RTTC: Reward-Guided Collaborative Test-Time Compute

J. Pablo Muñoz*
Intel Labs
Santa Clara, CA, USA
jpablomch@gmail.com

Jinjie Yuan*
Intel Corporation
Beijing, China
jinjie.yuan@intel.com

Abstract

Test-Time Compute (TTC) has emerged as a powerful paradigm for enhancing the performance of Large Language Models (LLMs) at inference, leveraging strategies such as Test-Time Training (TTT) and Retrieval-Augmented Generation (RAG). However, the optimal adaptation strategy varies across queries, and indiscriminate application of TTC strategy incurs substantial computational overhead. In this work, we introduce **R**eward-Guided **T**est-**T**ime **C**ompute (**RTTC**), a novel framework that adaptively selects the most effective TTC strategy for each query via a pretrained reward model, maximizing downstream accuracy across diverse domains and tasks. RTTC operates in a distributed server-client architecture, retrieving relevant samples from a remote knowledge base and applying RAG or lightweight fine-tuning on client devices only when necessary. To further mitigate redundant computation, we propose Query-State Caching, which enables the efficient reuse of historical query states at both retrieval and adaptation levels. Extensive experiments across multiple LLMs and benchmarks demonstrate that RTTC consistently achieves superior accuracy compared to vanilla RAG or TTT, validating the necessity of adaptive, reward-guided TTC selection and the potential of RTTC for scalable, high-performance language model adaptation.

1 Introduction

Large language models have achieved remarkable performance across a wide range of tasks. However, their robustness and adaptability to new domains or distribution shifts at inference time remain open challenges. Traditionally, two major paradigms have been explored to enhance LLM performance at test time: Retrieval-Augmented Generation (RAG) (Lewis et al., 2020; Gao et al., 2023b), which augments model input with retrieved knowledge, and Test-Time Training (TTT) (Sun et al., 2020; Hardt and Sun, 2024; Hübotter et al., 2024; Snell et al., 2025), which adapts model parameters using relevant samples. Both approaches have demonstrated significant effectiveness.

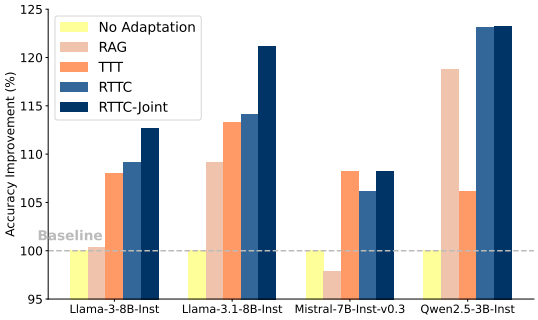

Figure 1: Performance overview of RTTC on downstream tasks across various LLMs. "Accuracy Improvement" indicates the relative average score on five downstream tasks evaluated in §4.

TTC Strategy	Latency	Memory	Accuracy
No Adaptation (Direct Inference)	+	+	+
Retrieval-Augmented Generation (RAG)	++	++	+++
Test-Time Training (TTT)	+++	+++	+++

Table 1: Comparison of computational cost and accuracy boosting benefits for different Test-Time Compute (TTC) strategies. More plus signs indicate higher cost or increased accuracy. While RAG and TTT both provide significant accuracy improvements, each has its strengths, depending on the query and scenario.

However, the practical deployment of RAG and TTT raises two concerns: accuracy and efficiency. **Accuracy:** The effectiveness of RAG or TTT also varies across queries. Sometimes RAG outperforms TTT, and vice versa. For some inputs, the model's direct response is already sufficiently accurate, while for others, the inference stage may benefit from retrieval augmentation or adaptive fine-tuning. **Efficiency:** Both RAG and TTT introduce significant computational overheads—RAG increases inference latency and memory usage by expanding the input context, while TTT requires

*Equal contribution.

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 24793–24809
November 4-9, 2025

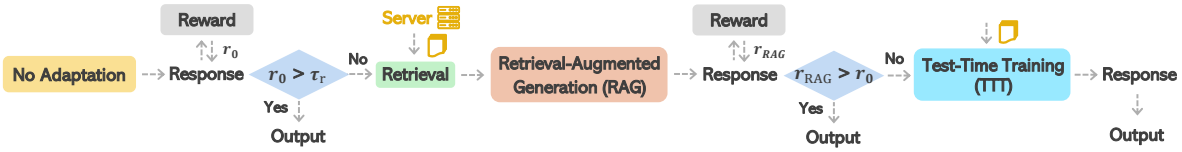

Figure 2: Workflow of **R**eward-guided **T**est-**T**ime **C**ompute (**RTTC**). For each query, a pretrained reward model evaluates candidate responses and selects the optimal adaptation strategy (No Adaptation, RAG, or TTT). τ_r is a predefined threshold.

additional fine-tuning steps and memory for model updates. Moreover, naively applying RAG or TTT to every query can lead to unnecessary computation and inefficient resource utilization. A summary of the computational cost and accuracy associated with each strategy is provided in Table 1. Therefore, **an adaptive approach that can dynamically select the optimal strategy at test-time for each query is crucial for maximizing performance while minimizing overhead**.

Aiming to tackle the above challenges, we propose **R**eward-guided **T**est-**T**ime **C**omputing (**RTTC**). This framework dynamically selects among three strategies for each query: No Adaptation (i.e., returning the model's response without adaptation), Retrieval-Augmented Generation (RAG), and Test-Time Training (TTT). At the core of RTTC is a pretrained reward model that evaluates candidate responses and guides the system to choose the most effective adaptation strategy in a query-adaptive manner (see Figure 2). This reward-guided collaboration enables RTTC to adaptively exploit the most suitable strategy for each query, achieving robust downstream performance improvements across diverse domains and tasks. Unlike prior work that statically applies either RAG or TTT, our approach introduces a principled decision-making mechanism that maximizes performance. Additionally, RTTC also introduces the Query-State Caching (QSC) mechanism to further optimize the test-time efficiency. QSC leverages historical query embeddings and their associated retrieved samples or fine-tuned model state (e.g., LoRA (Hu et al., 2022)) to potentially bypass the need for repeated retrieval and fine-tuning, thus reducing computational overhead and latency. Overall, our main contributions are:

1. We introduce RTTC, a reward-guided collaborative test-time compute framework. We design an effective decision process that leverages a pre-trained reward model to adaptively choose the optimal inference strategy, enabling robust LLM adaptation.

2. We further propose a Query-State Caching (QSC) mechanism that reuses historical query information, reducing redundant computation and latency during inference.

3. Extensive experiments demonstrate that RTTC consistently outperforms baselines, achieving higher accuracy across multiple LLMs and downstream tasks.

In summary, our work is the first to unify direct inference, RAG, and TTT within a reward-guided, query-adaptive framework. The remainder of the paper is organized as follows. We discuss related work in §2. Then, §3 describes the RTTC system, while §4 presents results on various downstream tasks. Our final thoughts are in §5.

2 Related Work

Test-Time Compute. Test-time compute techniques have been proposed as an alternative to scaling model parameters for improving model performance (Snell et al., 2025). Notable strategies include *Chain-of-Thought (CoT) prompting* (Wei et al., 2022) and few-shot learning (Brown et al., 2020). CoT prompting guides the model through intermediate steps to break down complex tasks, enhancing its ability to handle intricate queries. Few-shot learning helps the model adapt to new tasks with just a few examples. Other test-time compute methods include verifying the model's results, for example, through code execution (Brown et al., 2025).

Retrieval-Augmented Generation (RAG). RAG has emerged as a prominent paradigm for enhancing model performance by incorporating external knowledge at inference time (Lewis et al., 2020; Gao et al., 2023b). By retrieving relevant documents and augmenting the model's input, RAG enables LLMs to access up-to-date or domain-specific information beyond their pretraining corpus. This

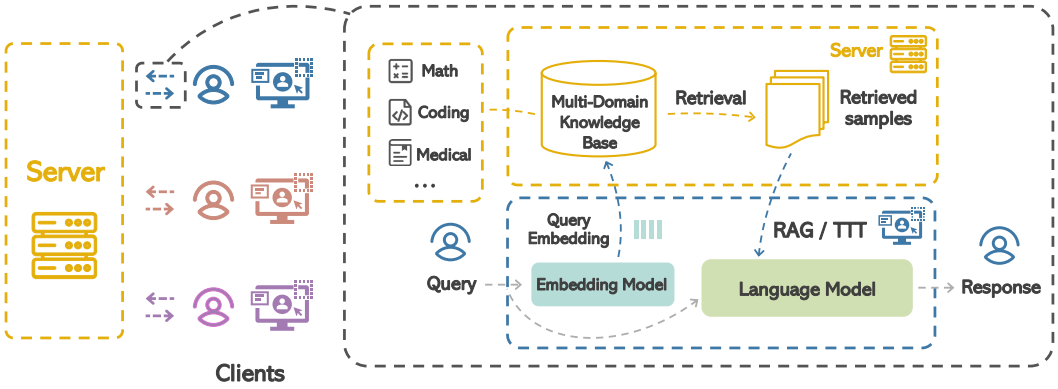

Figure 3: Overview of the retrieval and test-time compute stages. When the reward model determines that the LLM response without adaptation does not meet expectations, relevant samples are retrieved from a remote multi-domain knowledge base. By leveraging advanced test-time compute strategies (RAG or TTT), RTTC improves model performance on client devices.

approach has demonstrated strong results across open-domain question answering, fact verification, and knowledge-intensive tasks. RAG introduces additional computational overhead due to retrieval and longer input sequences, and its effectiveness can vary depending on the quality of retrieved content and the task.

Test-Time Training (TTT). TTT (Hardt and Sun, 2024; Hübotter et al., 2024; Akyürek et al., 2024) has proven effective for adapting models to distribution shifts by fine-tuning on retrieved samples during inference. Recent work, such as SIFT (Hübotter et al., 2024), has improved retrieval strategies for TTT, while Omni-ARC (IronbarArc24, 2024) leveraged TTT to achieve state-of-the-art results in the ARC-AGI challenge (ARC-AGI, 2025; Chollet, 2019). Despite these advances, TTT can incur significant computational and memory costs.

System Considerations. In distributed settings, prior work has explored scalable retrieval and adaptation using distributed indexes and multi-server architectures to accelerate query processing over large datasets (Hardt and Sun, 2024; Douze et al., 2024). While such approaches focus on efficient data access and retrieval, our work emphasizes adaptive test-time compute and downstream task performance. RTTC can flexibly incorporate these distributed retrieval techniques to further optimize efficiency when needed.

Next, we explore RTTC, a system that enhances model performance via reward-guided collaborative test-time compute.

3 RTTC System

This section outlines the architecture and workflow of Reward-Guided Collaborative Test-Time Compute (RTTC), a system designed to optimize the performance of large language models (LLMs). RTTC unifies direct inference, retrieval-augmented generation (RAG), and test-time training (TTT) within a reward-driven, query-adaptive framework, leveraging a remote multi-domain knowledge base for robust adaptation. The overall workflow is shown in Figure 2 and formally summarized in Algorithm 1.

3.1 Reward-Guided Test-Time Compute Pipeline

Given an input query $x \in \mathcal{X}$, RTTC orchestrates a multi-stage adaptive inference process, guided by a pretrained reward model R that is utilized to dynamically select the most effective computation strategy for each query x. The pipeline proceeds as follows:

Step 1. Initial Inference and Reward Evaluation Upon receiving a query x, the LLM M_0 generates an initial response $\hat{y}_0 = M_0(x)$. This response is assessed by the pretrained reward model R, which estimates its quality $r_0 = R(x, \hat{y}_0)$. If r_0 surpasses a predefined threshold τ_r, the system returns $\hat{y}_0$, minimizing latency and computational overhead.

Step 2. Retrieval of Relevant Knowledge When the initial response does not meet the requirements we set (i.e., $r_0 < \tau_r$), the system transitions to a retrieval phase. The query x is encoded into a dense embedding $\mathbf{e}_x = E(x)$ using a shared embedding model E. This embedding is transmitted

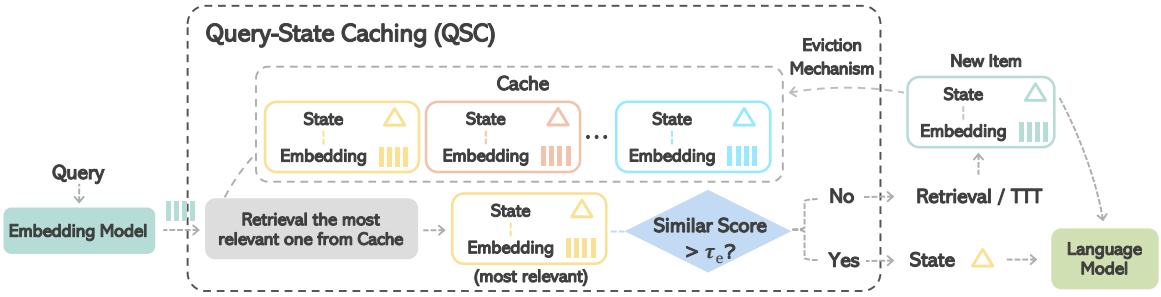

Figure 4: Overview of the Query-State Caching (QSC) strategy for multi-turn caching and efficient test-time compute. QSC is compatible with both RAG and TTT: for RAG, the cached state consists of retrieved samples; for TTT, the cached state stores the fine-tuned model state (e.g., LoRA adapters (Hu et al., 2022)). By managing historical query embeddings and their associated states, QSC accelerates the test-time compute cost (retrieval and fine-tuning).

to a remote server hosting a multi-domain knowledge base $\mathcal{D}$, which returns a set of relevant samples $\mathcal{S}_k = \{(x_i, y_i)\}_{i=1}^k$ identified via similarity search (e.g., using FAISS (Douze et al., 2024) and SIFT(Hübotter et al., 2024) algorithms). The details of distributed retrieval workflow are illustrated in Figure 3.

Step 3. Retrieval-Augmented Generation (RAG) The retrieved samples $\mathcal{S}_k$ are prepended to the original query, forming an augmented input $x' = [\mathcal{S}_k; x]$. The LLM then generates a new response $\hat{y}_{\text{RAG}} = M_0(x')$ conditioned on this expanded context. The reward model re-evaluates the new response, yielding $r_{\text{RAG}} = R(x', \hat{y}_{\text{RAG}})$. If $r_{\text{RAG}} > r_0$, $\hat{y}_{\text{RAG}}$ is returned as the final output.

Algorithm 1 Reward-Guided Test-Time Compute Pipeline

Input: Query x; LLM M_0; reward model R; embedding model E; knowledge base $\mathcal{D}$; threshold τ_r.

Initial Inference: Generate $\hat{y}_0 = M_0(x)$.
Reward Evaluation: Compute $r_0 = R(x, \hat{y}_0)$.
if $r_0 \geq \tau_r$ **then**
  Return $\hat{y}_0$
else
  Retrieval: Encode x as $\mathbf{e}_x = E(x)$.
  Retrieve relevant samples $\mathcal{S}_k = \{(x_i, y_i)\}_{i=1}^k$ from $\mathcal{D}$ using similarity search.
  RAG Inference: Form $x' = [\mathcal{S}_k; x]$ and generate $\hat{y}_{\text{RAG}} = M_0(x')$.
  Compute $r_{\text{RAG}} = R(x, \hat{y}_{\text{RAG}})$.
  if $r_{\text{RAG}} > r_0$ **then**
    Return $\hat{y}_{\text{RAG}}$
  else
    Test-Time Training: Adapt M_0 on $\mathcal{S}_k$ to obtain $M_{\text{TTT}} = \textsc{Train}(M_0, \mathcal{S}_k)$.
    Generate final response $\hat{y}_{\text{TTT}} = M_{\text{TTT}}(x)$.
    Return $\hat{y}_{\text{TTT}}$
  end if
end if

Step 4. Test-Time Training (TTT) If neither direct inference nor RAG yields a satisfactory response, RTTC invokes test-time training. The same retrieved samples $\mathcal{S}_k$ are used to perform lightweight, query-specific fine-tuning of the LLM, resulting in an adapted model $\mathcal{M}_{TTT} = \textsc{Train}(\mathcal{M}_0, \mathcal{S}_k)$ via LoRA (Hu et al., 2022). The adapted model generates the final response $\hat{y}_{\text{TTT}} = M_{TTT}(x)$.

Alternative: Joint RAG and TTT Decision In addition to the sequential decision process described above, RTTC also supports a joint strategy wherein both RAG and TTT are executed in parallel for queries where the initial response is insufficient. The system then returns the response (either $\hat{y}_{\text{RAG}}$ or $\hat{y}_{\text{TTT}}$) with the higher reward score as determined by the reward model. While this approach can further enhance robustness by consistently selecting the best available response, it incurs additional computational and latency overhead due to the need to perform both TTT and reward evaluation for some queries. In practice, this joint strategy is optional and can be selectively enabled for scenarios where maximizing response quality is prioritized over efficiency.

3.2 Distributed Architecture

RTTC is implemented in a server-client paradigm, where the remote server maintains the knowledge base $\mathcal{D}$ and handles retrieval, while all inference, reward evaluation, and adaptation steps are performed locally on the client device. This design can help mitigate privacy risks by keeping sensitive inference and eliminates the need to store the knowledge base locally, thereby significantly reducing memory overhead on client devices. An

overview of the distributed retrieval workflow is illustrated in Figure 3, which highlights how relevant samples are retrieved from a remote multi-domain knowledge base to support advanced test-time compute strategies on client devices.

Algorithm 2 Query-State Caching (QSC)

Input:
Current query embedding e_{x^t}; set of historical query embeddings $\mathcal{Q}$;
Reuse threshold τ_e; budget b; similarity metric γ; eviction mechanism κ.

[RAG] Input: RAG cache $\mathcal{C}_{\text{RAG}} : e_x \rightarrow$ retrieved samples.
[RAG] Output: S_t^{RAG} (retrieved samples).

$e_{x^*} = \underset{e_{x^i} \in \mathcal{Q}}{\text{argmin}}\, \gamma(e_{x^i}, e_{x^t})$
if $\gamma(e_{x^*}, e_{x^t}) > \tau_e$ **then**
  $S_t^{\text{RAG}} \leftarrow \mathcal{C}_{\text{RAG}}[e_{x^*}]$ // RAG cache hit
else
  $S_t^{\text{RAG}} \leftarrow \text{RETRIEVESAMPLES}(e_{x^t})$
  if $|\mathcal{C}_{\text{RAG}}| \geq b$ **then**
    $\mathcal{U} = \{e^r_{x^1} ... e^r_{x^m}\} = \kappa(\mathcal{C}_{\text{RAG}})$
    $\mathcal{C}_{\text{RAG}} \leftarrow \mathcal{C}_{\text{RAG}} \setminus \mathcal{U}$
  end if
  $\mathcal{C}_{\text{RAG}}[e_{x^t}] \leftarrow S_t^{\text{RAG}}$
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{e_{x^t}\}$
end if

[TTT] Input: TTT cache $\mathcal{C}_{\text{TTT}} : e_x \rightarrow$ trained adapters; retrieved samples S_t^{RAG}; initial adapter S_0.
[TTT] Output: S_t^{TTT} (trained adapters).

$e_{x^*} = \underset{e_{x^i} \in \mathcal{Q}}{\text{argmin}}\, \gamma(e_{x^i}, e_{x^t})$
if $\gamma(e_{x^*}, e_{x^t}) > \tau_e$ **then**
  $S_t^{\text{TTT}} \leftarrow \mathcal{C}_{\text{TTT}}[e_{x^*}]$ // TTT cache hit
else
  $S_t^{\text{TTT}} \leftarrow \text{TRAIN}(S_0, S_t^{\text{RAG}})$ // Use the same retrieved samples as RAG for TTT
  if $|\mathcal{C}_{\text{TTT}}| \geq b$ **then**
    $\mathcal{U} = \{e^r_{x^1} ... e^r_{x^m}\} = \kappa(\mathcal{C}_{\text{TTT}})$
    $\mathcal{C}_{\text{TTT}} \leftarrow \mathcal{C}_{\text{TTT}} \setminus \mathcal{U}$
  end if
  $\mathcal{C}_{\text{TTT}}[e_{x^t}] \leftarrow S_t^{\text{TTT}}$
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{e_{x^t}\}$
end if

3.3 Query-State Caching (QSC)

The retrieval and fine-tuning stages at the client introduce notable computational and latency overhead in the RTTC pipeline. To address this, we propose Query-State Caching (QSC), a unified caching strategy at both the retrieval (RAG) and model state (TTT) levels. As described in Algorithm 2 and illustrated in Figure 4, QSC maintains a set of historical query embeddings $\mathcal{Q}$ and two corresponding caches: one mapping embeddings to retrieved samples for RAG, and another mapping embeddings to fine-tuned adapters for TTT. For each new query, the most similar historical embedding is identified using a similarity metric γ. If the similarity exceeds a reuse threshold τ_e, the corresponding cached state (retrieved samples for RAG or adapters for TTT) is reused, allowing the system to bypass redundant retrieval or fine-tuning. Otherwise, new retrieval or fine-tuning is performed, and the cache is updated accordingly, with an eviction mechanism κ ensuring the cache stays within a fixed budget b. This unified approach substantially reduces redundant computation and latency, enabling efficient and scalable test-time adaptation.

4 Experiments

We implement a prototype of the RTTC system as a testbed, demonstrating the potential benefits in a larger deployment. Next, we discuss the resources utilized in our experimentation, followed by results demonstrating the benefits of enabling RTTC.

4.1 Setup

Knowledge base To rigorously evaluate the RTTC prototype, we construct a comprehensive multi-domain knowledge base by integrating several representative datasets spanning Coding, Math, and Medical domains. This diverse collection ensures robust and generalizable evaluation across tasks. For further details regarding dataset composition and sources, please refer to Appendix §A.

Evaluation We comprehensively evaluate the RTTC prototype across Coding, Math, and Medical domains using a suite of established benchmarks and evaluation tools. For the coding domain, we assess performance on MBPP (Austin et al., 2021) and HumanEval (Chen et al., 2021) using the Bigcode-Evaluation-Harness (Ben Allal et al., 2022). In the math domain, we evaluate on MathQA (Amini et al., 2019) with the LLM-Adapters evaluation scripts (Zhiqiang et al., 2023), and on GSM-Plus (Li et al., 2024) using LM-Eval-Harness (Gao et al., 2023a). For the medical domain, we employ MedConceptsQA ATC (Pal et al., 2022), also evaluated with LM-Eval-Harness; we refer to this task as ATC for brevity. All experiments are conducted under the zero-shot setting to reflect real-world deployment scenarios. For experimental efficiency, we evaluate RTTC on subsets of some benchmarks: 200 samples for MathQA and GSM-Plus, and 600 samples for ATC. The details of all evaluation tasks are summarized in Table 2.

Models We test our method on various LLMs, including LLAMA-3-8B-INST, LLAMA-3.1-8B-

Domain	Task	Evaluation Tool	# Samples
Coding	MBPP (Austin et al., 2021)	Bigcode-Evaluation-Harness (Ben Allal et al., 2022)	500
	HumanEval (Chen et al., 2021)		164
Math	MathQA (Amini et al., 2019)	LLM-Adapters (Zhiqiang et al., 2023)	200*
	GSM-Plus (Li et al., 2024)	LM-Eval-Harness (Gao et al., 2023a)	200*
Medical	MedConceptsQA ATC (Pal et al., 2022)		600*

Table 2: Details of the tasks evaluated in the experiments. *Only a subset of the original test set is used for these tasks (200 samples for MathQA and GSM-Plus, 600 for ATC) to increase experimental efficiency.

Model	Strategy	MBPP	HumanEval	MathQA*	GSM-Plus*	ATC*	Avg.	Impr.	Strategy Distribution (%)
LLAMA-3-8B-INST	No Adaptation	51.6	54.9	29.0	19.5	36.8	38.4	–	100% / – / –
	RAG	42.0	48.8	**39.5**	20.5	**41.8**	38.5	100.4%	– / 100% / –
	TTT	49.0	54.9	37.0	29.5	36.7	41.4	108.0%	– / – / 100%
	RTTC	**53.6**	56.1	39.0	23.5	37.3	41.9	109.2%	13.3% / 26.6% / 60.1%
	RTTC-Joint	50.4	**57.9**	35.5	**33.0**	39.3	**43.2**	**112.7%**	13.3% / 30.2% / 56.6%
LLAMA-3.1-8B-INST	No Adaptation	52.2	59.8	16.5	20.0	38.5	37.4	–	100% / – / –
	RAG	42.4	51.8	**32.5**	33.0	**44.3**	40.8	109.2%	– / 100% / –
	TTT	51.6	**63.4**	25.0	34.0	37.8	42.4	113.3%	– / – / 100%
	RTTC	54.0	61.0	29.0	30.5	38.8	42.7	114.1%	6.6% / 25.8% / 67.6%
	RTTC-Joint	**55.2**	62.8	31.0	**37.5**	40.2	**45.3**	**121.2%**	6.6% / 23.9% / 69.5%
MISTRAL-7B-INST-V0.3	No Adaptation	37.6	33.5	28.0	15.0	23.8	27.6	–	100% / – / –
	RAG	30.2	28.1	**35.0**	16.0	**25.8**	27.0	97.9%	– / 100% / –
	TTT	**38.4**	**38.4**	32.0	16.5	24.0	**29.9**	**108.2%**	– / – / 100%
	RTTC	32.4	37.2	32.0	21.0	23.8	29.3	106.1%	23.2% / 24.9% / 51.9%
	RTTC-Joint	33.4	36.6	32.5	**22.5**	24.3	**29.9**	**108.2%**	23.2% / 29.4% / 47.4%
QWEN2.5-3B-INST	No Adaptation	41.2	26.2	26.0	32.5	**26.5**	30.5	–	100% / – / –
	RAG	38.8	41.5	**32.5**	42.0	26.3	36.2	118.8%	– / 100% / –
	TTT	42.4	25.6	30.5	37.0	26.3	32.4	106.2%	– / – / 100%
	RTTC	47.4	**43.3**	28.5	42.0	**26.5**	37.5	123.1%	42.8% / 18.1% / 39.1%
	RTTC-Joint	**48.0**	42.7	27.5	**43.5**	26.2	**37.6**	**123.2%**	42.8% / 15.7% / 41.5%

Table 3: Performance comparison of different adaptation strategies across representative LLMs and evaluation tasks. "Impr." denotes the relative improvement over the No Adaptation baseline. "Strategy Distribution (%)" reports the proportion of queries handled by each branch in the RTTC pipeline: No Adaptation, RAG, and TTT, respectively. RTTC-Joint corresponds to the "Alternative: Joint RAG and TTT Decision" described in section 3. All reported results reflect the best performance achieved across the evaluated retrieval sample sizes $\{1, 2, 4, 8, 16\}$ for each method.

INST (Dubey et al., 2024), MISTRAL-7B-INST-V0.3 (Jiang et al., 2023) and QWEN2.5-3B-INST (Yang et al., 2024). For the retrieval stage in RTTC, we utilize QWEN3-EMBEDDING-0.6B (Zhang et al., 2025) as the embedding model. For the reward model, we employ SKYWORK-REWARD-V2-QWEN3-0.6B (Liu et al., 2025).

Hyperparameters and Implementation For RAG and TTT, the number of retrieval samples is selected from $\{1, 2, 4, 8, 16\}$. TTT is performed for two epochs with a learning rate of 5×10^{-5} and a batch size of 1. LoRA fine-tuning is applied in TTT, using a rank of 32 and an alpha of 16, targeting the Query, Key, Value, Up, and Down projection layers. The threshold τ_r is 2.0. For QSC, the reuse threshold τ_e and budget b are set to 0.5 and 8, respectively. The similarity metric γ is the inner product, and the eviction mechanism κ adopts a Least Frequently Used (LFU) policy.

4.2 Main Results

Table 3 presents a comprehensive comparison of adaptation strategies across multiple LLMs and tasks. Several key observations: (1) RTTC consistently outperforms both RAG and TTT baselines, achieving the highest average accuracy improvements across all models and tasks. For example, on LLAMA-3.1-8B-INST, RTTC yields a 114.1% relative improvement over the no adaptation baseline, while the joint variant (RTTC-Joint) further boosts

Strategy	Total Cost
No Adaptation	$N \cdot C_0$
RAG	$N \cdot (C_0 + C_{\text{Ret}} + C_{\text{RAG}})$
TTT	$N \cdot (C_0 + C_{\text{Ret}} + C_{\text{TTT}})$
RTTC	$N \cdot (C_0 + C_{\text{Rew}}) + (d_{\text{RAG}} + d_{\text{TTT}}) \cdot N \cdot (C_{\text{Ret}} + C_{\text{RAG}} + C_{\text{Rew}}) + d_{\text{TTT}} \cdot N \cdot C_{\text{TTT}}$
RTTC-Joint	$N \cdot (C_0 + C_{\text{Rew}}) + (d_{\text{RAG}} + d_{\text{TTT}}) \cdot N \cdot (C_{\text{Ret}} + C_{\text{RAG}} + C_{\text{TTT}} + 2C_{\text{Rew}})$

Table 4: Total cost comparison for N queries under different adaptation strategies. C_0 denotes the base inference cost per query (No Adaptation); C_{Ret}, C_{RAG}, and C_{TTT} represent the additional costs for retrieval, RAG, and TTT, respectively, with $C_{\text{TTT}} > C_{\text{RAG}} > 0$; C_{Rew} is the reward model evaluation cost. For RTTC, d_{RAG} and d_{TTT} indicate the fractions of queries routed to the RAG and TTT branches, as reported in the main results (see "Strategy Distribution (%)" of Table 3).

Metric	No Adapt.	RAG	TTT
Context Length	96.1	96.1	96.1
Token Generation Count	326.6	353.5	344.7
Inference Latency (sec)	7.5	8.7	8.4
Total Latency (sec)	**7.5**	**9.8**	**12.1**
Retrieval:			
Embedding Processing (sec)	-	0.06	0.06
Retrieval (sec)	-	1.06	1.06
RAG:			
Augmented Context Length	-	3058.2	-
TTT:			
Train (sec)	-	-	2.60
Merge (sec)	-	-	0.01
Unmerge (sec)	-	-	0.01
Training Token Count	-	-	5,921.6

Table 5: Performance comparison of RAG and TTT against No Adaptation using MISTRAL-7B-INST-V0.3 on MathQA dataset. Results are averaged over 10 test samples with 8 test-time training samples per query. The experiments were conducted on NVIDIA A100.

performance to 121.2%. (2) The joint decision strategy (RTTC-Joint), which selects the best response between RAG and TTT per query, achieves the best overall results, highlighting the benefit of adaptive, reward-guided selection. (3) The strategy distribution indicates that RTTC predominantly leverages TTT for challenging queries, while efficiently falling back to direct inference or RAG when appropriate, thus balancing accuracy and computational cost. (4) Notably, the effectiveness of RAG and TTT varies by task and model, underscoring the necessity of a unified, query-adaptive framework. Overall, these results demonstrate that RTTC robustly enhances downstream performance across diverse domains and models, validating the effectiveness of reward-guided, collaborative test-time compute.

4.3 Cost Analysis

Table 4 presents a comparative analysis of the computational cost associated with different TTC strategies. While RAG and TTT each introduce substantial additional overhead due to retrieval, longer context or fine-tuning, the cost profile of RTTC is inherently query-adaptive and cannot be strictly characterized as lower or higher than either baseline. The overall cost of RTTC depends on the distribution of queries across its decision branches (see "Strategy Distribution (%)" in Table 3). For queries where the initial response is sufficient, RTTC terminates early, incurring only minimal inference and reward evaluation costs. However, for queries routed to RAG or TTT, RTTC will incur additional overhead compared to vanilla RAG or TTT, as each branch is preceded by an initial inference and reward evaluation step. Thus, RTTC embodies an adaptive early-stopping mechanism, dynamically allocating computation to maximize response quality.

To complement the theoretical cost analysis, Table 5 reports empirical measurements of computational performance across different adaptation strategies. These results, obtained on 10 test samples of MathQA, detail latency and other metrics under realistic deployment conditions. Compared to the baseline (No Adaptation), RAG and TTT increase total latency due to retrieval, augmented context and fine-tuning, with TTT incurring the highest cost. RAG significantly expands context length, while TTT introduces additional training steps; both yield higher token generation counts.

Given that RTTC might introduce additional overhead in certain scenarios, we propose Query-State Caching (QSC) to mitigate redundant retrieval and fine-tuning operations. QSC leverages historical query states to reuse previously computed results, thereby reducing unnecessary computation and latency. The effectiveness of QSC is evaluated in the following subsection.

Model	Query-State Caching	MBPP	HumanEval	MathQA*	GSM-Plus*	ATC*	Avg.	Rel.	RAG Cache Utilization	TTT Cache Utilization
LLAMA-3-8B-INST	✗	**53.8**	55.5	**35.0**	**19.0**	38.8	**40.4**	**100.00%**	–	–
	✓	52.4	**56.1**	33.0	16.5	**39.5**	39.5	97.71%	**66.53%**	**70.11%**
LLAMA-3.1-8B-INST	✗	**55.0**	60.4	**28.0**	26.5	39.3	41.8	100.00%	–	–
	✓	54.4	**64.0**	21.5	**29.0**	**40.3**	**41.9**	**100.02%**	**62.72%**	**67.03%**
MISTRAL-7B-INST-V0.3	✗	**41.0**	**37.2**	**30.5**	**20.5**	23.0	**30.4**	**100.00%**	–	–
	✓	38.4	**37.2**	28.0	20.0	**24.2**	29.6	97.09%	**62.34%**	**70.10%**
QWEN2.5-3B-INST	✗	**42.0**	36.6	24.5	**44.5**	**26.2**	**34.8**	**100.00%**	–	–
	✓	41.2	**40.9**	**25.0**	40.0	25.8	34.6	99.49%	**64.02%**	**69.85%**

Table 6: Performance comparison of RTTC with and without Query-State Caching (QSC). All results use 4 retrieved samples per query for fair comparison (note: this differs from Table 3, which reports the results for the best-performing retrieval sample size). "RAG Cache Utilization" and "TTT Cache Utilization" report the proportion of queries that successfully reused cached retrieval results and cached adapters, respectively, as defined in Algorithm 2. These metrics reflect the effectiveness of QSC in reducing redundant retrieval and fine-tuning operations. "Rel." denotes the relative average performance compared to the baseline (RTTC without QSC).

4.4 Query-State Caching (QSC)

Table 6 presents the evaluation of Query-State Caching (QSC) across multiple LLMs and tasks. The results demonstrate that enabling QSC yields substantial reductions in redundant retrieval and fine-tuning operations, as evidenced by high cache utilization rates for both RAG retrieval sample (62–66%) and TTT adapter (67–70%) caches. Importantly, QSC achieves these efficiency gains with only marginal impact on average task performance, maintaining relative accuracy within 97–100% of the baseline (w/o QSC). This indicates that QSC effectively balances efficiency and quality.

It should be noted that the reported cache utilization rates may be somewhat optimistic due to the experimental protocol, which involves evaluating benchmark samples from the same domain in succession. This sequential testing increases the likelihood of cache hits, thereby inflating the observed utilization. In real-world deployment scenarios with more diverse and interleaved query streams, the actual cache hit rates are expected to be lower. Nevertheless, the results substantiate the potential of QSC to reduce redundant computation while preserving robust downstream performance.

4.5 In-Domain Sample Retrieval

To quantitatively assess the effectiveness of RTTC in retrieving domain-relevant samples from a multi-domain knowledge base, we analyze the domain composition of retrieved samples for each evaluation task. Figure 5 presents a heatmap of the proportional distribution of retrieved samples across all domains for five tasks.

The results demonstrate strong in-domain alignment: for each task, the majority of retrieved samples originate from the corresponding domain, as highlighted by the red-bordered cells. For instance, MBPP and Humaneval retrieve over 97% and 98% of samples from The Stack Python, respectively. This high degree of domain specificity is primarily attributable to the quality of the embedding model, which enables precise semantic matching between queries and knowledge base entries. The effectiveness of RTTC in test-time adaptation fundamentally relies on both a high-quality embedding model and a well-curated, diverse knowledge base, which together ensure that retrieved samples are highly relevant to the current task.

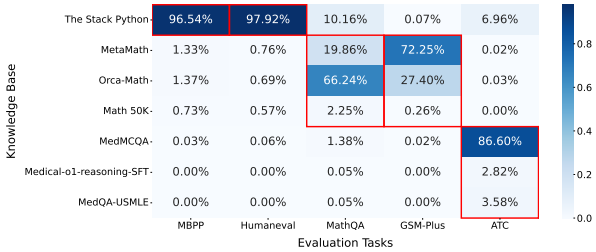

Figure 5: Domain distribution of retrieval samples from knowledge base across evaluation tasks (counted all test samples). The red border indicates the domain corresponding to the current task (in-domain). Higher values within the red border reflect stronger domain alignment.

Additionally, we performed a t-SNE visualization of the embeddings for some samples in the database, as shown in Figure 6. The visualization demonstrates a clear clustering by domain, further illustrating the discriminative power of the embedding model and the importance of database quality. These factors are essential for enabling RTTC to

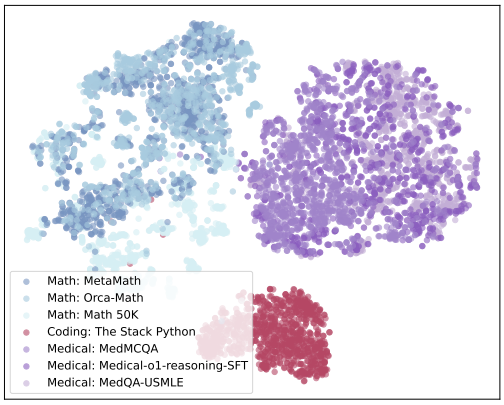

Figure 6: t-SNE visualization of the embeddings for some samples in the knowledge base using the QWEN3-EMBEDDING-0.6B embedding model. Each dataset contains 1000 randomly selected samples.

reliably retrieve domain-specific samples, thereby maximizing the benefit of test-time adaptation.

4.6 Extended Experimental Results

For completeness and to facilitate deeper analysis, we provide additional experimental results and ablation studies in the Appendix. These include comprehensive comparisons across varying numbers of retrieved samples (see Appendix §B, Table 8, Table 9, Table 10, and Table 11), reward thresholds τ_r (see Appendix §C and Table 12), and reward models (see Appendix §D and Table 13). The extended results further substantiate the robustness and generalizability of RTTC across diverse models, tasks, and settings. Readers are referred to the Appendix for full tables and discussion.

5 Conclusion

Test-time Compute (TTC) is an effective paradigm for enhancing model performance at the expense of increased computation during inference. We present RTTC, a system that adaptively selects the optimal TTC strategy for each query at test-time. The results of the RTTC prototype, which utilizes supervisory signals from a knowledge base, serve as a call to action to investigate further improvements to learning at test-time methods. RTTC is guided by a reward model that assists in the decision of the TTC method to apply at inference time. Importantly, the reward-guided approach in RTTC is theoretically extensible to any TTC strategy, providing an open and generalizable direction for future research in adaptive test-time compute. The current version of RTTC works on text queries. With increased sophistication, future versions of RTTC must handle more complex tasks and multimodal queries.

Limitations

Although RTTC produces compelling results and demonstrates that a reward-guided approach can alleviate the challenges of deciding between popular test-time compute techniques at runtime, it also opens new research opportunities for the future. Currently, the proposed approach is limited by the manual definition of several hyperparameters, for instance determining the value for the threshold τ_r to trigger RAG or TTT. This current limitations present exciting opportunities in future work. Another current limitation is related to determining the right data mix in the knowledge base. In real-world applications, the effectiveness of RTTC is likely more pronounced with a more extensive and diverse knowledge base. A larger knowledge base would provide a broader range of samples, potentially improving the relevance and quality of the data retrieved for TTT, thereby further enhancing the model's performance. Our experimental results have demonstrated the feasibility and potential benefits of RTTC.

For experimental efficiency, our TTT experiments employ a single set of hyperparameters (learning rate, number of epochs, LoRA configuration, etc.) across all tasks and models, without extensive hyperparameter exploration. We believe that TTT has significant potential for further improvement, and that more optimal hyperparameter choices could further enhance the performance of RTTC.

In the current version of RTTC's prototype, the server can recover the content of the user's prompt. A real-world solution should incorporate privacy mechanisms to protect the user. In addition to encrypting the query for transmission, e.g., utilizing Secure Sockets Layer (SSL)/Transport Layer Security (TLS), several open research challenges exist to enhance the privacy and handling of the user's content on the server.

Ethical Considerations

Test-time compute (TTC) techniques offer improvements in model performance, making them more accurate, albeit at the cost of increased computation. However, they alone do not solve existing challenges in large foundation models and their smaller

counterparts. Our research explores systems and techniques to enable running TTC in client devices with resource constraints. However, applying our system and techniques to real-world applications must include additional safeguards to prevent hallucinations or intentional misinformation that could negatively impact the well-being of system users. The research community must continue to investigate solutions to address these and other open challenges in popular language models.

References

Ekin Akyürek, Mehul Damani, Linlu Qiu, Han Guo, Yoon Kim, and Jacob Andreas. 2024. The surprising effectiveness of test-time training for abstract reasoning. *arXiv preprint arXiv:2411.07279*.

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2357–2367, Minneapolis, Minnesota. Association for Computational Linguistics.

ARC-AGI. 2025. ARC Prize — arcprize.org. https://arcprize.org. [Accessed 13-03-2025].

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Loubna Ben Allal, Niklas Muennighoff, Logesh Kumar Umapathi, Ben Lipkin, and Leandro von Werra. 2022. A framework for the evaluation of code generation models. https://github.com/bigcode-project/bigcode-evaluation-harness.

Bradley Brown, Jordan Juravsky, Ryan Saul Ehrlich, Ronald Clark, Quoc V Le, Christopher Re, and Azalia Mirhoseini. 2025. Large language monkeys: Scaling inference compute with repeated sampling.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Junying Chen, Zhenyang Cai, Ke Ji, Xidong Wang, Wanlong Liu, Rongsheng Wang, Jianye Hou, and Benyou Wang. 2024. Huatuogpt-o1, towards medical complex reasoning with llms. *Preprint*, arXiv:2412.18925.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

François Chollet. 2019. On the measure of intelligence. *CoRR*, abs/1911.01547.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2023a. A framework for few-shot language model evaluation.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023b. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1).

Moritz Hardt and Yu Sun. 2024. Test-time training on nearest neighbors for large language models. In *International Conference on Learning Representations*.

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

Jonas Hübotter, Sascha Bongni, Ido Hakimi, and Andreas Krause. 2024. Efficiently learning at test-time: Active fine-tuning of llms. *arXiv preprint arXiv:2410.08020*.

IronbarArc24. 2024. arc24 — ironbar.github.io. https://ironbar.github.io/arc24/. [Accessed 13-03-2025].

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego

de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, and 1 others. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. 2020. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *arXiv preprint arXiv:2009.13081*.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. 2022. The stack: 3 tb of permissively licensed source code. *Preprint*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. 2024. Gsm-plus: A comprehensive benchmark for evaluating the robustness of llms as mathematical problem solvers. *Preprint*, arXiv:2402.19255.

Chris Yuhao Liu, Liang Zeng, Yuzhen Xiao, Jujie He, Jiacai Liu, Chaojie Wang, Rui Yan, Wei Shen, Fuxiang Zhang, Jiacheng Xu, Yang Liu, and Yahui Zhou. 2025. Skywork-reward-v2: Scaling preference data curation via human-ai synergy. *arXiv preprint arXiv:2507.01352*.

Arindam Mitra, Hamed Khanpour, Corby Rosset, and Ahmed Awadallah. 2024. Orca-math: Unlocking the potential of slms in grade school math. *Preprint*, arXiv:2402.14830.

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. 2022. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pages 248–260. PMLR.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*.

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 *International conference on machine learning*, pages 9229–9248. PMLR.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, and 1 others. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

Hu Zhiqiang, Lan Yihuai, Wang Lei, Xu Wanyu, Lim EePeng, Lee Roy Ka-Wei, Bing Lidong, and Poria Soujanya. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*.

A Knowledge Base Details

To support rigorous and multi-domain evaluation of the RTTC system, we aggregate a large-scale knowledge base comprising datasets from Coding, Math, and Medical domains. Table 7 summarizes the dataset composition, sample counts, and sources.

Coding Domain We sample 600,000 entries from the Python subset of The Stack (Kocetkov et al., 2022), providing diverse programming knowledge.

Math Domain Three datasets are included: MetaMath (Yu et al., 2023) (395,000 samples), Orca-Math (Mitra et al., 2024) (200,035 samples), and Math 50K (Zhiqiang et al., 2023) (50,000 samples), collectively covering a wide range of mathematical reasoning and problem-solving scenarios.

Medical Domain The medical subset consists of MedMCQA (Pal et al., 2022) (182,822 samples), Medical-o1-reasoning-SFT (Chen et al., 2024) (19,704 samples), and MedQA-USMLE (Jin et al., 2020) (10,178 samples), focusing on medical question answering and reasoning.

All datasets are strictly used for experimental purposes. This knowledge base provides a robust foundation for cross-domain adaptation and benchmarking.

B Detailed Comparison Across Retrieval Sample Sizes

Tables 8–11 present comprehensive results for all evaluated models and adaptation strategies under varying numbers of retrieved samples. Across all settings, both RTTC and RTTC-Joint consistently outperform baseline RAG and TTT approaches, achieving the highest average accuracy and relative improvement. This superiority holds regardless of the retrieval sample size, demonstrating the robustness and effectiveness of reward-guided, query-adaptive selection in collaborative test-time compute.

C Analysis of Reward Threshold

Table 12 presents an evaluation of the effect of the reward threshold (hyper-parameter) on the selection and effectiveness of adaptation strategies within the RTTC framework. The threshold controls the minimum quality required for the initial model response, as assessed by the reward model. Queries with reward scores below this threshold are routed to more advanced adaptation stages (RAG or TTT).

Accuracy vs. Cost Trade-off: As the threshold increases, the average accuracy and relative improvement often improve, reflecting the benefit of more advanced adaptation. However, this comes at the expense of increased computational cost, since more queries undergo RAG and/or fine-tuning. The distribution of queries across No Adaptation, RAG, and TTT branches shifts toward greater use of adaptation strategies as the threshold rises, further illustrating the trade-off between performance and efficiency.

In summary, the reward threshold is a critical parameter for balancing accuracy and efficiency in adaptive test-time compute. Careful tuning is required to achieve optimal results for specific deployment scenarios.

D Analysis of Reward Model

Table 13 presents a comprehensive evaluation of adaptation strategies across different LLMs, reward thresholds, and reward models. Here, "0.6B" denotes the SKYWORK-REWARD-V2-QWEN3-0.6B reward model, and "8B" denotes the SKYWORK-REWARD-V2-LLAMA-3.1-8B reward model.

RTTC relies critically on the reward model to guide adaptive strategy selection. We investigate the impact of increasing reward model size by comparing the 0.6B and 8B variants. As shown in the table, using a larger reward model (8B) does not consistently yield significant improvements in accuracy across tasks. However, the 8B model tends to assign higher reward scores, resulting in a greater proportion of queries being handled by No Adaptation at the same threshold compared to 0.6B reward model. This shift indicates improved efficiency, as fewer queries require costly adaptation. These findings highlight the necessity of tuning the reward threshold for each reward model to achieve optimal performance and efficiency in specific deployment scenarios. Additionally, it is important to consider that larger reward models incur higher inference costs, although this effect is negligible in the overall pipeline.

Domain	Dataset	# Samples	# Samples per Domain	Link
Coding	The Stack Python (Kocetkov et al., 2022)	600,000	600,000	bigcode/the-stack
Math	MetaMath (Yu et al., 2023)	395,000	645,035	meta-math/MetaMathQA
	Orca-Math (Mitra et al., 2024)	200,035		microsoft/orca-math-word-problems-200k
	Math 50K (Zhiqiang et al., 2023)	50,000		math_50k.json
Medical	MedMCQA (Pal et al., 2022)	182,822	212,704	openlifescienceai/medmcqa
	Medical-o1-reasoning-SFT (Chen et al., 2024)	19,704		FreedomIntelligence/medical-o1-reasoning-SFT
	MedQA-USMLE (Jin et al., 2020)	10,178		GBaker/MedQA-USMLE-4-options-hf
		Total:	**1,457,739**	

Table 7: Knowledge base composition. These datasets cover various domains, including coding, math, and medical. The Stack Python dataset consists of a random sample of 600,000 entries from the original Stack dataset (Python).

Number of Retrieval Samples	Strategy	MBPP	HumanEval	MathQA*	GSM-Plus*	ATC*	Avg.	Impr.
/	No Adaptation	51.6	54.9	29.0	19.5	36.8	38.4	–
1	RAG	36.6	41.5	36.5	25.5	38.2	35.7	92.9%
	TTT	52.2	55.5	28.5	20.5	36.8	38.7	100.9%
	RTTC	52.8	53.7	32.0	27.5	37.3	40.7	106.0%
	RTTC-Joint	53.4	54.9	33.5	30.0	37.5	**41.9**	**109.1%**
2	RAG	40.0	45.7	37.0	28.0	40.3	38.2	99.6%
	TTT	51.8	56.1	27.0	21.5	37.2	38.7	100.9%
	RTTC	52.0	54.9	30.5	27.0	37.5	40.4	105.3%
	RTTC-Joint	52.6	54.9	31.0	26.0	38.5	**40.6**	**105.8%**
4	RAG	42.0	48.8	39.5	20.5	41.8	38.5	100.4%
	TTT	52.2	56.1	35.5	19.5	37.2	40.1	104.5%
	RTTC	53.6	56.1	39.0	23.5	37.3	41.9	109.2%
	RTTC-Joint	54.8	56.7	41.5	21.5	37.2	**42.3**	**110.4%**
8	RAG	42.6	49.4	35.5	14.0	41.8	36.7	95.6%
	TTT	52.0	57.9	40.5	16.0	37.5	**40.8**	**106.3%**
	RTTC	51.8	54.9	34.0	19.0	36.0	39.1	102.0%
	RTTC-Joint	52.4	56.1	37.0	18.5	38.5	40.5	105.6%
16	RAG	39.6	48.2	37.0	11.5	41.5	35.6	92.7%
	TTT	49.0	54.9	37.0	29.5	36.7	41.4	108.0%
	RTTC	49.6	54.3	32.5	24.0	36.8	39.4	102.8%
	RTTC-Joint	50.4	57.9	35.5	33.0	39.3	**43.2**	**112.7%**

Table 8: Performance comparison of different adaptation strategies for LLAMA-3-8B-INST. "Impr." denotes the relative improvement over the No Adaptation baseline. RTTC-Joint corresponds to the "Alternative: Joint RAG and TTT Decision" described in section 3.

Number of Retrieval Samples	Strategy	MBPP	HumanEval	MathQA*	GSM-Plus*	ATC*	Avg.	Impr.
/	No Adaptation	52.2	59.8	16.5	20.0	38.5	37.4	–
1	RAG	38.4	45.7	26.0	41.5	41.5	38.6	103.3%
	TTT	51.4	60.4	18.0	18.5	38.8	37.4	100.1%
	RTTC	52.6	60.4	21.5	31.0	38.8	**40.9**	**109.3%**
	RTTC-Joint	52.8	59.8	21.5	30.0	39.7	40.8	109.0%
2	RAG	42.4	51.8	32.5	33.0	44.3	40.8	109.2%
	TTT	51.4	60.4	19.0	23.0	39.0	38.6	103.1%
	RTTC	51.6	59.2	24.0	29.0	41.3	41.0	109.7%
	RTTC-Joint	52.6	59.2	25.0	30.0	40.8	**41.5**	**111.0%**
4	RAG	45.0	50.6	28.0	36.0	43.5	40.6	108.6%
	TTT	52.2	61.0	17.0	20.0	38.8	37.8	101.1%
	RTTC	55.2	61.0	22.0	25.0	40.0	40.6	108.7%
	RTTC-Joint	54.8	60.4	22.5	30.0	40.3	**41.6**	**111.3%**
8	RAG	41.8	53.1	32.5	23.0	43.5	38.8	103.7%
	TTT	51.6	61.0	20.0	22.0	39.5	38.8	103.8%
	RTTC	53.0	59.2	26.5	25.5	41.0	41.0	109.7%
	RTTC-Joint	53.0	60.4	27.5	27.5	41.5	**42.0**	**112.3%**
16	RAG	38.0	54.9	31.0	21.0	41.2	37.2	99.5%
	TTT	51.6	63.4	25.0	34.0	37.8	42.4	113.3%
	RTTC	54.0	61.0	29.0	30.5	38.8	42.7	114.1%
	RTTC-Joint	55.2	62.8	31.0	37.5	40.2	**45.3**	**121.2%**

Table 9: Performance comparison of different adaptation strategies for LLAMA-3.1-8B-INST. "Impr." denotes the relative improvement over the No Adaptation baseline. RTTC-Joint corresponds to the "Alternative: Joint RAG and TTT Decision" described in section 3.

Number of Retrieval Samples	Strategy	MBPP	HumanEval	MathQA*	GSM-Plus*	ATC*	Avg.	Impr.
/	No Adaptation	37.6	33.5	28.0	15.0	23.8	27.6	–
1	RAG	24.6	26.8	29.5	25.0	26.5	26.5	96.0%
	TTT	38.4	38.4	32.0	16.5	24.0	**29.9**	**108.2%**
	RTTC	29.0	34.8	32.0	22.0	26.2	28.8	104.3%
	RTTC-Joint	29.6	37.2	32.0	23.0	24.8	29.3	106.3%
2	RAG	25.4	23.8	33.0	22.0	25.0	25.8	93.6%
	TTT	40.2	36.6	29.0	14.0	23.5	28.7	103.9%
	RTTC	31.2	37.8	34.0	20.0	23.0	29.2	105.8%
	RTTC-Joint	32.4	36.6	36.5	20.5	22.7	**29.7**	**107.8%**
4	RAG	27.8	27.4	29.5	16.0	24.7	25.1	90.9%
	TTT	39.0	37.8	30.5	15.0	23.3	**29.1**	**105.6%**
	RTTC	30.6	37.2	32.0	17.5	23.3	28.1	101.9%
	RTTC-Joint	31.8	36.6	32.5	18.0	24.5	28.7	103.9%
8	RAG	29.4	22.0	32.5	19.5	26.3	25.9	94.0%
	TTT	39.8	32.3	31.5	19.0	24.0	29.3	106.3%
	RTTC	32.4	37.2	32.0	21.0	23.8	29.3	106.1%
	RTTC-Joint	33.4	36.6	32.5	22.5	24.3	**29.9**	**108.2%**
16	RAG	30.2	28.1	35.0	16.0	25.8	27.0	97.9%
	TTT	36.4	29.9	30.0	13.0	24.8	26.8	97.2%
	RTTC	30.8	34.2	35.5	19.0	23.8	28.7	103.9%
	RTTC-Joint	31.4	36.6	35.5	19.5	24.7	**29.5**	**107.0%**

Table 10: Performance comparison of different adaptation strategies for MISTRAL-7B-INST-V0.3. "Impr." denotes the relative improvement over the No Adaptation baseline. RTTC-Joint corresponds to the "Alternative: Joint RAG and TTT Decision" described in section 3.

Number of Retrieval Samples	Strategy	MBPP	HumanEval	MathQA*	GSM-Plus*	ATC*	Avg.	Impr.
/	No Adaptation	41.2	26.2	26.0	32.5	26.5	30.5	–
1	RAG	29.2	23.8	27.5	30.0	26.8	27.5	90.1%
	TTT	42.4	27.4	23.5	32.0	26.0	30.3	99.3%
	RTTC	45.4	40.2	25.5	36.0	26.3	34.7	113.8%
	RTTC-Joint	45.4	41.5	26.0	36.5	25.5	**35.0**	**114.7%**
2	RAG	32.6	25.6	32.0	36.0	25.5	30.3	99.5%
	TTT	39.8	26.8	24.0	31.0	25.8	29.5	96.8%
	RTTC	45.4	39.0	27.5	41.0	25.2	35.6	116.8%
	RTTC-Joint	45.2	40.2	27.0	41.5	26.0	**36.0**	**118.1%**
4	RAG	36.6	36.0	30.5	40.5	27.0	34.1	111.9%
	TTT	41.2	27.4	22.5	32.5	26.0	29.9	98.2%
	RTTC	46.0	41.5	26.5	41.5	26.2	36.3	119.2%
	RTTC-Joint	45.6	43.9	27.0	43.5	26.0	**37.2**	**122.0%**
8	RAG	38.8	41.5	32.5	42.0	26.3	36.2	118.8%
	TTT	41.8	27.4	26.5	35.0	26.2	31.4	103.0%
	RTTC	47.4	43.3	28.5	42.0	26.5	37.5	123.1%
	RTTC-Joint	48.0	42.7	27.5	43.5	26.2	**37.6**	**123.2%**
16	RAG	35.4	37.8	32.5	46.0	25.7	35.5	116.4%
	TTT	42.4	25.6	30.5	37.0	26.3	32.4	106.2%
	RTTC	46.2	41.5	28.0	45.5	25.7	37.4	122.6%
	RTTC-Joint	46.4	45.1	29.0	46.0	26.3	**38.6**	**126.5%**

Table 11: Performance comparison of different adaptation strategies for QWEN2.5-3B-INST. "Impr." denotes the relative improvement over the No Adaptation baseline. RTTC-Joint corresponds to the "Alternative: Joint RAG and TTT Decision" described in section 3.

Model	Strategy	Threshold (τ_r)	MBPP	HumanEval	MathQA*	GSM-Plus*	ATC*	Avg.	Impr.	Strategy Distribution (%)
LLAMA-3-8B-INST	No Adaptation	-	51.6	54.9	29.0	19.5	36.8	38.4	–	100% / – / –
	RAG	-	42.0	48.8	39.5	20.5	41.8	38.5	100.4%	– / 100% / –
	TTT	-	49.0	54.9	37.0	29.5	36.7	41.4	108.0%	– / – / 100%
	RTTC	2.0	53.6	56.1	39.0	23.5	37.3	41.9	109.2%	13.3% / 26.6% / 60.1%
		5.0	53.4	56.1	39.5	22.0	37.3	41.7	108.6%	1.2% / 26.9% / 71.9%
		8.0	53.4	56.1	39.5	22.0	36.8	41.6	108.4%	0.1% / 27.0% / 73.0%
	RTTC-Joint	2.0	50.4	57.9	35.5	33.0	39.3	43.2	112.7%	13.3% / 30.2% / 56.6%
		5.0	54.6	56.7	42.0	20.5	37.7	42.3	110.3%	1.2% / 28.2% / 70.6%
		8.0	54.6	56.7	42.0	20.5	38.2	42.4	110.5%	0.1% / 28.4% / 71.6%
LLAMA-3.1-8B-INST	No Adaptation	-	52.2	59.8	16.5	20.0	38.5	37.4	–	100% / – / –
	RAG	-	42.4	51.8	32.5	33.0	44.3	40.8	109.2%	– / 100% / –
	TTT	-	51.6	63.4	25.0	34.0	37.8	42.4	113.3%	– / – / 100%
	RTTC	2.0	54.0	61.0	29.0	30.5	38.8	42.7	114.1%	6.6% / 25.8% / 67.6%
		5.0	53.6	62.2	30.5	30.0	38.8	43.0	115.1%	1.7% / 23.3% / 75.0%
		8.0	53.6	62.2	30.5	29.5	38.8	42.9	114.8%	0.0% / 23.3% / 76.7%
	RTTC-Joint	2.0	55.2	62.8	31.0	37.5	40.2	45.3	121.2%	6.6% / 23.9% / 69.5%
		5.0	55.2	63.4	32.5	37.0	39.8	45.6	121.9%	1.7% / 24.4% / 73.9%
		8.0	55.2	64.0	32.5	37.0	40.2	45.8	122.4%	0.0% / 24.6% / 75.4%
MISTRAL-7B-INST-V0.3	No Adaptation	-	37.6	33.5	28.0	15.0	23.8	27.6	–	100% / – / –
	RAG	-	30.2	28.1	35.0	16.0	25.8	27.0	97.9%	– / 100% / –
	TTT	-	38.4	38.4	32.0	16.5	24.0	29.9	108.2%	– / – / 100%
	RTTC	2.0	32.4	37.2	32.0	21.0	23.8	29.3	106.1%	23.2% / 24.9% / 51.9%
		5.0	32.4	37.8	34.5	21.0	23.2	29.8	107.9%	4.6% / 32.2% / 63.2%
		8.0	32.8	37.8	37.5	19.5	23.3	30.2	109.4%	1.4% / 33.1% / 65.5%
	RTTC-Joint	2.0	33.4	36.6	32.5	22.5	24.3	29.9	108.2%	23.2% / 29.4% / 47.4%
		5.0	35.2	36.0	31.5	24.0	24.3	30.2	109.5%	4.6% / 35.0% / 60.4%
		8.0	33.8	36.6	39.0	20.5	22.7	30.5	110.6%	1.4% / 32.9% / 65.8%
QWEN2.5-3B-INST	No Adaptation	-	41.2	26.2	26.0	32.5	26.5	30.5	–	100% / – / –
	RAG	-	38.8	41.5	32.5	42.0	26.3	36.2	118.8%	– / 100% / –
	TTT	-	42.4	25.6	30.5	37.0	26.3	32.4	106.2%	– / – / 100%
	RTTC	2.0	47.4	43.3	28.5	42.0	26.5	37.5	123.1%	42.8% / 18.1% / 39.1%
		5.0	46.6	42.7	31.5	43.5	27.2	38.3	125.6%	12.1% / 24.0% / 63.9%
		8.0	46.0	39.0	31.0	50.0	25.0	38.2	125.3%	4.5% / 25.4% / 70.1%
	RTTC-Joint	2.0	48.0	42.7	27.5	43.5	26.2	37.6	123.2%	42.8% / 15.7% / 41.5%
		5.0	46.4	45.1	31.0	49.5	26.0	39.6	129.9%	12.1% / 21.6% / 66.4%
		8.0	46.4	43.9	32.5	51.5	26.2	40.1	131.5%	4.5% / 24.7% / 70.8%

Table 12: **Impact of Reward Threshold in RTTC.** Performance comparison of different adaptation strategies under varying reward thresholds. The "Threshold (τ_r)" column denotes the value of the reward model's first-stage evaluation parameter: a higher threshold enforces stricter quality requirements for the initial response, increasing the likelihood of triggering RAG or TTT adaptation. Larger thresholds generally yield higher accuracy, but also incur greater computational cost due to more frequent execution of advanced adaptation strategies. "Impr." denotes the relative improvement over the No Adaptation baseline. "Strategy Distribution (%)" reports the proportion of queries handled by each branch in the RTTC pipeline: No Adaptation, RAG, and TTT, respectively.

Model	Strategy	Threshold (τ_r)	Reward Model Size	MBPP	HumanEval	MathQA*	GSM-Plus*	ATC*	Avg.	Impr.	Strategy Distribution (%)
LLAMA-3-8B-INST	No Adaptation	-	-	51.6	54.9	29.0	19.5	36.8	38.4	–	100% / – / –
	RAG	-	-	42.0	48.8	39.5	20.5	41.8	38.5	100.4%	– / 100% / –
	TTT	-	-	49.0	54.9	37.0	29.5	36.7	41.4	108.0%	– / – / 100%
	RTTC	5.0	0.6B	53.4	56.1	39.5	22.0	37.3	41.7	108.6%	1.2% / 26.9% / 71.9%
			8B	53.6	54.9	34.0	28.5	38.8	42.0	109.4%	38.5% / 25.4% / 36.1%
		8.0	0.6B	53.4	56.1	39.5	22.0	36.8	41.6	108.4%	0.1% / 27.0% / 73.0%
			8B	53.4	55.5	33.5	29.5	37.5	41.9	109.2%	19.6% / 28.2% / 52.2%
	RTTC-Joint	5.0	0.6B	54.6	56.7	42.0	20.5	37.7	42.3	110.3%	1.2% / 28.2% / 70.6%
			8B	52.0	57.3	34.0	31.0	37.7	42.4	110.5%	38.5% / 22.5% / 39.0%
		8.0	0.6B	54.6	56.7	42.0	20.5	38.2	42.4	110.5%	0.1% / 28.4% / 71.6%
			8B	53.8	56.1	32.5	30.5	38.0	42.2	110.0%	19.6% / 28.5% / 51.9%
LLAMA-3.1-8B-INST	No Adaptation	-	-	52.2	59.8	16.5	20.0	38.5	37.4	–	100% / – / –
	RAG	-	-	42.4	51.8	32.5	33.0	44.3	40.8	109.2%	– / 100% / –
	TTT	-	-	51.6	63.4	25.0	34.0	37.8	42.4	113.3%	– / – / 100%
	RTTC	5.0	0.6B	53.6	62.2	30.5	30.0	38.8	43.0	115.1%	1.7% / 23.3% / 75.0%
			8B	53.6	59.8	21.5	31.0	41.2	41.4	110.7%	45.1% / 19.0% / 35.9%
		8.0	0.6B	53.6	62.2	30.5	29.5	38.8	42.9	114.8%	0.0% / 23.3% / 76.7%
			8B	53.8	59.8	27.5	31.0	39.3	42.3	113.1%	27.3% / 18.2% / 54.6%
	RTTC-Joint	5.0	0.6B	55.2	63.4	32.5	37.0	39.8	45.6	121.9%	1.7% / 24.4% / 73.9%
			8B	53.8	59.8	24.5	36.5	40.0	42.9	114.8%	45.1% / 13.5% / 41.4%
		8.0	0.6B	55.2	64.0	32.5	37.0	40.2	45.8	122.4%	0.0% / 24.6% / 75.4%
			8B	53.8	60.4	27.5	37.5	40.2	43.9	117.3%	27.3% / 16.2% / 56.6%
MISTRAL-7B-INST-V0.3	No Adaptation	-	-	37.6	33.5	28.0	15.0	23.8	27.6	–	100% / – / –
	RAG	-	-	30.2	28.1	35.0	16.0	25.8	27.0	97.9%	– / 100% / –
	TTT	-	-	38.4	38.4	32.0	16.5	24.0	29.9	108.2%	– / – / 100%
	RTTC	5.0	0.6B	32.4	37.8	34.5	21.0	23.2	29.8	107.9%	4.6% / 32.2% / 63.2%
			8B	31.4	34.8	32.0	22.5	28.0	29.7	107.8%	38.1% / 31.5% / 30.4%
		8.0	0.6B	32.8	37.8	37.5	19.5	23.3	30.2	109.4%	1.4% / 33.1% / 65.5%
			8B	31.0	36.0	33.5	20.5	24.8	29.2	105.7%	22.4% / 34.0% / 43.6%
	RTTC-Joint	5.0	0.6B	35.2	36.0	31.5	24.0	24.3	30.2	109.5%	4.6% / 35.0% / 60.4%
			8B	31.4	36.6	34.0	21.0	27.0	30.0	108.7%	38.1% / 28.9% / 33.0%
		8.0	0.6B	33.8	36.6	39.0	20.5	22.7	30.5	110.6%	1.4% / 32.9% / 65.8%
			8B	31.4	36.6	35.0	21.0	26.2	30.0	108.8%	22.4% / 34.2% / 43.4%
QWEN2.5-3B-INST	No Adaptation	-	-	41.2	26.2	26.0	32.5	26.5	30.5	–	100% / – / –
	RAG	-	-	38.8	41.5	32.5	42.0	26.3	36.2	118.8%	– / 100% / –
	TTT	-	-	42.4	25.6	30.5	37.0	26.3	32.4	106.2%	– / – / 100%
	RTTC	5.0	0.6B	46.6	42.7	31.5	43.5	27.2	38.3	125.6%	12.1% / 24.0% / 63.9%
			8B	48.6	45.1	27.0	41.5	27.0	37.8	124.1%	52.7% / 20.6% / 26.7%
		8.0	0.6B	46.0	39.0	31.0	50.0	25.0	38.2	125.3%	4.5% / 25.4% / 70.1%
			8B	48.6	42.7	28.5	44.0	27.7	38.3	125.6%	35.8% / 24.0% / 40.2%
	RTTC-Joint	5.0	0.6B	46.4	45.1	31.0	49.5	26.0	39.6	129.9%	12.1% / 21.6% / 66.4%
			8B	46.8	46.3	27.5	45.5	26.0	38.4	126.1%	52.7% / 16.7% / 30.7%
		8.0	0.6B	46.4	43.9	32.5	51.5	26.2	40.1	131.5%	4.5% / 24.7% / 70.8%
			8B	49.6	43.9	29.0	45.0	27.2	38.9	127.7%	35.8% / 22.4% / 41.8%

Table 13: **Impact of Reward Model in RTTC.** Performance comparison of adaptation strategies under varying reward models. "0.6B" refers to the SKYWORK-REWARD-V2-QWEN3-0.6B, and "8B" refers to the SKYWORK-REWARD-V2-LLAMA-3.1-8B. The "Threshold (τ_r)" column denotes the minimum reward score required for direct response acceptance. "Impr." reports relative improvement over the No Adaptation baseline. "Strategy Distribution (%)" indicates the proportion of queries handled by No Adaptation, RAG, and TTT branches, respectively.