FLOP-Efficient Training: Early Stopping Based on Test-Time Compute Awareness

Anonymous ACL submission

Abstract

Scaling training compute, measured in FLOPs, has long been shown to improve the accuracy of large language models, yet training remains resource-intensive. Prior work shows that increasing test-time compute (TTC)—for example through iterative sampling—can allow smaller models to rival or surpass much larger ones at lower overall cost. We introduce TTC-aware training, where an intermediate checkpoint and a corresponding TTC configuration can together match or exceed the accuracy of a fully trained model while requiring substantially fewer training FLOPs. Building on this insight, we propose an early stopping algorithm that jointly selects a checkpoint and TTC configuration to minimize training compute without sacrificing accuracy. To make this practical, we develop an efficient TTC evaluation method that avoids exhaustive search, and we formalize a break-even bound that identifies when increased inference compute compensates for reduced training compute. Experiments demonstrate up to 92% reductions in training FLOPs while maintaining and sometimes remarkably improving accuracy. These results highlight a new perspective for balancing training and inference compute in model development, enabling faster deployment cycles and more frequent model refreshes.

1 Introduction

Large language models (LLMs) have driven significant progress in natural language processing and beyond (Vaswani et al., 2017; Achiam et al., 2023; Liu et al., 2024a). Open-source architectures such as Llama and Mistral demonstrate strong capabilities in text understanding and generation across diverse tasks. However, these performance gains come at a high cost: training state-of-the-art LLMs demands massive compute, large datasets, and results in considerable financial and environmental burdens (Strubell et al., 2020; Hajimolahoseini et al., 2023; Tawfilis et al., 2025). For example, training Llama-3.1-405B required 30.84 million GPU hours on NVIDIA H100 hardware, while continued pretraining or finetuning consumes over 2,000 H100 GPU hours per billion tokens. Usually, only large labs and companies have access to these resources. These challenges underscore the growing need for different strategies that improve efficiency without compromising accuracy (AI, 2024).

The development of LLMs has been guided by scaling laws describing how accuracy improves with increases in model size, dataset size, and compute. (Kaplan et al., 2020) established that training loss decreases as a power law with respect to model parameters, training tokens, and compute. (Hoffmann et al., 2022) later proposed the *Chinchilla paradigm*, showing that—under a fixed compute budget—optimal performance is achieved by balancing model size and training tokens, challenging the trend toward excessively large models.

While these frameworks focus primarily on training compute, recent research has begun to account for inference-time costs (Moradi et al., 2025). (Sardana et al., 2023) introduced *inference-aware scaling*, demonstrating that for high-deployment scenarios, smaller models trained longer can yield better overall cost-efficiency. Empirical studies by (Snell et al., 2024) and (Wu et al., 2024) show that increasing *test-time compute* (TTC) through methods such as verifier-guided search, or iterative sampling allow smaller models to match or even exceed the performance of much larger models at a lower total compute cost. Related work on repeated sampling and iterative refinement (Li et al., 2024; Brown et al., 2024; Chen et al., 2024) highlights the potential of multipass inference to increase accuracy without training.

Despite these advances, most studies compare fully trained models of different sizes while overlooking training dynamics within a single model, particularly how performance evolves with vary-

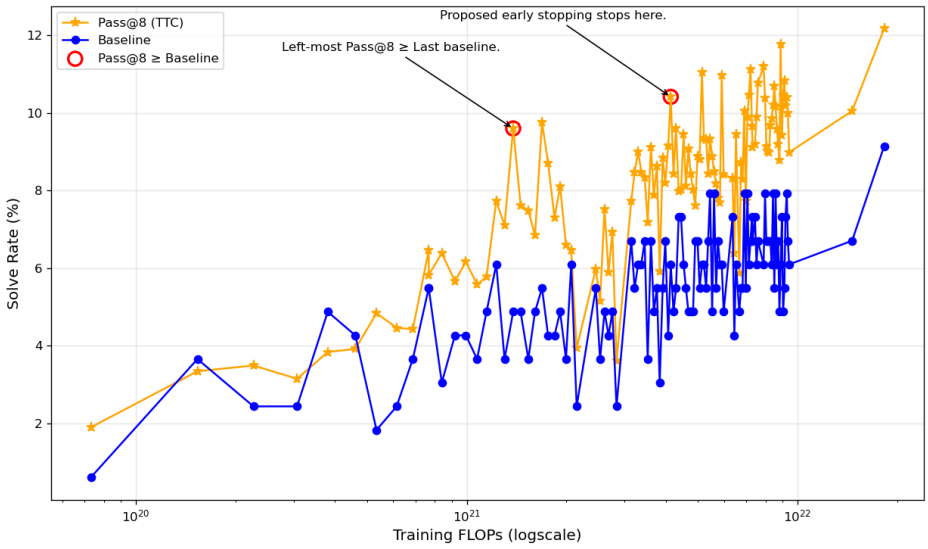


Figure 1: Solve rate (%) versus training FLOPs for TinyLlama on HumanEval. The solid blue line shows the fully trained baseline without TTC, while the solid orange line shows models using TTC (Pass@8). The red circle indicates that TTC can achieve accuracy at least equivalent to a fully trained checkpoint without TTC, but with significantly fewer training FLOPs. Discussion and results for other K values (e.g., Pass@4) are in Section 4.

ing training FLOPs. This gap is critical: if TTC is considered, models could achieve competitive or even superior performance through early stopping. Existing early stopping methods, however, remain TTC-agnostic, focusing only on validation accuracy and neglecting the accuracy gains achievable at inference. Likewise, current TTC techniques, especially those based on repeated sampling, increase inference cost as K grows, where K denotes the number of iterative passes (Pass@K). These techniques do not consider predicting an optimal K to run TTC more efficiently.

These shifting paradigms, from Chinchilla-optimal training to inference-aware scaling, suggest a unifying principle: modern LLM development should jointly optimize training and inference compute. We extend this principle by incorporating *TTC awareness into training*, enabling early stopping decisions based on projected TTC accuracy. Our approach reduces training FLOPs while at least preserving final accuracy, and a new small-K evaluation strategy identifies the optimal K with minimal inference overhead. As Figure 1 shows, TTC-aware early stopping matches and could sometimes exceed fully trained checkpoint without TTC at a fraction of the compute, *allowing faster model deployments and more frequent model refreshes*.

Overall, this paper contributes the following: 1) Introduces *TTC-aware training*, a strategy to reduce total FLOPs while maintaining or improving model accuracy. 2) Proposes a TTC-aware early stopping algorithm that projects TTC accuracy during training and halts when favorable FLOPs–accuracy trade-offs are achieved. This approach enables faster model refreshes and deployment cycles for large-scale model development, and also benefits academic and open-source labs with limited compute. 3) Develops an efficient TTC estimation method to predict the optimal K from small-K runs, lowering inference cost. 4) Provides a break-even analysis that identifies when TTC-aware training and inference match or outperform traditional training in total compute. 5) Empirically validates the effectiveness of TTC-aware training through extensive experiments.

2 Proposed Method: TTC-Aware Training Procedure

Figure 2 illustrates the overall proposed training procedure with TTC-awareness to achieve FLOP-efficient training. In this procedure, training hyperparameters and training FLOPs budget, B, are given before training. The goal is to train a model such that its final validation accuracy under TTC inference is maximized, subject to the constraint that the total training and inference costs with TTC are less than without TTC. Based on the TTC-awareness insight, we propose a FLOP-efficient training framework that matches or exceeds the validation performance of a fully trained check-

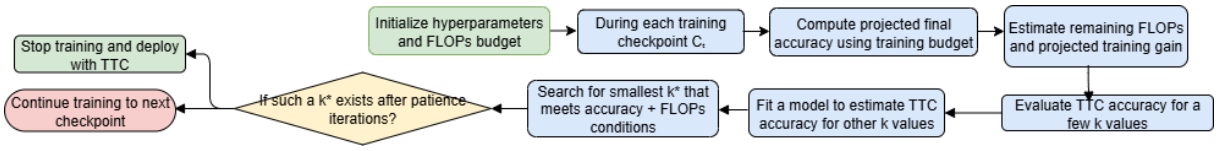


Figure 2: Flowchart of the proposed TTC-aware training procedure. The method efficiently selects the checkpoint with TTC configuration that is better in terms of FLOPs and accuracy than the fully trained checkpoint, enabling early stopping. Algorithm is in Appendix D.

point under budget B. During training, we collect validation accuracy values $\{\mathcal{A}_i\}_{i=1}^n$ from selected intermediate checkpoints $\{t_i\}_{i=1}^n$, and use them to construct a curve-fitted projection of the expected validation accuracy $\hat{\mathcal{A}}(B)$ at budget B. As training progresses, this projection is dynamically refined as more checkpoints are observed.

This final accuracy projection relies on empirical observations that typical learning curves follow a saturating pattern: a rapid initial increase in accuracy followed by diminishing returns. Thus, we fit the collected validation accuracy data to an exponential saturation function:

$$\hat{\mathcal{A}}(B) = a\left(1 - e^{-bB}\right) + c \tag{1}$$

where a controls the scale of the gain in accuracy, b determines the rate of convergence (how fast the model saturates), and c represents the base accuracy at initialization. These parameters are learned by minimizing the mean squared error between observed validation accuracies and the fitted curve. The fitted function $\hat{\mathcal{A}}(B)$ is then used to predict final validation performance under budget B, guiding early stopping and inference allocation.

To support this framework with efficient inference, we introduce a TTC-aware mechanism that estimates the optimal number of inference samples per query, denoted K^*, without incurring excessive compute. Since standard TTC performs inference by sampling the model K times and aggregating the responses, it can become prohibitively expensive if K is large. Instead, we propose to approximate the relationship between validation accuracy under TTC at time t and the number of samples K using a sigmoid curve:

$$\hat{\mathcal{A}}_K(t) = \frac{L}{1 + e^{-k(K - x_0)}}, \tag{2}$$

where L is the maximum achievable TTC accuracy, k controls the growth rate of accuracy with increasing K, and x_0 is the inflection point, where accuracy reaches half of its maximum. This relation aligns with the intuitive expectation that increasing K will eventually lead to a saturation point, beyond which accuracy gains level off (more evidence is in Figure 7 in the appendix).

These parameters are estimated using non-linear least squares fitting over a few sampled values of K, obtained by querying the model at small K values (e.g., $K = 1, 2, 4$). Evaluating once at K implicitly yields predictions for all $K' \leq K$, so querying at $K = 4$ provides the effective results for $K = 1, 2, 4$ at no additional cost. This formula allows us to extrapolate and predict K^*—the smallest value of K that meets both budget and accuracy constraints.

The optimal K^* must satisfy the total FLOPs constraint:

$$\begin{aligned}\text{FLOPs}_{\text{train}}[t] + \text{FLOPs}_{\text{inference}}[t, K^*] \\ < \text{FLOPs}_{\text{train}}[B] + \text{FLOPs}_{\text{inference}}[B, 1]\end{aligned} \tag{3}$$

where $\text{FLOPs}_{\text{train}}[t]$ is the training cost up to timestep t, $\text{FLOPs}_{\text{inference}}[t, K^*]$ is the cost of evaluating TTC inference at that timestep using K^* samples. In addition, K^* must meet the projected accuracy requirement:

$$\hat{\mathcal{A}}_{K^*}(t) \geq \hat{\mathcal{A}}(B) \tag{4}$$

That is, the TTC accuracy at timestep t with K^* samples must be at least as good as the projected final validation accuracy under full budget B. If such a K^* exists after a patience window of p checkpoints, training is terminated early; otherwise, training continues to the next checkpoint.

3 Break-Even Bound

The proposed TTC-aware training procedure with early stopping can introduce additional computation, because TTC-driven inference is periodically performed both during training and deployment. It is therefore useful to determine *when* these extra

costs are offset by the training savings from TTC-aware early stopping. We refer to this as the *break-even* point: the largest amount of downstream inference that can be performed before the total cost of "TTC-aware training + deployment" exceeds the cost of training the same model *without* TTC.

Here, the total cost explicitly includes both the training cost (which may already incorporate TTC-driven inference) and the deployment cost, i.e., the FLOPs or tokens required to serve end-user queries. Since modern LLMs are not trained once and frozen indefinitely, but rather are *continuously refreshed* through pretraining, finetuning, and ongoing updates (Ramaswamy et al., 2019; Gamal et al., 2023; Amer et al., 2024). Thus, characterizing this bound is especially important: it indicates the consumption of inference tokens before the model should be refreshed.

We denote the number of *training* tokens for the baseline schedule (without TTC) by N_{train}, and the number of *inference* tokens N_{infer}. Here, N_{infer} accounts for all inference tokens, including those used during TTC sampling as well as tokens served to end users. The ratio of training FLOPs with TTC relative to the baseline is $r \in (0, 1]$, where $r < 1$ indicates training savings from earlier stopping. The inference cost of serving the model with TTC is captured by a multiplier $\lambda > 1$, relative to serving the baseline model without TTC. Assuming that training FLOPs per token are roughly $6\times$ the inference FLOPs per token (accounting for forward + backward passes and optimizer updates), the break-even condition — i.e., the maximum number of inference tokens that can be served while ensuring TTC does not exceed the total FLOPs of the baseline without TTC — is:

$$N_{\text{infer}} \leq \frac{6\,(1-r)}{\lambda - 1} \cdot N_{\text{train}}. \tag{5}$$

A detailed derivation and discussion are provided in Appendix A.

4 Experimental Results

This section first outlines the experimental setup, including the models, datasets, and hyperparameters used to ensure reproducibility. We then present detailed analyses that establish the TTC-aware insight. Finally, we assess the effectiveness of the proposed early stopping method with curve fitting informed by this insight.

	TinyLlama	**Pythia**	**FineMath**
Size (B)	1.1	1 / 2.8 / 6.9	3
Arch. Base	LLaMA-2 (GQA)	GPT-style	LLaMA-3
Training Data	SlimPajama StarCoder	The Pile	FineWeb-Edu FineMath
Tokens	∼2T	300B	160B
Optimizer	AdamW	Adam + ZeRO	AdamW

Table 1: Models used in our experiments.

4.1 Experimental Setup

Models and Checkpoints We evaluate our approach using three open-source model families: TinyLlama (Zhang et al., 2024), Pythia (Biderman et al., 2023), and FineMath (Liu et al., 2024b). These models were selected because they provide publicly available intermediate checkpoints, unlike most large-scale models where reproducing training would be prohibitively expensive. Table 1 summarizes the models under evaluation. We use TinyLlama and Pythia to assess the impact of TTC-aware training in a *pretraining* setting, while FineMath enables evaluation in a *continued training* scenario.

Datasets We evaluate our models on a diverse set of benchmarks spanning code generation, reading comprehension, and mathematical reasoning. Specifically, we include HumanEval (Chen et al., 2021), a standard benchmark for assessing code synthesis from natural language prompts; DROP (Dua et al., 2019), which tests discrete reasoning and reading comprehension over passages; Math-500 (Hendrycks et al., 2021), a curated set of advanced mathematics problems; GSM8k (Cobbe et al., 2021), a collection of grade-school math word problems designed to evaluate multi-step arithmetic reasoning.

Inference Hardware and Environment Inference experiments are carried out on a 8xNVIDIA Tesla V100 GPU. Float16 inference is enabled because it speeds up the inference while maintaining accuracy. Our inference environment is built on top of the InstructEval benchmark (Chia et al., 2023).

TTC Hyperparameter Settings Our experiments use a temperature of 0.8, a patience value of 10, a maximum input sequence length of 1,024 tokens, and a maximum output length of 512 tokens. The rest of configurations are borrowed from InstructEval repository (Chia et al., 2023).

4.2 TTC-Aware Training Insight

TTC-Aware training allows models to reach equal or better accuracy using significantly fewer training

Model	Dataset	Step	Tokens	FLOPs	Baseline	Pass@4	Pass@8	Pass@16	Pass@32	Pass@64
Pythia-1B	HumanEval	80,000	167.8T	9.6×10^{20}	1.8	2.8	**3.8**	4.9	6.4	7.9
Pythia-1B	HumanEval	143,000	299.9T	1.7×10^{21}	**3.04**	4.01	5.5	7.5	9.8	12.2
Pythia-1B	GSM8k	80,000	167.8T	9.6×10^{20}	1.74	6.06	10.99	18.07	27.17	35.24
Pythia-1B	GSM8k	143,000	299.9T	1.7×10^{21}	2.20	6.97	11.22	18.34	27.52	36.16
Pythia-2.8B	HumanEval	80,000	167.8T	1.93×10^{21}	8.5	6.47	**8.84**	11.49	14.17	17.07
Pythia-2.8B	HumanEval	143,000	299.9T	3.43×10^{21}	**6.7**	8.38	11.02	14.01	17.53	20.73
Pythia-2.8B	GSM8k	80,000	167.8T	1.93×10^{21}	1.54	6.29	11.29	18.19	26.99	35.86
Pythia-2.8B	GSM8k	143,000	299.9T	3.43×10^{21}	2.04	6.30	11.30	18.73	27.75	38.21
Pythia-6.9B	HumanEval	80,000	167.8T	6.7×10^{21}	6.8	8.17	11.09	13.96	16.98	20.12
Pythia-6.9B	HumanEval	143,000	299.9T	1.2×10^{22}	6.8	10.28	13.22	16.40	20.15	24.39
Pythia-6.9B	GSM8k	80,000	167.8T	6.7×10^{21}	1.74	7.05	12.28	18.50	29.04	40.33
Pythia-6.9B	GSM8k	143,000	299.9T	1.2×10^{22}	2.80	7.53	13.16	19.27	29.67	40.79

Table 2: Comparison of intermediate (80k) and final (143k) checkpoints across Pythia models of different sizes using **HumanEval** and **GSM8k** datasets. Results are reported using Pass@k metrics. TTC-aware training at earlier steps achieves accuracy comparable to or exceeding the fully trained baseline without TTC.

FLOPs or less data. This section presents four experiments to demonstrate this insight.

Our *first* experiment evaluates the available TinyLlama checkpoints on HumanEval. As shown in Figure 1, an intermediate checkpoint achieves at least the same or even better accuracy than the fully trained baseline without TTC, while providing up to 92.44% training FLOP savings. In particular, this checkpoint produces a positive accuracy gain of 0.45%. When accounting for the additional FLOPs consumed by TTC inference during training, the total savings remain substantial at 85.5%. Although the final checkpoint with Pass@8 attains higher accuracy, this improvement comes at the cost of a substantially larger FLOPs budget compared to the intermediate checkpoint.

Figure 4 provides a more detailed analysis for TinyLlama on HumanEval across different Pass@K metrics. We observe three consistent trends. First, there always exists an intermediate checkpoint where the Pass@K performance is at least as strong as that of the fully trained baseline without TTC, while requiring substantially fewer FLOPs. Second, when comparing across different K values, if the intermediate checkpoint uses a *larger* K than the fully trained model (e.g., Pass@8 or Pass@16 vs. Pass@4), it can often maintain or even surpass the accuracy of the fully trained baseline while still providing FLOPs savings. However, if the intermediate checkpoint uses a *smaller* K than the fully trained baseline, the fully trained model generally remains superior, and achieving comparable accuracy would require training to the full budget. Third, when both models are compared at the same K, the fully trained baseline attains higher accuracy, but only at the cost of a substantially larger FLOPs budget.

To validate this insight, the *second* experiment replicated the TinyLlama experiment on DROP. The results, shown in Figure 3, closely align with those on HumanEval: TTC enables early stopping while preserving—and in some cases improving—the final accuracy compared to the fully trained model. This confirms that the TTC-aware training phenomenon is not limited to a single dataset, but rather generalizes across tasks.

In our *third* experiment, we extend the analysis to the Pythia family of models with 1B, 2.8B, and 6.9B parameters (Biderman et al., 2023). For each model, we compare an intermediate checkpoint at 80k training steps with the fully trained baseline at 143k training steps (see Table 2).

Across both HumanEval and GSM8k, we observe the same TTC-aware trends identified earlier with TinyLlama. Intermediate checkpoints with TTC consistently match or exceed the performance of fully trained baselines while requiring substantially fewer FLOPs. For example, on HumanEval, the 1B Pythia model at 80k achieves Pass@8 accuracy (3.8%) comparable to its 143k baseline without TTC (3.04%), while the 2.8B model at 80k with Pass@8 (8.84%) not only surpasses its 143k baseline (6.7%) but also outperforms the much larger 6.9B baseline at 143k (6.8%).

A similar pattern appears on GSM8k: the 2.8B checkpoint at 80k matches or exceeds the 143k baseline across all Pass@K values, and the 6.9B checkpoint at 80k with TTC outperforms its fully trained counterpart without TTC. Furthermore, when comparing different K values within the same model size, if the intermediate checkpoint uses a larger K than the fully trained baseline, its accuracy is often higher. For example, Pythia-2.8B at 80k with Pass@8 (11.29%) surpasses Pythia-2.8B at 143k with Pass@4 (6.3%). Consistent patterns are also observed across Pythia model sizes.

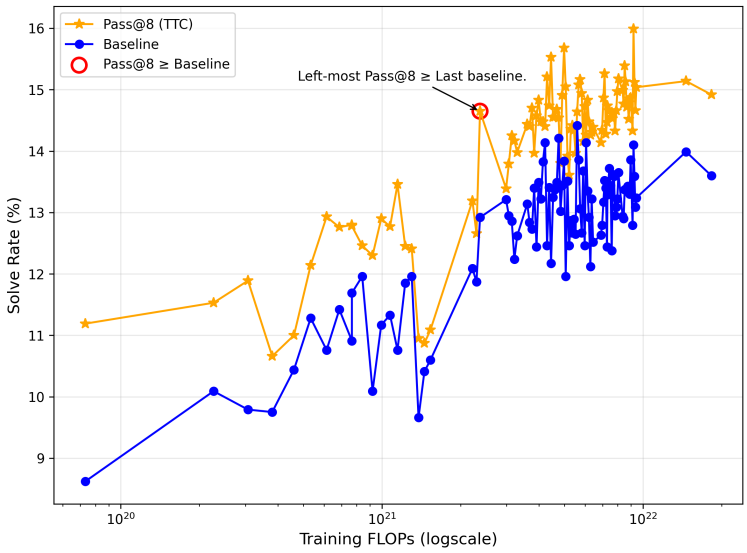


Figure 3: Results of TTC-aware training on the DROP dataset using TinyLlama-1B. Similar to Figure 1, applying TTC allows early stopping while saving up to 92% of the training FLOPs without sacrificing final accuracy. This demonstrates that TTC awareness generalizes across datasets and is not limited to a single benchmark.

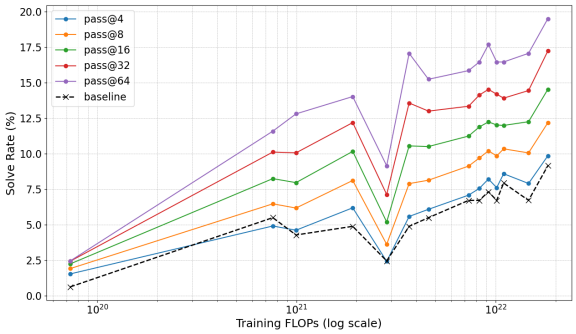


Figure 4: TinyLlama on HumanEval across Pass@K. Intermediate checkpoints can match or exceed the fully trained baseline with far fewer FLOPs, sometimes outperforming the baseline at higher K, while the final checkpoint is better only at the cost of much larger FLOPs.

Overall, these results reinforce the findings from TinyLlama: the TTC-aware effect generalizes across model scales and datasets, demonstrating that intermediate checkpoints can achieve comparable or better performance at a fraction of the training cost.

In all previous experiments, we focused on Pass@K, which captures the potential performance of a model under multiple sampled outputs. As a result, the *fourth* experiment extends the TTC-aware analysis to more advanced TTC techniques beyond Pass@K. Specifically, we evaluate several approaches: (1) the Baseline without TTC, (2) Pass@K, (3) Majority Voting, (4) Diverse Tree Search (DVTS) with naive weighting, (5) DVTS with weighted aggregation, and (6) Compute-Optimal Search, which selects the best-performing method among Majority Voting and the two DVTS variants (Beeching et al., 2024; Snell et al., 2024).

In DVTS with weighted aggregation, identical responses are grouped, and the final answer is chosen based on the highest cumulative reward across all identical responses. In contrast, DVTS with naive weighting treats each generated response independently, selecting the one with the highest reward model (RM) score as the final output.

We apply these techniques to the FineMath model in a continued-training setup at 10B, 30B, 40B, and 50B training tokens within the mathematics domain and evaluate on the Math-500 benchmark. An important observation from Table 3 is that even when a lower training budget underperforms at a given Pass@K, there often exists a larger K where it surpasses higher-budget checkpoints evaluated at smaller K. For example, at $K{=}2$ the 10B checkpoint underperforms the 50B one (15.05% vs. 18.28%), but at $K{=}4$, the 10B checkpoint outperforms the 50B checkpoint with $K{=}2$ (24.73% vs. 18.28%). This pattern recurs across multiple settings (e.g., 10B vs. 40B), illustrating the broader TTC-aware insight in a continued training scenario as shown with FineMath.

We further find that this TTC-aware insight generalizes beyond Pass@K to verifier-based and ag-

gregation metrics. For instance, the 10B checkpoint with K=16 under the DVTS-WEIGHTED verifier achieves a 19.35% solve rate, surpassing the 50B checkpoint without TTC (15.05%). Similarly, the 50B checkpoint improves from 13.98% at K=2 to 21.51% at K=4, demonstrating the consistent performance gains achieved with larger K. To reach comparable or better accuracy than the 50B checkpoint, one could instead use the 10B checkpoint with K=32 (33.33%) or the 30B checkpoint with K=16 (24.73%) or the 40B checkpoint with K=32 (41.94%).

Across all TTC techniques evaluated, we observe that solve rates generally increase as K grows. However, verifier-based TTC scores are typically lower than raw Pass@K values, underscoring the importance of designing strong verifiers for realistic evaluation. Among the TTC techniques, DVTS-WEIGHTED consistently yields the best overall performance, while the Compute-Optimal Search provides an approximate upper bound over the tested methods.

Overall, these findings reinforce that the TTC-aware insight persists even under verifier-based and aggregation metrics, further validating the generality of the TTC framework across both metric types and model scales.

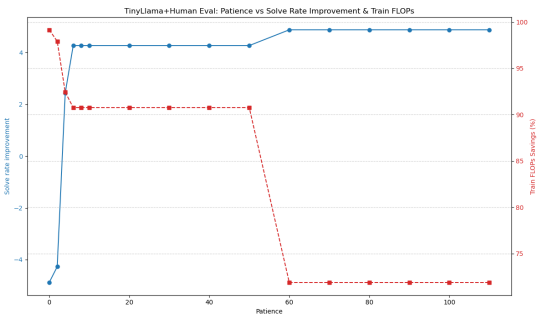


Figure 5: Effect of the patience hyperparameter on TinyLlama (HumanEval). Larger patience generally increases solve-rate at TTC Pass@16 relative to the fully trained checkpoint without TTC, but also raises the fraction of training FLOPs consumed. An intermediate value (around 10) balances gains and cost, yielding substantial solve-rate improvement while keeping training FLOPs reasonable. Patience also mitigates occasional curve-fitting instabilities in the early-stopping procedure. However, setting patience too high may negate the benefits of early stopping by driving the model to train nearly to the full budget.

4.3 Early Stopping Analysis and Results

The proposed early stopping algorithm achieves noticeable gains. In TinyLlama with the HumanEval dataset, it reaches Pass@8 accuracy roughly 1.3% higher than the fully trained checkpoint without TTC while providing 77.3% training FLOP savings, and delivers a 4.3% improvement in Pass@16 with 90.76% training FLOP savings. Using our early stopping method, Pass@8 reduces training FLOPs by 77.3% relative to the baseline, while we could practically reach 92.44% overall FLOP savings. Similar trends hold across datasets and model families, including Pythia (Biderman et al., 2023). Overall, TTC-aware training can save up to 92% of training FLOPs without sacrificing accuracy and often improves it.

We also implemented curve fitting for both training and TTC inference. Figure 6 illustrates exponential curve fitting on the checkpoints, showing strong alignment and minimal fitting error. Additionally, Figure 7 (Appendices) presents sigmoid curve projections, which closely track TTC accuracy across checkpoints at different FLOP levels. To guard against potential fitting failures, we incorporate *patience*, a standard hyperparameter in early stopping.

Figure 5 illustrates the impact of varying patience on the solve rate improvement for TTC Pass@16—relative to the fully trained checkpoint—and the corresponding training FLOPs savings for TinyLlama on the HumanEval dataset. A clear tradeoff emerges: increasing patience generally improves the solve rate but also decreases the fraction of training FLOPs saved. While larger patience values yield slightly higher improvements, the FLOPs savings decrease, resulting in diminishing returns in computational efficiency. An intermediate patience value of around 10 appears optimal, providing substantial solve rate gains while keeping training FLOPs within a reasonable range. This underscores the importance of carefully tuning patience to balance performance and efficiency.

5 Conclusion

We introduced *TTC-aware training*, demonstrating that models can match or even surpass the accuracy of fully trained baselines while using only a fraction of the training FLOPs. Our proposed early stopping and efficient TTC estimation methods consistently deliver strong empirical results across diverse benchmarks and model scales.

Train Tokens	K	Baseline	Pass@K	Majority	Naive	Weighted	CTS
10B	2	12.90	15.05	12.90	12.90	12.90	13.98
	4	12.90	24.73	11.83	11.83	12.90	16.13
	16	12.90	53.76	19.35	21.51	19.35	29.03
	32	12.90	67.74	26.88	23.66	33.33	44.09
	64	12.90	74.19	25.81	24.73	34.41	39.78
30B	2	7.53	12.90	8.60	11.83	11.83	12.90
	4	7.53	21.51	11.83	19.35	20.43	21.51
	16	7.53	46.24	19.35	24.73	24.73	33.33
	32	7.53	61.29	23.66	23.66	27.96	36.56
	64	7.53	68.82	31.18	26.88	35.48	45.16
40B	2	11.83	13.98	11.83	12.90	12.90	13.98
	4	11.83	24.73	15.05	18.28	17.20	21.51
	16	11.83	51.61	29.03	22.58	29.03	38.71
	32	11.83	67.74	37.63	27.96	41.94	50.54
	64	11.83	73.12	39.78	32.26	49.46	52.69
50B	2	15.05	18.28	13.98	13.98	13.98	15.05
	4	15.05	29.03	18.28	18.28	21.51	24.73
	16	15.05	59.14	36.56	27.96	37.63	45.16
	32	15.05	66.67	38.71	32.26	39.78	49.46
	64	15.05	76.34	40.86	34.41	44.09	53.76

Table 3: Solve Rate (%) of *FineMath* checkpoints (10B, 30B, 40B, 50B tokens) across different K and Verifier is Skywork-o1-Open-PRM-Qwen-2.5-1.5B. We report Baseline without TTC, Pass@k, Majority, DVTS Naive, DVTS Weighted, and Compute-Optimal Search (CTS) aggregation methods on Math-500.

Moreover, TTC-aware early stopping could offer an alternative to continual training: rather than resuming training on new data, one can freeze the model early and leverage additional test-time compute to recover or exceed the performance of a fully trained model. These improvements could enable faster model development and more frequent model refreshes, potentially benefiting both large-scale model efforts and academic or open-source research.

We further introduced the breakeven bound analysis to explicitly account for the total cost of training, inference, and user queries at deployment, and presented methods to forecast the accuracy of a fully trained baseline without TTC. Our efficient TTC estimation techniques enable these evaluations at reduced cost. Beyond reducing training expenses, this work promotes a unified view in which training decisions are guided by the eventual deployment budget and inference strategy. As demonstrated in our results, the proposed early stopping approach can achieve up to 92% training FLOP savings.

Limitations

Models and Checkpoints. Our experiments are limited to publicly available models and checkpoints with sizes up to 6.9B parameters. While we attempted to cover both GPT-style and LLaMA-style architectures, we acknowledge that there exist many other models—larger models, multimodal models, and models specialized for images or video—that are not included in our evaluation.

Dataset and Model Limitations. Despite our efforts, some combinations of models and difficult datasets (e.g., TinyLlama with Math-500) were not feasible due to limited compute resources and insufficient domain-specific pretraining (e.g., TinyLlama checkpoints were not trained enough on math content). This limits our ability to fully evaluate TTC-aware training across all benchmark combinations. Please see Appendix C for further details.

TTC Inference Efficiency. Efficient inference under TTC remains a key challenge. Realizing the benefits of TTC-aware training in practice may require carefully designed inference strategies, such as beam search or diverse tree search, while accounting for latency constraints. Without such considerations, potential gains may be difficult to achieve in deployment.

Verifier-Based Evaluation. We report verifier-based results primarily in the continued pretraining scenario. While extending these evaluations across multiple verifier types, scales, and model families would offer a more comprehensive view of TTC-aware training effectiveness, our primary objective in this work is to demonstrate that TTC-aware training can substantially reduce FLOPs while main-

taining or even improving accuracy. The continued pretraining setup serves to highlight this benefit in a realistic setting. Broader verifier-based studies, including across architectures and scales, are promising directions for future work. In addition, our proposed early stopping strategy further emphasizes the practical side of applying TTC-aware training efficiently.

TTC-Aware Training Scaling Law. Establishing a principled scaling law to determine the optimal stopping point—where the benefits of TTC-aware training are maximized—remains an open question and is left for future work.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Meta AI. 2024. Llama 3.1 405b. https://huggingface.co/meta-llama/Llama-3.1-405B. Accessed: 2025-08-21.

Hossam Amer, Joe Osborne, Michael Zaki, and Mohamed Afify. 2024. On-device emoji classifier trained with gpt-based data augmentation for a mobile keyboard. *arXiv preprint arXiv:2411.05031*.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, and 1 others. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Edward Beeching, Lewis Tunstall, and Sasha Rush. 2024. Scaling test-time compute with open models. Hugging Face Space. https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, and 1 others. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

Chung-Chi Chen, Hen-Hsen Huang, and Hsin-Hsi Chen. 2021. Evaluating the rationales of amateur investors. In *Proceedings of the Web Conference 2021 (WWW '21)*, pages 372–383. ACM.

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei A Zaharia, and James Y Zou. 2024. Are more llm calls all you need? towards the scaling properties of compound ai systems. *Advances in Neural Information Processing Systems*, 37:45767–45790.

Yew Ken Chia and 1 others. 2023. Instructeval: Towards holistic evaluation of instruction-tuned large language models. *arXiv preprint arXiv:2306.04757*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 2368–2378. Association for Computational Linguistics.

Karim Gamal, Ahmed Gaber, and Hossam Amer. 2023. Federated learning based multilingual emoji prediction in clean and attack scenarios. *arXiv preprint arXiv:2304.01005*.

Habib Hajimolahoseini, Walid Ahmed, and Yang Liu. 2023. Training acceleration of low-rank decomposed networks using sequential freezing and rank quantization. *arXiv preprint arXiv:2309.03824*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, and 1 others. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. 2024. More agents is all you need. *arXiv preprint arXiv:2402.05120*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Yan Liu, Renren Jin, Ling Shi, Zheng Yao, and Deyi Xiong. 2024b. Finemath: A fine-grained mathematical evaluation benchmark for chinese large language models. *arXiv preprint arXiv:2403.07747*.

Mohammad Mahdi Moradi, Hossam Amer, Sudhir Mudur, Weiwei Zhang, Yang Liu, and Walid Ahmed. 2025. Continuous self-improvement of large language models by test-time training with verifier-driven sample selection. *arXiv preprint arXiv:2505.19475*.

Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. 2019. Federated learning for emoji prediction in a mobile keyboard. *arXiv preprint arXiv:1906.04329*.

Nikhil Sardana, Jacob Portes, Sasha Doubov, and Jonathan Frankle. 2023. Beyond chinchilla-optimal: Accounting for inference in language model scaling laws. *arXiv preprint arXiv:2401.00448*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2020. Energy and policy considerations for modern deep learning research. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13693–13696.

Youssef Tawfilis, Hossam Amer, Minar El-Aasser, and Tallal Elshabrawy. 2025. A distributed generative ai approach for heterogeneous multi-domain environments under data sharing constraints. *arXiv preprint arXiv:2507.12979*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Preprint*, arXiv:1706.03762.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model. *Preprint*, arXiv:2401.02385.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

A Derivation: Break-even Condition for Test-Time Compute (TTC) with Continuous Refreshing

Define the following quantities per unit time (e.g., per year):

- $\mathrm{FLOPs}_{\text{train}}$: training FLOPs for the baseline (no TTC).
- $\mathrm{FLOPs}_{\text{infer}}$: inference FLOPs for the baseline (no TTC).
- $r \in (0, 1]$: ratio of training FLOPs under TTC to baseline training FLOPs, i.e. $r = \frac{\mathrm{FLOPs}_{\text{train}}^{\text{TTC}}}{\mathrm{FLOPs}_{\text{train}}}$.
- $\lambda_{\text{TTC}} > 0$: inference cost multiplier when using TTC (per-token inference cost relative to the "no TTC" baseline).
- $\lambda_{\text{base}} \geq 1$: inference cost multiplier already present in the baseline deployment (equals 1 when the baseline has no extra inference overhead).
- $f > 0$: refresh frequency (number of training+deployment cycles per unit time). For example, if the model is retrained and deployed three times per year, then f =3. For the algebra below f cancels and therefore does not affect the bound.

Total FLOPs per unit time for the baseline and for TTC-aware operation are

$$C_{\text{baseline}} = f \cdot (\mathrm{FLOPs}_{\text{train}} + \lambda_{\text{base}}\, \mathrm{FLOPs}_{\text{infer}}), \tag{6}$$

$$C_{\text{TTC}} = f \cdot (r\, \mathrm{FLOPs}_{\text{train}} + \lambda_{\text{TTC}}\, \mathrm{FLOPs}_{\text{infer}}). \tag{7}$$

Here we assume that we can train and/or deploy the underlying model with TTC controlled by λ_{base} or λ_{TTC}.
Requiring TTC to be no more expensive than the baseline gives

$$C_{\text{TTC}} \leq C_{\text{baseline}}.$$

Cancel f (assuming $f > 0$):

$$r\, \mathrm{FLOPs}_{\text{train}} + \lambda_{\text{TTC}}\, \mathrm{FLOPs}_{\text{infer}} \leq \mathrm{FLOPs}_{\text{train}} + \lambda_{\text{base}} \mathrm{FLOPs}_{\text{infer}}$$

Thus the FLOPs-level break-even condition is

$$\boxed{(1 - r)\, \mathrm{FLOPs}_{\text{train}} \;\geq\; (\lambda_{\text{TTC}} - \lambda_{\text{base}})\, \mathrm{FLOPs}_{\text{infer}}.}$$

Next express the bound in terms of token counts. Let N_{train} be the number of training tokens per refresh (baseline) and N_{infer} the number of inference tokens served before the next refresh. Using the common approximation that one training token costs roughly $6\times$ the FLOPs of one inference token (accounting for forward + backward + optimizer) (Kaplan et al., 2020; Anil et al., 2023),

$$\frac{\text{FLOPs}_{\text{train}}}{\text{FLOPs}_{\text{infer}}} \approx \frac{6\,N_{\text{train}}}{N_{\text{infer}}}.$$

Substitute this ratio into the FLOPs inequality:

$$\begin{aligned}(1-r)\cdot\frac{6\,N_{\text{train}}}{N_{\text{infer}}}\cdot\text{FLOPs}_{\text{infer}}\\ \geq(\lambda_{\text{TTC}}-\lambda_{\text{base}})\,\text{FLOPs}_{\text{infer}}\end{aligned}$$

Cancel $\text{FLOPs}_{\text{infer}} > 0$ and solve for N_{infer}:

$$(1-r)\cdot 6\,N_{\text{train}} \;\geq\; (\lambda_{\text{TTC}}-\lambda_{\text{base}})\,N_{\text{infer}}$$

$$N_{\text{infer}} \;\leq\; \frac{6\,(1-r)}{\lambda_{\text{TTC}}-\lambda_{\text{base}}}\,N_{\text{train}}.$$

Setting $\lambda_{\text{base}} = 1$ recovers the special case where the baseline has no extra inference overhead:

$$\boxed{N_{\text{infer}} \leq \frac{6(1-r)}{(\lambda-1)}\cdot N_{\text{train}}}$$

Remarks:

- The denominator $\lambda_{\text{TTC}} - \lambda_{\text{base}}$ must be positive for the bound to be meaningful. If $\lambda_{\text{TTC}} \leq \lambda_{\text{base}}$, the accuracy of the intermediate checkpoint with TTC will not be better than the fully trained checkpoint.

- For $\lambda_{\text{base}} > 1$, the denominator $(\lambda_{\text{TTC}} - \lambda_{\text{base}})$ decreases, which increases the upper bound on N_{infer}. This means that as the baseline incurs more inference overhead, TTC-aware training allows proportionally more inference tokens before the break-even point is reached. For this reason, we set λ_{base} to 1 to minimize the overhead of the baseline.

- This bound gives the maximum number of inference tokens that can be served between refreshes while keeping the total FLOPs (training plus deployment) at or below the non-TTC baseline.

Table 4 reports the number of inference examples permitted under our bound, given a model output sequence length of 1024 and a total of 3T training tokens. These values are shown for illustrative purposes; for instance, when $\lambda \approx 1.2\times$, inference with TTC can be applied for up to 70B examples before requiring the next model refresh.

r	λ	N_{infer}	Inference Examples
0.4	8	1.54T	1.5B
0.2	16	0.96T	937M
0.4	1.2	54T	52.7B
0.2	1.2	72T	70.1B

Table 4: Values for r, λ, N_{infer}, and inference samples.

B Fitting for the Final Accuracy and TTC

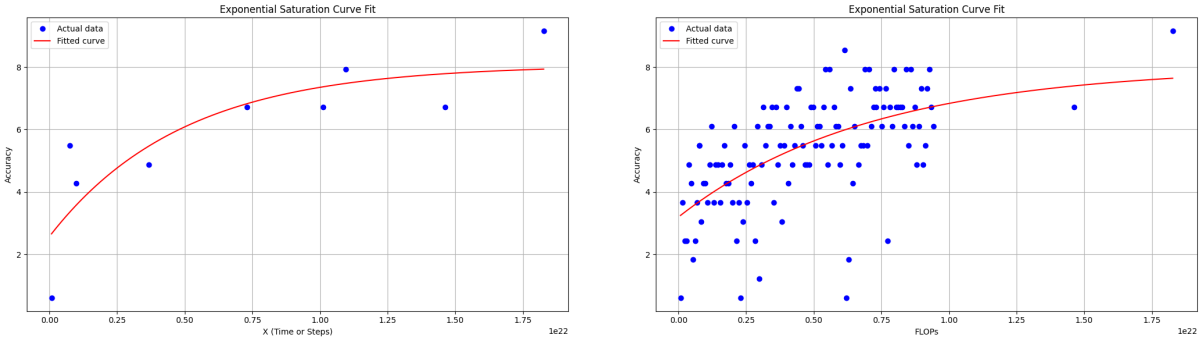
Figures 6 and 7 show the successful fitting of for the exponential saturating function and sigmoid functions, respectively.

C TinyLlama on Math-500

In this appendix, we evaluate TinyLlama on the Math-500 benchmark, a standard dataset for mathematical reasoning tasks. Math-500 is particularly challenging for TinyLlama because the model was not originally trained on sufficient math-domain data, resulting in a 0% score for the original fully trained checkpoint. To address this, we conducted additional training to include enough math data, despite our limited compute resources. Table 5 details the full training configuration.

Due to these compute constraints, we were only able to use a subset of intermediate checkpoints. Nevertheless, even with this partial coverage, the baseline without TTC shows a roughly monotonic increase across available checkpoints, albeit with some variation in scores. As shown in Figure 8, the results reveal two clusters of points: one corresponding to checkpoints trained from scratch on cosmopedia dataset with varying amounts of FLOPs appearing on the left, and another corresponding to fine-tuned checkpoints with varying amount of FLOPs appearing on the right.

Despite these practical limitations, the TTC-aware insight remains visible. This demonstrates both the promise of TTC for improving training efficiency and the challenges in fully exploiting it across models and checkpoints, highlighting the need for appropriate training resources and setup in future work.



(a) Curve fit using a small set of checkpoints.

(b) Curve fit across all checkpoints.

Figure 6: Exponential saturation curve fitting results. (a) shows the fitted curve using a small number of checkpoints, while (b) shows the fit when using all available checkpoints. In both cases, the red curve represents the fitted function and the blue points denote actual data, where each point corresponds to a TinyLlama checkpoint evaluated on the HumanEval dataset.

D TTC algorithm

This appendix shows the details of the TTC pseudo code as shown in Algorithm 1.

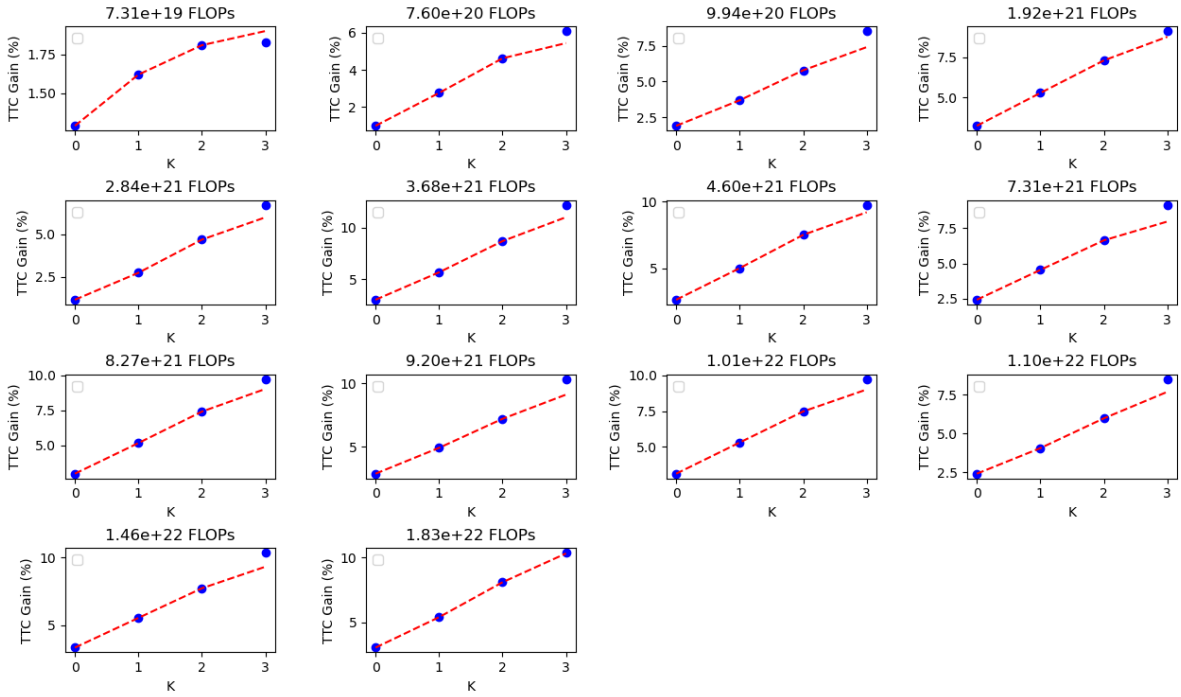


Figure 7: Sigmoid fitting for the parameter K in TTC at different FLOPs. TTC gain is recorded for $K = 1, 2, 4$ and the fitted curve is used to estimate accuracy at $K = 8$ on TinyLlama-1B with the HumanEval dataset.

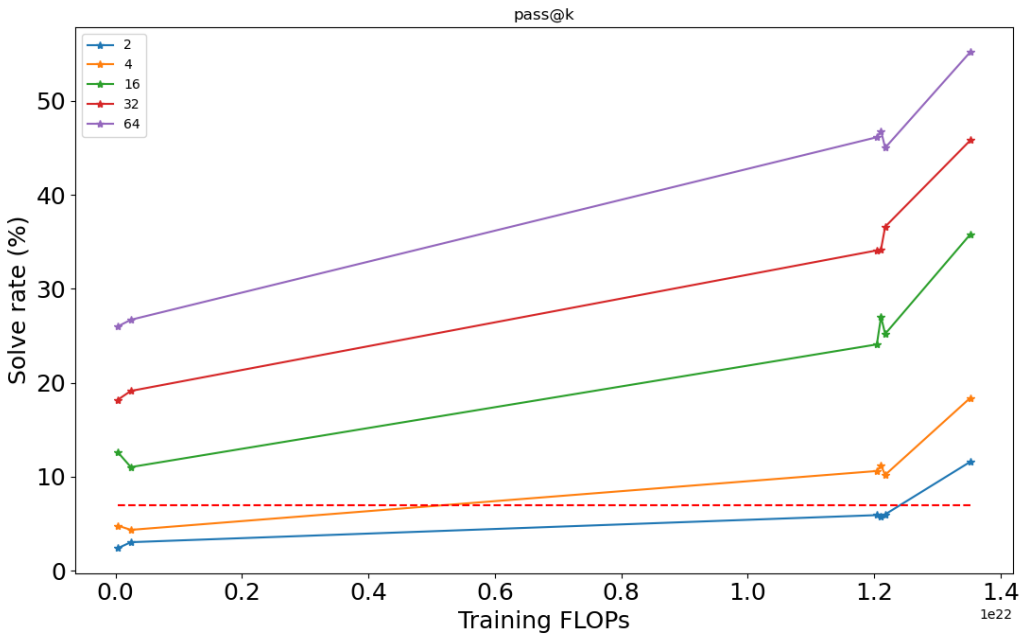


Figure 8: TinyLlama on Math-500 dataset with different Pass@K. The dotted horizontal line represents the score of the fully trained TinyLlama without TTC. Challenging setup given the TinyLlama available checkpoints accuracy on Math-500 and our compute resources.

Algorithm 1 TTC Aware Exponential Function Early Stopping

Input: Patience p, Training budget B
Initialize $best_val_acc \leftarrow -\infty$
Initialize $best_ttc_val_acc \leftarrow -\infty$
Initialize $patience_counter \leftarrow 0$
Define accuracy fitting function $f(x) = a \cdot (1 - \exp(-b \cdot x)) + c$
for each FLOPs $t = 3, 4, \ldots$ **until** $t > B$ **do**
  Train model on training data
  Evaluate model on validation data
  Compute val_acc_t
  Fit function $f(x)$ using observed FLOPs and val_acc_t
  Estimate final accuracy at FLOPs budget B using $f(B)$
  Estimate the minimum TTC K^* such that:

$$\text{FLOPs}_{train}[t] + \text{FLOPs}_{inference}[t, K^*] < \text{FLOPs}_{train}[B] + \text{FLOPs}_{inference}[B, 1]$$

  AND

$$val_ttc_acc_t[\text{K*}] >= f(B)$$

  Let $val_ttc_acc_t \leftarrow$ validation accuracy of model at K^*
  Update $best_ttc_val_acc_t \leftarrow \max(best_ttc_val_acc_t, val_ttc_acc_t)$
  if $best_val_ttc_acc_t < f(B)$ **then**
    $patience_counter \leftarrow 0$
  else
    $patience_counter \leftarrow patience_counter + 1$
  end if
  if $patience_counter \geq p$ **then**
    Stop training
    Load best TTC model checkpoint
    break
  end if
end for

Table 5: Pretraining configuration for TinyLLaMA in LLaMA-Factory (Zheng et al., 2024).

Category	Configuration
Model	quantization_method: None train_from_scratch: True disable_gradient_checkpointing: False
Method	stage: pt do_train: True finetuning_type: full deepspeed: examples/deepspeed/ds_z3_config.json
Dataset	dataset: cosmopedia-v2, From HuggingFaceTB/smollm-corpus cutoff_len: 2048 preprocessing_num_workers: 120 streaming: False, packing: True overwrite_cache: True dataloader_num_workers: 8
Training	per_device_train_batch_size: 16 gradient_accumulation_steps: 4 learning_rate: 2.0e-5 lr_scheduler_type: cosine warmup_ratio: 0.005 fp16: True, bf16: False ddp_timeout: 180000000 num_train_epochs: 1
Evaluation	val_size: 5000 per_device_eval_batch_size: 1 eval_strategy: steps eval_steps: 200
Output	logging_steps: 1, save_steps: 200 plot_loss: True overwrite_output_dir: True save_total_limit: 500 report_to: tensorboard