CT3: Boosting Downstream Performance through Test-time Training on AI PCs with Remote Multi-Domain Knowledge Bases

Anonymous ACL submission

Abstract

Learning at test time is an effective strategy for improving the performance of large language models (LLMs), although at the expense of increased computational costs during inference. In this paper, we introduce **C**ollaborative **T**est-**T**ime **T**raining (**CT3**), a novel system designed to enhance the downstream accuracy of LLMs on client devices such as AI PCs through Test-Time Training (TTT) leveraging a remote multi-domain knowledge base. CT3 efficiently distributes the TTT process using a server-client architecture, allowing clients to fine-tune their models using relevant samples from the server. It also proposes a local state management mechanism and a simple but effective sample size reduction strategy to optimize test-time training without compromising accuracy. Our experiments demonstrate significant accuracy improvements across multiple domains and various LLMs with up to 44% increase in average downstream performance and a speedup ranging from 1.5× to 2.5× compared with vanilla TTT. The code to reproduce CT3's results will be released open-source.

1 Introduction

There is an increasing interest in techniques for enhancing model performance by using additional computational resources at test time rather than scaling model parameters during training (Snell et al., 2025). Test-time training (TTT) is one such approach that leverages test-time compute resources to fine-tune the model during inference. This paradigm improves the robustness and accuracy of large pre-trained models, addressing the key challenge of *covariate shift*, wheretest and training data distributions differ, potentially causing suboptimal performance. TTT enhances language modeling performance by accessing a reference dataset (Hardt and Sun, 2024; Hübotter et al., 2024), from which a subset of data points is retrieved and used for temporary model fine-tuning.

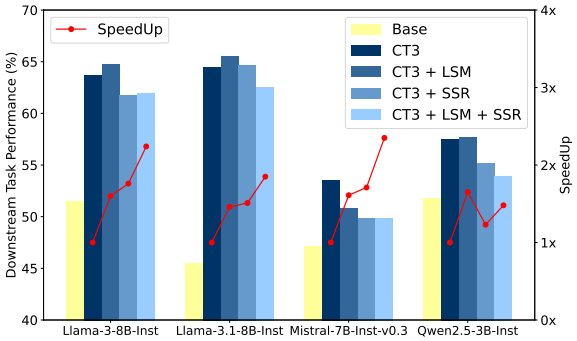

Figure 1: Performance overview of CT3 on downstream tasks across various LLMs. "Downstream Task Performance" indicates the average score on six downstream tasks with multiple domains evaluated in §4.

When the size of the TTT reference dataset reaches several petabytes, using these techniques on resource-constrained devices at the Edge becomes challenging due to their limited resources and the need for real-time fine-tuning, which demands significant computational power and storage capacity. Our work is motivated by these challenges and a new segment of client computing devices, namely Artificial Intelligence Personal Computers (AI PCs), which are equipped with AI accelerators such as Neural Processing Units (NPUs) and exhibit outstanding power efficiency. These AI PCs efficiently run sophisticated language models locally, such as LLAMA-3-8B-INSTRUCT (Dubey et al., 2024). The increasing demand for AI PCs motivates the development of solutions to improve model performance on these devices.

This paper addresses these challenges by proposing **C**ollaborative **TTT** (CT3), a solution that enables client devices to benefit from TTT without storing large reference datasets locally. Specifically, our approach leverages an efficient client-server solution to allow the limited device to benefit from the knowledge stored on a remote server while minimizing the communication rounds. To

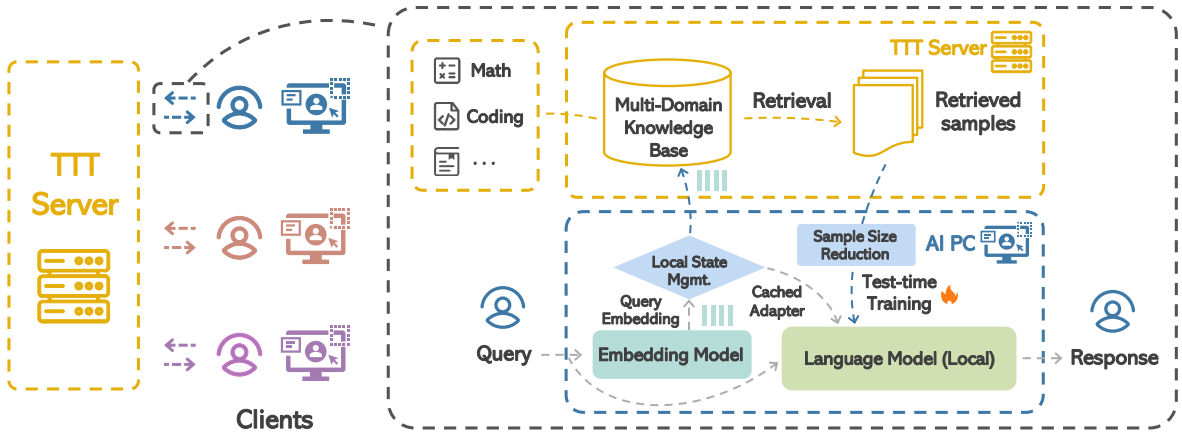

Figure 2: Overview of Collaborative TTT (CT3). The performance of models at client devices improves by utilizing Test-Time Training with a remote multi-domain knowledge base.

enhance the setup's sophistication, we explore using a multi-domain knowledge base and investigate the tolerance of TTT algorithms to complex data mixes. CT3 allows users to run custom models in their resource-constrained devices with TTT, overcoming limitations such as training data availability while benefiting from downstream task performance improvements. CT3 introduces Local State Management (LSM) and Sample Size Reduction (SSR) strategies to optimize the test-time fine-tuning process on client devices. LSM leverages historical query embeddings and their associated fine-tuned adapters to potentially bypass the need for repeated fine-tuning, thus significantly reducing computational overhead and latency. SSR strategies complement LSM by filtering out irrelevant samples, ensuring that only the most relevant samples are used, thereby reducing the number of training samples and accelerating the process without compromising accuracy. Figure 1 provides a performance overview of CT3. The following sections discuss these contributions:

1. CT3, a system for improving the downstream reasoning capabilities of LLMs running in client devices (Figure 2).
2. An investigation into using multi-domain knowledge bases with clear supervisory signals in the context of test-time training for downstream tasks.
3. Local State Management (LSM) and Sample Size Reduction (SSR) strategies to optimize the test-time fine-tuning process, reducing computational overhead and improving system responsiveness.

The rest of the paper is organized as follows. We discuss related work in §2. Then, §3 describes the CT3 system, while §4 presents results on various downstream tasks. Our final thoughts are in §5.

2 Related Work

Test-time compute techniques have been proposed as an alternative to increasing the number of model parameters for improving language model performance (Snell et al., 2025). Notable test-time 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, e.g., by code execution (Brown et al., 2025). The renewed focus on test-time compute has even motivated the development of improved neural architectures, e.g., Titans (Behrouz et al., 2024).

Among the many recent test-time compute approaches proposed, *Test-Time Training (TTT)* (Hardt and Sun, 2024; Hübotter et al., 2024; Akyürek et al., 2024) has been crucial in improving the performance of solutions, e.g., Omni-ARC (IronbarArc24, 2024) in the ARC-AGI challenge (ARC-AGI, 2025; Chollet, 2019).

TTT effectively adapts the model by fine-tuning it with selected samples from an available training dataset during inference. Recently, SIFT (Hübotter et al., 2024) demonstrated that it could retrieve informative samples, outperforming traditional Nearest-Neighbor (NN) retrieval (Hardt and Sun, 2024). Their experimental setup uses the

Pile dataset (Gao et al., 2020), which lacks a clear question-answer pair structure despite covering a wide range of topics and fields with plain text data. This weak supervision signal makes the training process more challenging, focusing on the model's language modeling capabilities (Pile benchmark (Gao et al., 2020)). In contrast, CT3 prioritizes downstream task performance. Our knowledge base comprises structured question-answer pairs with clear supervisory signals, which helps the model explicitly learn the relationship between input and output during training.

In the context of distributed architectures, the work by Hardt and Sun (2024) proposes a client-server structure to speed up query time across large datasets. Their approach splits FAISS (Douze et al., 2024) indexes across multiple servers, allowing clients to send queries to each server and aggregate the results. This method primarily focuses on accelerating query processing and handling large-scale data efficiently. In contrast, our CT3 leverages a server-client architecture to facilitate TTT using a remote multi-domain knowledge base. While both approaches utilize distributed architectures, our system is designed to enhance model performance on resource-constrained client devices by retrieving relevant samples for fine-tuning during inference. Their architecture (Hardt and Sun, 2024) can be integrated into our remote server to further optimize sample retrieval, highlighting the potential of combining between these approaches.

Next, we explore CT3, a system that enhances model downstream performance on client devices.

3 CT3 System

This section introduces Collaborative Test-Time Training (CT3), a system designed to distribute Test-Time Training to allow resource-constrained clients to enhance their model downstream performance using a remote multi-domain knowledge base. CT3 also incorporates Local State Management (LSM) and Sample Size Reduction (SSR) strategies to speed up the fine-tuning stage without sacrificing accuracy. The following sections detail each of CT3's components and strategies.

3.1 Preliminaries: Test-Time Training (TTT)

Given a pre-trained model and a query x, a TTT solution utilizes a similarity metric ϕ to retrieve k samples from a reference dataset $\mathcal{D}$. These samples are then used to fine-tune the model weights at test time, resulting in improved model performance for the current query. Next, we discuss how CT3 leverages the TTT paradigm in a distributed setting to improve model performance in resource-constrained client devices.

3.2 Distributed Test-time Training

CT3 operates efficiently in heterogeneous compute environments (Figure 2), following a server-client paradigm implemented using FastAPI (Ramírez, 2025). In this section, we discuss the functionality of CT3 at both the server and client levels.

Server The remote server hosts a comprehensive multi-domain knowledge base $\mathcal{D}$, which encompasses training samples across commonsense reasoning, reading comprehension, coding, and math reasoning domains. The data is stored along with their corresponding embeddings, which are generated using a pre-trained model from sentence-transformers (Reimers and Gurevych, 2019). The dataset can be represented as $\mathcal{D} = \{(x_i, \mathbf{e}_i, y_i)\}_{i=1}^{d}$, where x_i denotes the i-th input sentence, $\mathbf{e}_i$ represents its embedding, and y_i is the associated label or output.

The server receives client queries as embeddings. To ensure consistency, servers and clients are required to use the same sentence-transformers to generate embeddings. A private embedding model and encryption can increase privacy protection in data-sensitive applications. Upon receiving a client query embedding $\mathbf{e}_x$, following SIFT (Hübotter et al., 2024), the server utilizes FAISS (Douze et al., 2024) to retrieve n relevant samples $\mathcal{D}_{\mathcal{F}} = \{(x_i, \mathbf{e}_i, y_i)\}_{i=1}^{n}$ from the knowledge base. The similarity metric ϕ employed by FAISS is the inner product, i.e., $\phi(\mathbf{e}_x, \mathbf{e}_i) = \mathbf{e}_x \cdot \mathbf{e}_i$, calculated between the embedding of the current query $\mathbf{e}_x$ and the embeddings $\mathbf{e}_i$ of candidate samples from the dataset $\mathcal{D}$. CT3 might use other search strategies to yield better performance based on the multi-domain data characteristics and mix.

To obtain more accurate and relevant samples, the SIFT algorithm (Hübotter et al., 2024) is applied to further refine the results retrieved by FAISS. SIFT filters the samples retrieved by FAISS, resulting in a smaller set, $\mathcal{D}_{\mathcal{S}} = \{(x_i, \mathbf{e}_i, y_i)\}_{i=1}^{k}$ ($k \ll n$), that is returned to the client without the embedding $\mathbf{e}_i$ for efficient transmission.

The server can extract and transmit the corresponding samples to clients based on the above process. The client utilizes these samples to fine-tune its custom model, thereby increasing the accuracy and quality of the output.

Clients On a client device, the workflow is as follows: a deployed large language model receives a query x from the user. The client locally generates an embedding of the query $\mathbf{e_x}$, using the same sentence-transformers model as the server. This embedding is then transmitted to the remote server. As discussed above, the server sends back a set $\mathcal{D}_S$ with k relevant samples to the query, excluding their embeddings. Using $\mathcal{D}_S$, the client temporarily fine-tunes its custom model via LoRA (Hu et al., 2022) for efficiency, resulting in improved performance during inference. Finally, nothing prevents a client with resources that satisfy the requirements of smaller deployments to run the complete CT3 pipeline on a single device.

In-Domain Sample Selection CT3 does not know the domain of the user's query in advance. During development, we explored two approaches for assisting the system when searching for in-domain samples:

1. User guidance to determine the domain
2. Automatic domain clustering purely based on embeddings

Although user guidance (1) is a viable strategy, our initial objective was to reduce user friction and minimize user actions to improve the system performance. In the automatic domain clustering approach (2), domain specialization occurs during the online search of relevant samples for test-time fine-tuning, allowing CT3 to provide samples tailored to the domain-specific query from the user. Experimentally (§4.3), we have observed that automatic domain clustering is feasible but with several challenges. For instance, the knowledge base must have the correct data mix, which presents an additional challenge to system administrators.

Local State Management (LSM) The fine-tuning stage at the client is the most expensive operation in the CT3 pipeline. For this reason, we propose caching strategies at the sampling and model state levels. A multi-turn historical Local State Management (LSM) strategy is described in Algorithm 1. Based on the availability of resources at the client, CT3 can budget the tracking of m past query embeddings $\mathcal{Q}$ and their associated groups of already fine-tuned LoRA adapters $\mathcal{A}$. Each new user query embedding is compared with historical query embeddings to identify the one that maximizes a similarity function γ, indicating the possibility of using the group of associated weight adapters and skipping the TTT stage to speed up inference without affecting accuracy. A simple version of Algorithm 1, is the one-look-back strategy in which $m = 1$ and CT3 forgets beyond the immediate previous query.

Algorithm 1 Multi-Turn Historical Local State Management (LSM)

Input: Current query embedding e_{x^t}, set $\mathcal{Q}$ of m historical query embeddings, i.e., $\mathcal{Q} = \{e_{x^1}, ..., e_{x^m}\}$, set $\mathcal{A}$ of m historical TTT fine-tuned adapter groups, i.e., $\mathcal{A} = \{a_{x^1}, ..., a_{x^m}\}$, reuse threshold τ, budget B for the number of adapter groups kept in the client device, similarity metric γ, and an eviction mechanism κ to determine the historical query to be evicted.
Output: Current adapter group state S_t.

$e_{x^*} = \underset{e_{x^i} \in \mathcal{Q}}{\operatorname{argmin}}\, \gamma(e_{x^i}, e_{x^t})$
if $\gamma(e_{x^*}, e_{x^t}) > \tau$ **then**
  // Retrieve corresponding adapter group
  $S_t \leftarrow a_{x^*}$
else
  $S_t \leftarrow \mathrm{TTT}(e_{x^t}, S_0)$
  if $|\mathcal{Q}| \geq B$ **then**
    // Evict one historical query using κ
    $e_{x^r} = \kappa(\mathcal{Q})$
    $\mathcal{Q} = \mathcal{Q} \setminus e_{x^r}$
    $\mathcal{A} = \mathcal{A} \setminus a_{x^r}$
  end if
  // Add new query and adapter to LSM
  $\mathcal{Q} \cup e_{x^t}$
  $\mathcal{A} \cup S_t$
end if
// Inference using S_t

Sample Size Reduction (SSR) We also explore simple but effective sample size reduction strategies to reduce further the samples selected by SIFT. The strategies follow this pattern: CT3 obtains statistics from the multiset of acquisition values, $\mathcal{V}$ ($|\mathcal{V}| = k$), associated with the k samples selected by SIFT. In particular, CT3 explores using the minimum of the mean and the median on $\mathcal{V}$ as the cut-off point for discarding selected samples. Based on these statistics, CT3 selects a subset, $\mathcal{V}'$, such that $|\mathcal{V}'| < |\mathcal{V}|$.

4 Experiments

We implement a prototype of the CT3 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 test-time training in client devices.

4.1 Setup

Knowledge base To rigorously evaluate the CT3 prototype, we utilize a comprehensive multi-domain knowledge database. This knowledge base

Dataset	Domain	# Samples
CoQA (Reddy et al., 2019)	Reading	7,199
MetaMath (Yu et al., 2023)	Math	395,000
Orca-Math (Mitra et al., 2024)	Math	200,035
Math 50K (Zhiqiang et al., 2023)	Math	50,000
The Stack Python (Kocetkov et al., 2022)	Coding	600,000
MBPP (Austin et al., 2021)	Coding	374
Total	/	1,252,608

Table 1: Knowledge base composition. These datasets cover various domains, including reading comprehension, math, and coding. The Stack Python dataset consists of a random sample of 600,000 entries from the original Stack dataset (Python). All these datasets are training sets and do not contain any of the test samples we evaluated.

is constructed by integrating data from several diverse datasets (Table 1). Specifically, we incorporate the CoQA (Reddy et al., 2019) training set, a large-scale dataset designed for developing conversational question-answering systems, which is expected to enhance the model's reading comprehension capabilities.

For mathematical problem solving, we include three substantial datasets: MetaMath (Yu et al., 2023), a dataset comprising 395,000 samples designed to improve mathematical reasoning by bootstrapping questions from multiple perspectives; Orca-Math (Mitra et al., 2024), which includes 200,035 high-quality synthetic math problems created using a multi-agent setup, and Math 50K (Zhiqiang et al., 2023), which consists of 50,000 samples drawn from various mathematics-related datasets.

In the coding domain, we integrate The Stack Python, a dataset we constructed by randomly sampling 600,000 entries from the Python subset of The Stack (Kocetkov et al., 2022), which covers a wide range of programming languages and serves as a pre-training dataset for code-generating AI systems. Additionally, we include the Mostly Basic Python Problems (MBPP) (Austin et al., 2021) training dataset, which consists of 374 crowdsourced Python programming problems aimed at entry-level programmers.

This amalgamation of datasets ensures a robust and diverse foundation for evaluating the CT3 system's performance across multiple domains, including Reading Comprehension, Math, and Coding.

Evaluation We employ a combination of evaluation tools for our experiments to rigorously assess CT3 prototype's performance across various domains. Specifically, we utilize the *lm-eval-harness* (Gao et al., 2023) for evaluating CoQA (Reddy et al., 2019) and GSM8K (Cobbe et al., 2021). For GSM8K, to better align with real-world TTT scenarios, we use zero-shot instead of the commonly used few-shot approach. For more mathematical reasoning evaluations, we also apply the evaluation scripts of LLM-Adapters (Zhiqiang et al., 2023) on MathQA (Amini et al., 2019) and MAWPS (Koncel-Kedziorski et al., 2016) datasets. For coding, we evaluate MBPP (Austin et al., 2021) and HumanEval (Chen et al., 2021) utilizing *bigcode-evaluation-harness* (Ben Allal et al., 2022). For experimental efficiency, we randomly sampled a subset of 200 samples from GSM8K and MathQA for evaluation, respectively.

Models We test our method on various LLMs, including LLAMA-3-8B-INSTRUCT, LLAMA-3.1-8B-INSTRUCT (Dubey et al., 2024), MISTRAL-7B-INSTRUCT-V0.3 (Jiang et al., 2023) and QWEN2.5-3B-INSTRUCT (Yang et al., 2024). For the retrieval stage in CT3, we utilize ALL-MPNET-BASE-V2 (Reimers and Gurevych, 2019) as a surrogate embedding model.

Hyperparameters and Implementation We employ two epochs for test-time training with a learning rate of 5e-5. We adjust the learning rate for coding tasks to either 1e-5 or 3e-5. The batch size is set to 1 across all tasks. We apply LoRA fine-tuning at test time, with a LoRA rank of 128 and a LoRA alpha of 16. The target modules for LoRA include the Query, Key, and Value and the Up and Down projection layers. Note that greedy decoding is applied to generate the responses for all tasks. The number of samples for test-time training is selected from the hyperparameter space [4, 8, 16, 32, 64] based on the best results. For LSM, the reuse threshold τ and k in LSM is 0.5 and 8, respectively. The similarity metric is the inner product, and the eviction mechanism is the Least Frequently Used (LFU) cache. Regarding the retrieval stage, we follow the pipeline of SIFT (Hübotter et al., 2024). More explorations can be found in §4.4 and Appendix C.

4.2 Main Results

Table 2 presents the performance of various large language models on multiple evaluation tasks, com-

Method	CoQA EM	CoQA F1	GSM8K* EM (0-shot)	MathQA* Acc.	MAWPS Acc.	MBPP Pass@1	HumanEval Pass@1	Avg.	Impr.	Evaluation Speedup
LLAMA-3-8B-INSTRUCT	61.78	78.40	40.00	30.00	43.70	51.60	54.88	51.48	/	/
+ CT3	70.38	82.25	**60.50**	36.00	86.13	**52.40**	57.93	63.66	+24%	1.00×
+ CT3 + SSR	**70.68**	**82.46**	55.00	28.50	85.71	**52.40**	57.32	61.72	+20%	1.76×
+ CT3 + LSM	70.17	82.24	59.50	**41.50**	**89.08**	51.60	**59.15**	**64.75**	**+26%**	1.60×
+ CT3 + LSM + SSR	70.20	82.17	56.00	33.00	84.45	51.80	56.10	61.96	+20%	**2.24**×
LLAMA-3.1-8B-INSTRUCT	63.63	78.82	23.00	15.50	25.21	52.20	59.76	45.45	/	/
+ CT3	71.12	83.48	**70.00**	21.50	89.50	**53.60**	**62.20**	64.48	+42%	1.00×
+ CT3 + SSR	70.52	82.49	67.00	**28.00**	**90.76**	52.00	61.59	64.62	+42%	1.51×
+ CT3 + LSM	**72.23**	**83.95**	**70.00**	**28.00**	**90.76**	52.20	61.59	**65.53**	**+44%**	1.46×
+ CT3 + LSM + SSR	71.02	82.78	61.00	22.50	87.82	51.40	60.98	62.50	+38%	**1.85**×
MISTRAL-7B-INSTRUCT-V0.3	65.55	80.04	17.00	28.00	68.07	37.60	33.54	47.11	/	/
+ CT3	**71.32**	**83.08**	**24.50**	32.50	**86.97**	**38.80**	37.20	**53.48**	**+14%**	1.00×
+ CT3 + SSR	70.85	82.65	14.50	29.00	78.99	38.00	34.76	49.82	+6%	1.71×
+ CT3 + LSM	70.58	82.62	13.50	29.50	83.61	37.80	**38.41**	50.86	+8%	1.61×
+ CT3 + LSM + SSR	69.80	82.14	10.50	**34.00**	78.15	38.60	35.98	49.88	+6%	**2.35**×
QWEN2.5-3B-INSTRUCT	54.70	71.22	61.50	20.00	87.82	41.20	26.22	51.81	/	/
+ CT3	67.27	80.53	61.50	28.00	88.24	**46.80**	**29.88**	57.46	+11%	1.00×
+ CT3 + SSR	66.22	79.77	63.50	**33.50**	70.17	45.80	27.44	55.20	+7%	1.23×
+ CT3 + LSM	**67.43**	**80.59**	63.50	32.50	**88.66**	44.40	26.83	**57.70**	**+11%**	**1.65**×
+ CT3 + LSM + SSR	67.15	80.56	**65.00**	31.00	61.34	43.40	29.27	53.96	+4%	1.48×

Table 2: Performance comparison of different versions of CT3. "SSR" stands for Sample Size Reduction, which approximately halves the TTT training samples for each query. "Evaluation Speedup" shows the speedup factor achieved by our proposed strategies. "EM" represents Exact Match. "Impr." indicates the improvement in the average score. Note that * for GSM8K and MathQA indicates a subset of 200 randomly sampled test samples.

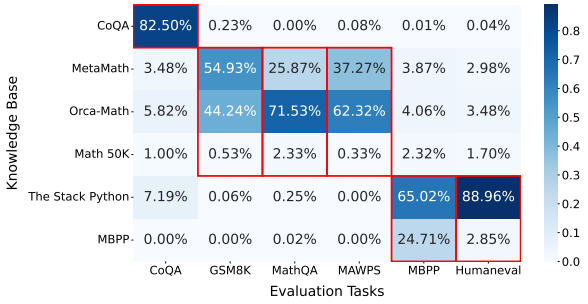

Figure 3: Domain distribution of training data samples across evaluation tasks (counted all test samples). The red border indicates the domain corresponding to the current task (in-domain). We expect higher values within the red border, meaning that the test-time training samples from the knowledge base are more aligned with the domain of the current task.

paring the baseline models without CT3, with CT3, and CT3 incorporating the LSM and SSR strategies. The results demonstrate that CT3 significantly enhances the performance across all downstream tasks and models. For example, regarding LLAMA-3-8B-INSTRUCT, CT3 improves the average performance from 51.48 to 63.66, representing a 24% improvement. When combined with LSM, CT3 achieves an average score of 64.75, a 26% improvement, and with both LSM and SSR, it achieves a 20% improvement with a 2.24× evaluation speedup compared to CT3. Similarly, LLAMA-3.1-8B-INSTRUCT shows a remarkable 42% improvement with CT3 and achieves higher performance with LSM, achieving up to 1.46× speedup. The results indicate that CT3 significantly improves model performance across various tasks and domains. The LSM and SSR strategies effectively maintain high performance while substantially reducing the TTT computational overhead, making CT3 more feasible for resource-constrained devices.

4.3 In-Domain Sample Retrieval

To evaluate the effectiveness of CT3 in retrieving relevant in-domain samples from multi-domain data, we conducted experiments to analyze the domain distribution of the retrieved training samples across various evaluation tasks. Figure 3 illustrates

the distribution of TTT data samples retrieved for each evaluation task, categorized by their respective domains.

The heatmap reveals that most retrieved samples align well with the domain of the current task, as indicated by higher values within the red-bordered cells. For instance, 82.50% of the samples retrieved for the CoQA task are from the CoQA dataset, demonstrating strong in-domain alignment. This is particularly noteworthy given that the CoQA dataset constitutes only a small fraction of the knowledge base (7,199 out of 1,252,608 samples). Despite the overwhelming presence of unrelated data, CT3's retrieval mechanism can precisely identify and select relevant CoQA samples for the CoQA task. Similarly, for GSM8K, 54.93% of the samples are from MetaMath, and for MathQA, 71.53% are from Orca-Math, indicating that CT3 successfully identifies and selects relevant mathematical datasets for these tasks.

These alignments are crucial for the success of test-time training, as it ensures that the model is fine-tuned with high-quality, relevant samples, thereby significantly enhancing performance across various tasks. The effectiveness of CT3 in achieving such precise retrieval is attributed to the robust embedding models (Reimers and Gurevych, 2019) and retrieval algorithms employed. These tools enable CT3 to navigate a vast and diverse knowledge base, accurately matching queries to the most relevant samples. Overall, the results highlight the importance of retrieving high-quality, relevant samples for test-time training, allowing CT3 to enhance model performance significantly by leveraging domain-specific data effectively.

Additionally, we performed a t-SNE visualization of the embeddings for some samples in the database, as shown in Figure 4. The visualization demonstrates a clear clustering of samples by domain that CT3 can exploit to retrieve domain-specific samples. For example, the coding datasets (The Stack Python and MBPP) exhibit clear clustering, validating the embedding model's capability to differentiate between domains. This clustering effect underscores the robustness of the embedding model and the retrieval algorithm, enabling CT3 to achieve high precision in sample retrieval. The t-SNE visualization highlights the importance of high-quality embeddings in facilitating accurate and efficient sample retrieval, making test-time training more practical and impactful.

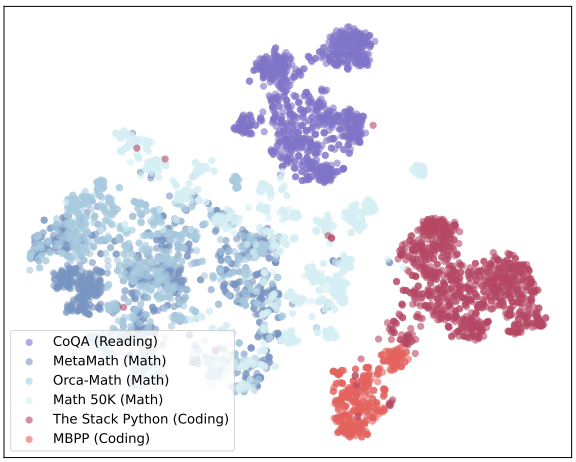

Figure 4: t-SNE visualization of the embeddings for some samples in the knowledge base. Each dataset contains 1000 randomly selected samples, except for MBPP (all).

Faiss Indexes	Peak CPU Memory Usage	Avg. Score
/	/	51.48
FlatIP	11568 MB	**63.66**
IVFPQ	**7936 MB**	63.47

Table 3: Comparison of different Faiss indexes regarding Peak CPU Memory Usage and Avg. Score for LLAMA-3-8B-INSTRUCT. The Avg. Score represents the average performance across six downstream tasks, while the peak CPU memory usage was measured using 16 test samples from the GSM8K dataset.

4.4 Memory-Efficient Indexing Alternatives for CT3 Setup

In the CT3 prototype, we use *flat inner product (FlatIP)* indexes for retrieval in FAISS. A more memory-efficient indexing can be used if the user desires to run the complete CT3 system in an AI PC. As described in Table 3, using *inverted file product quantization (IVFPQ)* indexes reduces the peak CPU memory by 31% without significantly impacting the average score.

4.5 Exploring Smaller Knowledge Bases

We created a subset of the original knowledge base to demonstrate the effectiveness of a large knowledge base and the efficiency and feasibility of deploying smaller knowledge bases on local devices. This smaller knowledge base includes only the CoQA, Math 50K, and MBPP datasets, reducing the size from 1,252,608 samples to 57,573 samples. Table 4 presents the performance of CT3 using both the full and reduced knowledge bases.

The results show that while the smaller knowl-

Model	Knowledge Size	CoQA EM	CoQA F1	GSM8K* EM (0-shot)	MathQA* Acc.	MAWPS Acc.	MBPP Pass@1	HumanEval Pass@1	Avg.
LLAMA-3-8B-INSTRUCT	/	61.78	78.40	40.00	30.00	43.70	51.60	54.88	51.48
	57.57K	71.90	82.99	59.00	**34.50**	75.63	51.80	57.93	61.96
	1.25M	**72.18**	**83.30**	**63.50**	32.00	**86.97**	**52.40**	**59.76**	**64.30**
LLAMA-3.1-8B-INSTRUCT	/	63.63	78.82	23.00	15.50	25.21	52.20	59.76	45.45
	57.57K	**72.57**	83.35	62.50	22.00	84.03	52.60	**62.80**	62.83
	1.25M	71.95	**83.82**	**70.00**	**28.00**	**88.66**	**53.00**	**62.80**	**65.46**

Table 4: Performance comparison of CT3 using a knowledge base (1.25M samples, Table 1) versus a reduced knowledge base (57.57K samples). The reduced knowledge base includes only CoQA, Math 50K, and MBPP datasets. * for GSM8K and MathQA indicates a subset of 200 randomly sampled test samples.

edge base (57.57K) improves performance over the baseline models without CT3, the larger knowledge base (1.25M) consistently yields better results. For instance, with LLAMA-3-8B-INSTRUCT, the average performance increases from 51.48 (baseline) to 61.96 with the smaller knowledge base and to 64.30 with the larger knowledge base. This indicates that a larger and more diverse knowledge base provides more relevant samples for test-time training, leading to superior performance.

These findings highlight the trade-off between knowledge base size and performance. While a smaller knowledge base is more efficient and feasible for local deployment, a larger knowledge base offers significant performance gains.

4.6 Performance with Domain-Specific Databases

To further investigate the impact of retrieving samples exclusively from the correct domain, we performed experiments where the query samples were extracted only from the corresponding domain-specific dataset. Table 5 presents the performance comparison of CT3 using a mixed-domain database versus an in-domain database for CT3. The results indicate that the in-domain scenario performs comparably or slightly better than the multi-domain scenario, suggesting that while domain-specific retrieval can enhance performance, the multi-domain setup proposed in this paper is feasible and effective. In real-world applications, incorporating a domain classifier or user-guided domain selection could further optimize the retrieval process, ensuring that the most relevant samples are used for test-time training, thus maximizing model performance.

Task & Metric	Baseline	Domain	CT3	CT3 + SSR
CoQA EM	61.78	Mix	**71.90**	69.85
		Reading	71.63	**70.05**
CoQA F1	78.40	Mix	**82.99**	81.97
		Reading	82.80	**82.19**
GSM8K* EM (0-shot)	40.00	Mix	59.00	54.50
		Math	**61.00**	**61.00**
MathQA* Acc.	30.00	Mix	**34.50**	**33.00**
		Math	32.00	30.00
MAWPS Acc.	43.70	Mix	75.63	68.07
		Math	**78.99**	**81.51**
MBPP Pass@1	51.60	Mix	51.80	**52.80**
		Coding	**52.00**	52.80
HumanEval Pass@1	54.88	Mix	**57.93**	**57.32**
		Coding	56.71	56.71
Avg.	51.48	Mix	61.96	59.64
		In-Domain	**62.16**	**62.04**

Table 5: Performance comparison of CT3 using LLAMA-3-8B-INSTRUCT with baseline (w/o CT3), CT3 using a mixed-domain database, and CT3 with an in-domain database across various tasks and metrics. This experiment uses the smaller knowledge base (§4.5) for efficiency. * for GSM8K and MathQA indicates a subset of 200 randomly sampled test samples.

5 Conclusion

Test-time Training (TTT) is an effective method for improving model performance at the expense of more computation at inference time. We present CT3, a system that enables the application of TTT in client devices that might have resource constraints. The results of the CT3 prototype using supervisory signals from a knowledge base are a call to action to investigate further improvements to learning at test-time methods. Future work should explore more challenging scenarios and datasets to better understand the potential and limitations of collaborative test-time training. The current version of CT3 works on text queries. With increased sophistication, future versions of CT3 must handle more complex tasks and multimodal queries.

Limitations

Although our CT3 prototype produces compelling results and demonstrates how to alleviate the knowledge base storage burden at client devices, it also presents limitations. For instance, more research is needed to design methods to determine the right data mix at the knowledge base. In real-world applications, the effectiveness of CT3 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 enhancing the model's performance even further. Our experimental results have demonstrated the feasibility and potential benefits of CT3. In the current version of CT3'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 its transmission, e.g., utilizing Secure Sockets Layer (SSL), many open research challenges exist to increase the privacy and handling of the user's content on the server.

Ethical Considerations

Test-time training (TTT) techniques promise improvements in model performance, making them more accurate at the cost of more 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 TTT 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 affect the well-being of users of the system. The research community must continue investigating 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*.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. 2024. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*.

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.

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.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

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, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

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. 2023. A framework for few-shot language model evaluation.

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*.

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*.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California. Association for Computational Linguistics.

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.

Sebastián Ramírez. 2025. FastAPI — fastapi.tiangolo.com. https://fastapi.tiangolo.com/. [Accessed 13-03-2025].

Siva Reddy, Danqi Chen, and Christopher D. Manning. 2019. CoQA: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

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*.

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*.

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*.

Dataset	Domain	Link
CoQA Training set	Reading comprehension	link
MetaMath	Math	link
Orca-Math	Math	link
Math 50K	Math	link
The Stack Python	Coding	link
MBPP training set	Coding	link

Table 6: The details of the datasets in the knowledge base.

Task	Domain	Eval Framework	Link
CoQA	Reading Comp.	lm-eval-harness	link
GSM8K	Math	lm-eval-harness	link
MathQA	Math	llm-adapters	link
MAWPS	Math	llm-adapters	link
MBPP	Coding	bigcode-eval-harness	link
HumanEval	Coding	bigcode-eval-harness	link

Table 7: Various evaluation tasks.

A Datasets

The details of the knowledge base datasets and the evaluation tasks used in the experiments are presented in Tables 6 and 7. We evaluated several language models on a diverse set of tasks. We categorize the tasks into:

- **Reading Comprehension Tasks**
  - CoQA (Conversational Question Answering, Reddy et al. (2019))
- **Mathematical Reasoning Tasks**
  - GSM8K (Grade School Math 8K, Cobbe et al. (2021))
  - MathQA (Math Word Problem Question Answering, Amini et al. (2019))
  - MAWPS (MAth Word ProblemS, Koncel-Kedziorski et al. (2016))
- **Coding Tasks**
  - MBPP (Mostly Basic Python Problems, Austin et al. (2021))
  - HumanEval (Hand-Written Coding Evaluation Set, Chen et al. (2021))

B Additional Results with LSM

Table 8 shows that incorporating CT3 and LSM consistently enhances performance. These results underscore the effectiveness of LSM in boosting both performance and efficiency.

C Comparing SIFT and Nearest Neighbors Performance in the CT3 Setup

SIFT adds an additional step when retrieving relevant samples from the knowledge base, requiring an initial retrieval utilizing the nearest neighbors (NN) method. In our exploration to make the CT3 pipeline more efficient, we explore the impact of removing SIFT and using directly the results from NN. Table 9 presents a comparative analysis of CT3 with and without the SIFT step across various evaluation tasks. The results indicate that while SIFT provides a slight performance edge in some cases, the difference is not substantial. For instance, in the CoQA task, CT3 with SIFT achieves an Exact Match (EM) score of 68.85 and an F1 score of 81.13, compared to 67.43 EM and 80.73 F1 without SIFT. This marginal improvement suggests that SIFT's additional filtering step refines the sample selection process, leading to slightly better performance. In conclusion, while SIFT provides a slight performance boost, NN alone offers a simpler and more computationally efficient approach for sample retrieval in the CT3 system.

D Case Studies

Tables 10, 11, and 12 compare different methods to solve various problems, highlighting the effectiveness of CT3 in providing clear, concise and accurate solutions to user queries on client devices.

Method	Budget B	Reuse Threshold τ	Avg. Score	Impr.	Evaluation Speedup
LLAMA-3-8B-INSTRUCT	/	/	51.48	/	/
+ CT3	/	/	63.66	+24%	1.00×
+ CT3 + LSM	2	0.5	63.99	+24%	1.21×
	4	0.5	64.17	+25%	1.36×
	8	0.5	**64.75**	**+26%**	1.60×
	2	0.4	64.13	+25%	1.59×
	4	0.4	63.26	+23%	1.98×
	8	0.4	63.68	+24%	**2.24**×
LLAMA-3.1-8B-INSTRUCT	/	/	45.45	/	/
+ CT3	/	/	64.48	+42%	1.00×
+ CT3 + LSM	2	0.5	65.29	+44%	1.16×
	4	0.5	64.71	+42%	1.29×
	8	0.5	**65.53**	**+44%**	1.46×
	2	0.4	63.85	+41%	1.46×
	4	0.4	64.10	+41%	1.72×
	8	0.4	63.12	+39%	**1.89**×
MISTRAL-7B-INSTRUCT-V0.3	/	/	47.11	/	/
+ CT3	/	/	**53.48**	**+14%**	1.00×
+ CT3 + LSM	2	0.5	51.67	+10%	1.20×
	4	0.5	51.52	+9%	1.42×
	8	0.5	50.86	+8%	1.61×
	2	0.4	51.92	+10%	1.60×
	4	0.4	51.05	+8%	2.08×
	8	0.4	50.66	+8%	**2.28**×
QWEN2.5-3B-INSTRUCT	/	/	51.81	/	/
+ CT3	/	/	57.46	+11%	1.00×
+ CT3 + LSM	2	0.5	57.16	+10%	1.21×
	4	0.5	**57.76**	**+11%**	1.39×
	8	0.5	57.70	+11%	1.65×
	2	0.4	56.85	+10%	1.57×
	4	0.4	55.98	+8%	2.08×
	8	0.4	55.80	+8%	**2.32**×

Table 8: Performance comparison of LSM with varying budgets and reuse thresholds.

Method	CoQA EM	CoQA F1	GSM8K* EM (0-shot)	MathQA* Acc.	MAWPS Acc.	MBPP Pass@1	HumanEval Pass@1	Avg.	Impr.
QWEN2.5-3B-INSTRUCT	54.70	71.22	61.50	20.00	**87.82**	41.20	26.22	51.81	/
+ CT3 w/ SIFT	**68.85**	**81.13**	66.00	**34.00**	86.13	**46.20**	27.44	**58.54**	**+13%**
+ CT3 w/o SIFT	67.43	80.73	**68.50**	30.00	86.13	46.00	**28.66**	58.21	+12%

Table 9: Performance comparison of CT3 w/ SIFT and CT3 w/o SIFT (QWEN2.5-3B-INSTRUCT). The ablation studies use the smaller knowledge base (§4.5) for efficiency.

Prompt	Question: Charlie wants to sell beeswax candles. For every pound of beeswax, he can make 10 tapered candles. One pound of beeswax and the wicks cost $10.00 in supplies. If he sells each candle for $2.00 each, what is his net profit if he makes and sells 20 candles? Answer:
Reference Solution	1 pound of beeswax and the wick make 10 candles and he wants to make 20 candles so he needs 20/10 = «20/10=2»2 pounds of beeswax 1 pound of beeswax costs $10.00 and he needs 2 pounds so his supplies will cost 10*2 = $«10*2=20.00»20.00 He wants to sell each candle for $2.00 each and he's selling 20 candles so that's 2*20 = $«2*20=40.00»40.00 If he makes $40.00 selling candles and his supplies cost $20.00 then his net profit is 40-20 = $«40-20=20.00»20.00 #### 20
w/o CT3	He spends $10.00 / candle * 20 candles = $200.00 on beeswax and wicks. He earns 2 * $2.00 = $4.00 per candle. So he earns 20 candles * $4.00 = $80.00 total from sales. That means his net profit is $80.00 - $200.00 = -$120.00 Therefore, the answer is -120. ✗
CT3	He uses 2 pounds of beeswax because 20 / 10 = 2 The supplies cost $20 because 2 x 10 = 20 He earns $40 from selling the candles because 20 x 2 = 40 His net profit is $20 because 40 - 20 = 20 Therefore, the answer is 20. ✓

Table 10: A case comparing different methods for solving a profit calculation problem with QWEN2.5-3B-INSTRUCT. The table illustrates the effectiveness of CT3 in accurately solving the problem, as opposed to the incorrect solution provided by the baseline.

Prompt	Question: Meredith is a freelance blogger who writes about health topics and submits to clients each day as her permanent job. A blog article takes an average of 4 hours to research and write about. Last week, she wrote 5 articles on Monday and 2/5 times more articles on Tuesday than on Monday. On Wednesday, she wrote twice the number of articles she wrote on Tuesday. Calculate the total number of hours she spent writing articles in the three days. Answer:
Reference Solution	If she wrote five articles on Monday, then on Tuesday she wrote 2/5* 5 = 2 more articles. The total number of articles she wrote on Tuesday is 5+2 = «5+2=7»7 On Wednesday, the number of articles was twice the number she wrote on Tuesday, so it was 2*7 = «2*7=14»14 articles. Over the three days, she wrote 5+7 +14 = «5+7+14=26»26 articles. If each article takes her 4 hours to research and write about, she spent 4*26 = «4*26=104»104 hours on all the articles. #### 104
w/o CT3	To determine the total number of hours Meredith spent writing articles over the three days, we need to follow these steps: 1. **Calculate the number of articles written on Tuesday:** - On Monday, Meredith wrote 5 articles. - On Tuesday, she wrote \\(\\frac25 \\) times more articles than on Monday. - First, find \\(\\frac25 \\) of the articles written on Monday: \\[\\frac25 \\times 5 = 2\\] - Therefore, the total number of articles written on Tuesday is: \\[5 + 2 = 7\\] 2. **Calculate the number of articles written on Wednesday:** - On Wednesday, she wrote twice the number of articles she wrote on Tuesday. - The number of articles written on Tuesday is 7. - Therefore, the number of articles written on Wednesday is: \\[2 \\times 7 = 14\\] 3. **Calculate the total number of articles written over the three days:** - Articles on Monday: 5 - Articles on Tuesday: 7 - Articles ✗
CT3	To find out how many articles Meredith wrote on Tuesday, we need to calculate 2 / 5 times more than Monday's articles. Since she wrote 5 articles on Monday, 2 / 5 * 5 = 2 more articles than Monday. This means she wrote 5 + 2 = 7 articles on Tuesday. On Wednesday, she wrote twice the number of articles she wrote on Tuesday, which is 2 * 7 = 14 articles. In total, she wrote 5 + 7 + 14 = 26 articles over the three days. Since it takes her 4 hours to research and write about one article, she spent 4 * 26 = 104 hours writing articles in the three days. Therefore, the answer is 104 hours. ✓

Table 11: A case comparing different methods for solving a time calculation problem with QWEN2.5-3B-INSTRUCT. The table demonstrates that the baseline (w/o CT3) unnecessarily complicates the problem, while CT3 effectively maximizes the model's potential for the current query. CT3 utilizes CT3 to provide a clear, concise, and accurate solution to the user's query at the client device.

Prompt	Question: A wooden bridge can carry no more than 5000 pounds. A delivery truck filled with identical boxes, each weighing 15 pounds, will pass over the bridge. The combined weight of the driver and the empty truck is 3755 pounds. What is the maximum number of boxes which can be loaded onto the truck while not exceeding the bridge's weight limit? Answer:
Reference Solution	The boxes can weigh up to 5000 pounds - 3755 pounds = «5000-3755=1245»1245 pounds in total. There can be 1245 / 15 = «1245/15=83»83 boxes loaded onto the truck without exceeding the bridge's weight limit. #### 83
w/o CT3	First find the total weight capacity of the truck by adding the weight limit of the bridge to the weight of the driver and empty truck: 5000 pounds + 3755 pounds = 8755 pounds. Then divide that number by the weight of each box to find the maximum number of boxes: 8755 pounds / 15 pounds / box = 583.666... boxes. Since we can't have a fraction of a box, we call this 583 boxes. Therefore, the answer is 583. ✗
CT3	First, we need to find out how much weight the boxes can add to the truck without exceeding the bridge's weight limit. We do this by subtracting the weight of the driver and the empty truck from the bridge's weight limit: 5000 - 3755 = 1245 pounds. Next, we divide the remaining weight capacity by the weight of a single box to find out how many boxes can be added: 1245 / 15 = 83 boxes. The answer is 83. ✓

Table 12: A case for comparing different methods of solving a weight calculation problem using QWEN2.5-3B-INSTRUCT. It demonstrates the model's misunderstanding of the problem when CT3 is not used, specifically by incorrectly adding the bridge's weight limit to the truck's weight.