we recommend future industrial applications prioritize more optimized and scalable I/O strategies.

I Use of LLMs

In this work, LLMs were used in this work for three purposes: (i) as base models under study for test-time training, (ii) as baselines for empirical comparison, and (iii) for minor assistance in refining the readability of this manuscript. Both open-source (e.g., Qwen (Yang et al., 2025)) and closed-source models (e.g., GPT-5-mini³) were employed for training and data-generation. The prompt used for improving writing quality was similar to *"Please make more clear sentence, making sure to remove any grammatical mistakes."* Importantly, all scientific ideas, methods, experiments, and conclusions originate from the authors. When LLMs were used for language refinement, outputs were carefully reviewed to prevent the introduction of hallucinated or incorrect content, ensuring that all arguments, findings, and perspectives are solely those of the authors.

³<https://platform.openai.com/docs/models/gpt-5-mini>

Algorithm 1 Test-Time Self-Improvement Framework

Require: Test dataset $\mathcal{D}_{\text{test}}$, model \mathcal{M} , data generation prompt \mathcal{P} , temporary dataset size K , initial model parameters θ_0

```
1: for each  $x_i \in \mathcal{D}_{\text{test}}$  do
2:   Step 1: Uncertainty Estimator (H)
3:   Compute uncertainty (softmax-difference):
4:      $\ell_n = -\log P_{\mathcal{M}}(a_n|x_i), \quad \forall a_n$   $\triangleright$  Negative Log-Likelihood (NLL) for candidate action
5:      $p_n = \frac{\exp(\ell_n - \max_j \ell_j)}{\sum_k \exp(\ell_k - \max_j \ell_j)}$   $\triangleright$  Apply Relative Softmax Scoring (RSS) normalization
6:      $u(x_i) = p^{(1)} - p^{(2)}$   $\triangleright$  Highest minus second-highest RSS scores
7:   Step 2: Data Synthesis Function (G)
8:   if  $u(x_i) < \tau$  then  $\triangleright$  Check uncertainty
9:     Generate  $K$  synthetic samples using LLM:
10:     $\mathcal{D}_i \leftarrow \mathcal{L}_{\text{gen}}(x_i, K)$   $\triangleright$  Equation (6)
11:   Step 3: Test-Time Training (T)
12:   Learn temporary model parameters  $\theta_i^*$  via LoRA:
13:     $\theta_i^* \leftarrow \arg \min_{\theta_0} \sum_{(x', y') \in \mathcal{D}_i} \ell(\mathcal{M}(x'; \theta_0), y')$   $\triangleright$  Equation (8)
14:   Perform inference with adapted parameters  $\theta_i^*$ :
15:     $\hat{y}_i \leftarrow \mathcal{M}(x_i; \theta_i^*)$ 
16:   Reset model parameters:
17:     $\theta_i^* \rightarrow \theta_0$   $\triangleright$  Restore original parameters
18:   else
19:     Perform inference directly:
20:     $\hat{y}_i \leftarrow \mathcal{M}(x_i; \theta_0)$ 
21:   end if
```

NexusRaven Test Sample Example (ID: 317)

You are an advanced assistant capable of using tools to help the user. You may call one or more functions to assist with the user query. For any user request that requires a function, respond by returning a function call inside `<tool_call>...</tool_call>` XML tags, with a JSON object specifying the "name" of the function and the "arguments".

Task Instruction

In order to complete the user's request, you need to select one or more appropriate tools from the following tools and fill in the correct values for the tool parameters. Your specific tasks are:

1. Make one or more function/tool calls to meet the request based on the question.
2. If none of the functions can be used, point it out as an empty list and refuse to answer.
3. If the given question lacks the parameters required by the function, also point it out.

Output Format

For each function call, return a JSON object with function name and arguments within `<tool_call></tool_call>` XML tags:

```
<tool_call>[ {"name": "<function-name>", "arguments": {"arg1": "value1", "arg2": "value2", ...} , ...} </tool_call>
```

If no function call is needed, please directly output an empty list `[]` as `<tool_call>[]</tool_call>`.

Available Tools:

In your response, you can use the following tools:

`<tools>`

1. Name: verifyUSAddress

Description: Verify a given US address to ensure it meets USPS standards and is deliverable.

Parameters: { 'addressLine1': { 'type': 'str', 'description': 'The primary address line, including street number and name.', 'required': True }, 'addressLine2': { 'type': 'str', 'description': 'The secondary address line, such as apartment or suite number.', 'required': True }, 'city': { 'type': 'str', 'description': 'The city of the address.', 'required': True }, 'state': { 'type': 'str', 'description': 'The state or territory of the address.', 'required': True }, 'zipCode': { 'type': 'str', 'description': 'The 5-digit ZIP code of the address.', 'required': True } }

2. Name: standardizeUSAddress

Description: Standardize a given US address to create consistency and accuracy in addressing.

Parameters: { 'addressLine1': { 'type': 'str', 'description': 'The primary address line, including street number and name.', 'required': True }, 'addressLine2': { 'type': 'str', 'description': 'The secondary address line, such as apartment or suite number.', 'required': True }, 'city': { 'type': 'str', 'description': 'The city of the address.', 'required': True }, 'state': { 'type': 'str', 'description': 'The state or territory of the address.', 'required': True }, 'zipCode': { 'type': 'str', 'description': 'The 5-digit ZIP code of the address.', 'required': True } }

`</tools>`

Question

User: I'm organizing a mailing list for my business, and I want to make sure all the addresses are standardized. Can you help me standardize this address? 456 Street, Suite 7891, Los Angeles, CA, 90011.

Your Response: `<tool_call>[{"name": "standardizeUSAddress", "arguments": {"addressLine1": "456 Street", "addressLine2": "Suite 7891", "city": "Los Angeles", "state": "CA", "zipCode": "90011"} }] </tool_call>`

Figure 10: Sample example from NexusRaven test data.

SealTool Test Sample Example (ID: 4)

You are an advanced assistant capable of using tools to help the user. You are given a conversation between a user and an assistant, together with the available tools.

You may call one or more functions to assist with the user query.

You will be provided with a set of Available Functions inside `<tools>...</tools>` tags.

For any user request that requires a function, respond by returning a function call inside `<tool_call>...</tool_call>` XML tags, with a JSON object specifying the "name" of the function and the "arguments".

Task

1. Think and recall relevant context, analyze the current user goal.
2. Refer to the previous dialogue records in the conversations, including the user's queries.
3. Decide on which tool to use from **Available Tools** and specify the tool name.
4. At the end, you need to output the JSON object of the function call inside the `<tool_call>` and `</tool_call>` tags.
5. Output format of the function calls must be EXACTLY like in the **Output Format** section, the function calls must be a list of JSON objects, each object must have a "name" key and an "arguments" key.
6. This year is 2023.

Output Format

For each function call, return a JSON object with function name and arguments within `<tool_call></tool_call>` XML tags:

```
<tool_call>[ {"name": "<function-name>", "arguments": {"arg1": "value1", "arg2": "value2", ...} , ...} </tool_call>
```

Available Tools

`<tools>`

1. Name: analyzeSample

Description: Analyze a given sample using analytical chemistry techniques

Field: Chemistry/Analytical chemistry

Parameters: { 'sample': { 'type': 'str', 'description': 'The sample to be analyzed' }, 'method': { 'type': 'str', 'description': 'The analytical method to be used for analysis (e.g., chromatography, spectroscopy)' }, 'instrument': { 'type': 'str', 'description': 'The instrument or equipment to be used for analysis (e.g., gas chromatograph, mass spectrometer)' }, 'conditions': { 'type': 'str', 'description': 'Any specific conditions required for the analysis (e.g., temperature, pressure)' } }

Required: [sample, method]

Responses: { 'results': { 'type': 'str', 'description': 'The analysis results containing information about the sample' } }

2. Name: analyzeEvidence

Description: Analyze the chemical evidence collected from a crime scene

Field: Chemical Engineering/Forensic engineering

Parameters: { 'evidence_type': { 'type': 'str', 'description': 'The type of evidence to be analyzed (e.g., DNA, fingerprints, blood, fibers)' }, 'method': { 'type': 'str', 'description': 'The method or technique to be used for analysis (e.g., spectroscopy, chromatography, microscopy)' }, 'sample': { 'type': 'str', 'description': 'The sample or specimen to be analyzed (e.g., crime scene swab, hair strand, fabric sample)' } }

Required: [evidence_type, method, sample]

Responses: { 'analysis_results': { 'type': 'str', 'description': 'The results of the chemical analysis of the evidence' }, 'conclusion': { 'type': 'str', 'description': 'The conclusion drawn from the analysis' } }

3. Name: getSampleSize

Description: Retrieve the sample size of a mixed methods research study

Field: Research/Mixed Methods Research

Parameters: { 'study_id': { 'type': 'str', 'description': 'The unique identifier of the research study' } }

Required: [study_id]

Responses: { 'sample_size': { 'type': 'int', 'description': 'The sample size of the research study' } }

4. Name: getFabricComposition

Description: Retrieve fabric composition information for a specific clothing item

Field: Fashion/Fashion Technology

Parameters: { 'clothing_item': { 'type': 'str', 'description': 'The type of clothing item for which you want fabric composition (e.g., t-shirt, jeans, dress)' }, 'brand': { 'type': 'str', 'description': 'The brand of the clothing item (e.g., Nike, Zara, Gucci)' } }

Required: [clothing_item]

Responses: { 'composition': { 'type': 'str', 'description': 'The fabric composition of the specified clothing item' }, 'brand': { 'type': 'str', 'description': 'The brand of the clothing item' } }

5. Name: evaluateDataBias

Description: Evaluate data bias in a dataset

Field: Data Analysis/Data Ethics

Parameters: { 'dataset': { 'type': 'str', 'description': 'The dataset to evaluate for bias (e.g., hiring records, loan applications)' }, 'protected_attributes': { 'type': 'str', 'description': 'The protected attributes to consider for bias assessment (e.g., gender, race)' }, 'measures': { 'type': 'str', 'description': 'The bias assessment measures to be used (e.g., disparate impact, statistical parity index)' }, 'reference_group': { 'type': 'str', 'description': 'The reference group to compare with for bias assessment' } }

Required: [dataset, protected_attributes]

Responses: { 'bias_score': { 'type': 'float', 'description': 'The overall bias score of the dataset' }, 'protected_attributes_bias': { 'type': 'str', 'description': 'Detailed bias assessment for each protected attribute' } }

`</tools>`

Input

User: Provide the statistics for the Real Madrid team.

```
Your Response: <tool_call>[ {"name": "getTeamStats", "arguments": {"team": "Real Madrid"} } ] textbf</tool_call>
```

Figure 11: Sample example from SealTool test data.

API-Bank Test Sample Example (ID: 0)

You are an advanced assistant capable of using tools to help the user. You are given a conversation between a user and an assistant, together with the available tools.

You may call one or more functions to assist with the user query.

For any user request that requires a function, respond by returning a function call inside `<tool_call>...</tool_call>` XML tags, with a JSON object specifying the "name" of the function and the "arguments".

Task

1. Think and recall relevant context, analyze the current user goal.
2. Refer to the previous dialogue records in the conversations, including the user's queries.
3. Decide on which tool to use from **Available Tools** and specify the tool name.
4. At the end, you need to output the JSON object of the function call inside the `<tool_call>` and `</tool_call>` tags.
5. Output format of the function calls must be EXACTLY like in the **Output Format** section, the function calls must be a list of JSON objects, each object must have a "name" key and an "arguments" key.
6. This year is 2023.

Output Format

For each function call, return a JSON object with function name and arguments within `<tool_call></tool_call>` XML tags:

```
<tool_call>[{"name": "<function-name>", "arguments": {"arg1": "value1", "arg2": "value2", ...} , ...] </tool_call>
```

Available Tools:

In your response, you can use the following tools:

`<tools>`

1. Name: QueryHealthData

Description: This API queries the recorded health data in database of a given user and time span.

Parameters: {'user_id': {'type': 'str', 'description': 'The user id of the given user. Cases are ignored.'}, 'start_time': {'type': 'str', 'description': 'The start time of the time span. Format: %Y-%m-%d %H:%M:%S'}, 'end_time': {'type': 'str', 'description': 'The end time of the time span. Format: %Y-%m-%d %H:%M:%S'}}

2. Name: CancelRegistration

Description: This API cancels the registration of a patient given appointment ID.

Parameters: {'appointment_id': {'type': 'str', 'description': 'The ID of appointment.'}}

3. Name: ModifyRegistration

Description: This API modifies the registration of a patient given appointment ID.

Parameters: {'appointment_id': {'type': 'str', 'description': 'The ID of appointment.'}, 'new_appointment_date': {'type': 'str', 'description': 'The new appointment date. Format: %Y-%m-%d.'}, 'new_appointment_doctor': {'type': 'str', 'description': 'The new appointment doctor.'}}

`</tools>`

Conversation

User: Can you please modify my appointment scheduled for March 25th with Dr. Kim to March 26th with Dr. Lee?

Assistant: Sure, I can help you with that. Please provide me with the appointment ID and the new appointment date and doctor's name.

User: The appointment ID is 34567890 and the new date is March 26th with Dr. Lee.

Assistant: Alright, I'll modify your appointment now.

User: Based on our conversation above, please only make one tool call to solve my need.

Output: `[<tool_call>[{"name": "ModifyRegistration", "arguments": {"appointment_id": "34567890", "new_appointment_date": "2023-03-26", "new_appointment_doctor": "Dr. Lee"}}]</tool_call>]`

Figure 12: Sample example from API-Bank test data.

ToolAlpaca Test Sample Example (ID: 35)

You are an advanced assistant capable of using tools to help the user.

You may call one or more functions to assist with the user query.

You will be provided with a set of Available Functions inside `<tools>...</tools>` tags.

For any user request that requires a function, respond by returning a function call inside `<tool_call>...</tool_call>` XML tags, with a JSON object specifying the "name" of the function and the "arguments".

Task Instruction

In order to complete the user's request, you need to select one or more appropriate tools from the following tools and fill in the correct values for the tool parameters. Your specific tasks are:

1. Make one or more function/tool calls to meet the request based on the question.
2. If none of the function can be used, point it out as empty list and refuse to answer.
3. If the given question lacks the parameters required by the function, also point it out.

Output Format

For each function call, return a JSON object with function name and arguments within `<tool_call></tool_call>` XML tags:

`<tool_call>[{"name": "<function-name>", "arguments": {"arg1": "value1", "arg2": "value2", ...}, ...}]</tool_call>`

If no function call is needed, please directly output an empty list `[]` as `<tool_call>[]</tool_call>`

Available Tools:

In your response, you can use the following tools:

`<tools>` 1. Name: `airports_get`

Description: Get an airport by its ICAO or FAA identifier

Parameters: `{'apt': {'type': 'string', 'description': 'FAA or ICAO facility identifier (KAVL or AVL). Separate multiple entries with a comma. Required is true.', 'required': True}}`

2. Name: `charts_get`

Description: Get charts for a specified airport

Parameters: `{'apt': {'type': 'string', 'description': 'FAA or ICAO airport identifier (KAVL or AVL). Separate multiple entries with a comma. Required is true.', 'required': True}, 'group': {'type': 'integer', 'description': 'Optional grouping of the charts. 1 -> General, Departures, Arrivals, Approaches; 2 -> Airport Diagram only; 3 -> General only; 4 -> Departures only; 5 -> Arrivals only; 6 -> Approaches only; 7 -> Everything but General.'}}`

3. Name: `charts_changes_get`

Description: Get chart changes by airport or chart name

Parameters: `{'apt': {'type': 'string', 'description': 'FAA or ICAO airport identifier (KAVL or AVL). Required is true.', 'required': True}, 'chart_name': {'type': 'string', 'description': 'Partial or full name of the chart/procedure.'}}`

4. Name: `charts_afd_get`

Description: Get the AFD for a specified airport

Parameters: `{'apt': {'type': 'string', 'description': 'FAA or ICAO airport identifier (KCLT or CLT). Required is true.', 'required': True}}`

5. Name: `preferred-routes_get`

Description: Get all of the preferred routes

Parameters: `{}`

`</tools>`

Input

User: I'm planning a trip to Chicago next week. Can you check the weather conditions at O'Hare International Airport (ICAO: ORD) for me? Also, I'd like to know the runway length and the type of surface of the longest runway there.

Your Response: `<tool_call>[{"name": "weather_metar_get", "parameters": {"apt": "ORD"}}, {"name": "airports_get", "parameters": {"apt": "ORD"}}]</tool_call>`

Figure 13: Sample example from ToolAlpaca test data.