arXiv:2601.01332v1 [cs.CL] 4 Jan 2026

FLOP 高效训练：基于测试时计算感知的提前停止

Hossam Amer[1]*, Maryam Dialameh[1, 2]*, Hossein Rajabzadeh[1, 2]*,
Walid Ahmed[1],Weiwei Zhang[1],Yang Liu[1],

[1]昇腾团队、华为技术有限公司、多伦多、加拿大[2]滑铁卢大学、滑铁卢、加拿大**通讯作者：**hossam.amer1@huawei.com

摘要

长期以来，研究表明，扩大以浮点运算次数衡量的训练计算量能够提高大语言模型的准确率，但训练仍然需要大量资源。先前研究显示，增加测试时计算（TTC）——例如通过迭代采样——可以让较小的模型以更低的总成本媲美甚至超越大得多的模型。我们提出了测试时计算感知训练：通过结合中间检查点和相应的测试时计算配置，其准确率可以达到或超过全量训练模型，同时大幅减少训练 FLOPs。基于这一发现，我们提出一种早停算法，联合选择检查点和测试时计算配置，在不牺牲准确率的前提下尽量降低训练计算量。为使这一方法切实可行，我们开发了一种高效的测试时计算评估方法，避免穷举搜索，并形式化给出盈亏平衡界限，用以确定何时增加推理计算量能够弥补训练计算量的减少。实验表明，在保持准确率、甚至显著提高准确率的同时，训练 FLOPs 最多可减少 92%。这些结果为模型开发中平衡训练与推理计算量提供了新视角，有助于加快部署周期并更频繁地更新模型。代码将公开发布。

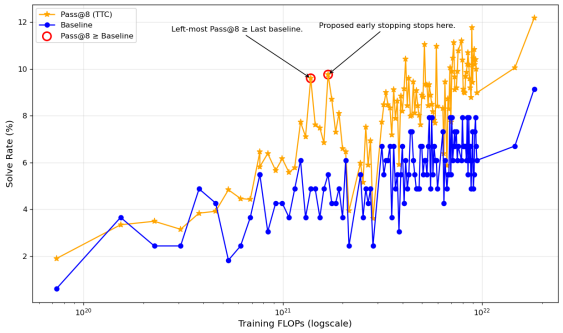


图1：TinyLlama 在 HumanEval 上的解题率（%）与训练 FLOPs 的关系。实线蓝线表示不使用 TTC 的基线检查点，实线橙线表示使用 TTC（Pass@8）的检查点。红色圆圈表示，TTC 能够在训练 FLOPs 显著减少的情况下，达到至少等同于不使用 TTC 的完全训练检查点的准确率。其他取值的讨论和结果 K（例如 Pass@4）见第 4 节。根据我们针对 K 的测试时计算估算方法，预测值为 $K^* = 8$。

1 引言

大语言模型（LLM）推动了自然语言处理及其他领域的显著进步（Vaswani 等人，2017；Achiam 等人，2023；Liu 等人，2024a）。Llama 和 Mistral 等开源架构在各类任务中展现出强大的文本理解与生成能力。然而，这些性能提升代价高昂：训练最先进的大语言模型需要海量计算资源和大型数据集，并造成沉重的财务与环境负担（Strubell 等人，2020年；Hajimolaho-seini 等人，2023；Tawfilis 等人，2025）。例如，训练 Llama-3.1-405B 在 NVIDIA H100 硬件上需要 3084 万个 GPU 小时，而持续预训练或微调每十亿词元的消耗超过 2,000 个 H100 GPU 小时。通常，只有大型实验室和公司才能获得这些资源。这些挑战凸显了人们日益迫切的需求：寻找其他策略，在不牺牲准确率的前提下提高效率（AI，2024）。

大语言模型的发展一直受到缩放定律的指导，这些定律描述了准确率如何随着模型规模、数据集规模和计算量的增加而提升。（Kaplan 等人，2020年）发现，训练损失会随着模型参数、训练词元和计算量的增加而按幂律下降。（Hoff- mann 等人，2022）随后提出了*Chinchilla* 范式，指出在固定计算预算下，通过平衡模型规模与训练词元，才能实现最优性能，从而对过度增大模型的趋势提出了挑战。

*共同贡献。

衡模型规模与训练词元，从而对过度增大模型的趋势提出了挑战。

尽管这些框架主要关注训练计算量，但近期研究已开始将推理时成本纳入考量（Moradi 等人，2025）。Sardana 等人，2023）提出了推理感知缩放，并证明在大规模部署场景下，训练时间更长的小模型能够实现更高的整体成本效益。（Snell 等人，2024）和（Wu 等人，2024）的实证研究表明，通过验证器引导搜索或迭代采样等方法增加测试时计算（TTC），可以让小模型以更低的总计算成本达到甚至超越大得多的模型的性能。关于重复采样和迭代优化的相关工作（Li 等人，2024；Brown 等人，2024；Chen 等人，2024）凸显了多轮推理在无需训练的情况下提升准确率的潜力。

尽管取得了这些进展，大多数研究仍只比较不同规模的全量训练模型，却忽视了单个模型内部的训练动态，尤其是性能如何随训练 FLOPs 的变化而演变。这一研究空白至关重要：若考虑 TTC，模型通过提前停止也能达到有竞争力甚至更优的性能。然而，现有的提前停止方法仍未考虑 TTC，只关注验证准确率，而忽视了推理阶段可能实现的准确率增益。同样，当前的 TTC 技术，尤其是基于重复采样的方法，会随着 K 的增加而提高推理成本，其中 K 表示迭代轮数（Pass@K）。这些技术并未考虑预测最优的 K，以便更高效地进行 TTC。

从 Chinchilla 最优训练到推理感知缩放，这些不断变化的范式指向一个统一原则：现代大语言模型的开发应联合优化训练和推理计算量。我们将这一原则进一步拓展，将*TTC 感知*融入训练，从而能够根据预测的 TTC 准确率做出提前停止决策。我们的方法在至少保持最终准确率的同时减少训练 FLOPs，而一种新的小型K 评估策略能够以极低的推理开销确定最佳 K。如图1 所示，TTC 感知提前停止以极少的计算量达到与未经 TTC 优化的完全训练检查点相当的效果，有时甚至更优，从而实现更快的模型部署和更频繁的模型更新。

总体而言，本文的贡献如下：1）揭示了*TTC 感知训练*这一洞见，即在保持或提升模型准确率的同时减少总浮点运算次数。2）提出一种 TTC 感知的早停算法，可在训练期间预测测试时计算准确率，并在实现有利的浮点运算次数—准确率权衡时停止训练。这种方法能够加快大规模模型开发中的模型更新和部署周期，也有益于计算资源有限的学术机构和开源实验室。3）开发了一种高效的测试时计算估算方法，可根据小规模K 运行预测最优 K，从而降低推理成本。4）提供盈亏平衡分析，确定 TTC 感知训练和推理在何种情况下总计算量能够匹配或优于传统训练。5）通过大量实验实证验证 TTC 感知训练的有效性。

2 方法：TTC 感知训练流程

图2 展示了所提出的融入 TTC 感知以实现 FLOP 高效训练的整体训练流程。在该流程中，训练超参数和训练 FLOPs 预算 B在训练开始前给定。目标是在总训练和 TTC 推理成本低于不采用 TTC 时的成本这一约束下，训练出一个在 TTC 推理下最终验证准确率最高的模型。基于 TTC 感知这一洞见，我们提出了一种 FLOP 高效训练框架，能够在预算 B内达到或超过完全训练检查点的验证性能。训练过程中，我们收集验证准确率 $\{\mathcal{A}_i\}_{i=1}^{n}$，这些数值来自选定的中间检查点 $\{t_i\}_{i=1}^{n}$，并利用它们构建曲线拟合预测，以估计预算$\hat{}$下的验证准确率 $\mathcal{A}(B)$。随着训练推进，观察到更多检查点后，我们会动态优化这一预测。

这一最终准确率预测依赖于经验观察：典型的学习曲线遵循饱和模式——准确率在初期快速提升，随后收益逐渐递减（Kaplan 等人，2020年）。因此，我们将收集到的验证准确率数据拟合到一个指数饱和函数：

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

其中，a 控制准确率增益的幅度，b 决定收敛速率（即模型达到饱和的速度），c 表示初始化时的基础准确率。这些参数通过学习获得

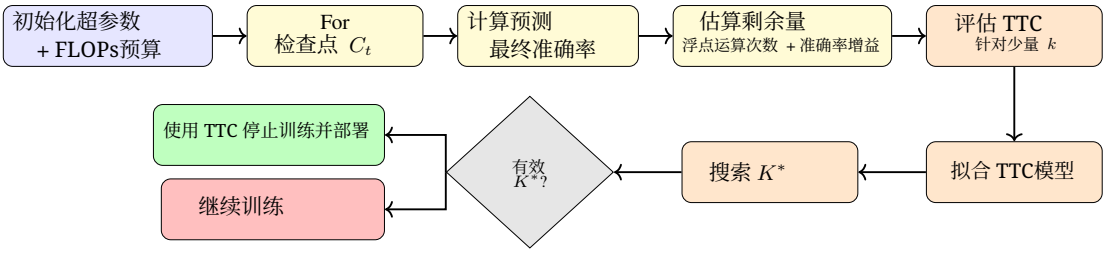


图 2：所提出的 TTC 感知训练流程图。该方法能够高效地选出一个在测试时计算配置下、在浮点运算次数和准确率方面优于全量训练检查点的中间检查点，从而实现提前停止。请参阅算法1 及附录E中的相关说明。

具体方法是最小化观测到的验证准确率与拟合曲线之间的均方误差。随后，利用拟合函数 $\hat{\mathcal{A}}(B)$ 预测预算 B 下的最终验证性能，从而指导提前停止和推理分配。

训练过程中，验证准确率通常会出现波动，这使得拟合 $\mathcal{A}(B)$变得复杂。为减轻此类波动带来的噪声影响，我们仅保留相对于迄今为止观测到的最佳准确率单调不减的验证点；具体而言，若某个点的验证准确率大于或等于此前观测到的最大值，则予以保留，否则舍弃。更一般地，可以通过在当前最大值附近引入用户定义的容差来放宽这一准则；不过，为了简化处理并避免引入额外的超参数，我们在所有实验中均采用严格的 $\geq$ 规则。随后，我们仅使用曲线上经过筛选、呈上升趋势的部分来拟合模型。因此，估计器不会引入超出底层训练过程本身固有波动之外的额外方差。这一设计选择体现了我们的目标：准确估计可达到的最高准确率，以用于作出提前停止决策（见图7和8）。

为支持这一框架实现高效推理，我们引入一种感知 TTC 的机制，用于估算每个查询的最佳推理样本数，记为 K^*，同时避免产生过多计算量。由于标准 TTC 通过对模型进行 K 次采样并汇总响应来执行推理，因此当 K 较大时，成本可能高得难以承受。为此，我们提出用 S 形曲线近似时间 t 时 TTC 下的验证准确率与样本数 K 之间的关系：

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

其中 L 表示 TTC 可达到的最高准确率，k 控制准确率随 K增加而上升的增长率，且 x_0 表示拐点，即准确率达到最高值一半的位置。这一关系符合直观预期：随着K 增加，准确率最终会达到饱和点，此后准确率增益趋于平缓（更多证据见附录中的图9 ）。

这些参数通过对少量采样值进行非线性最小二乘拟合来估计；这些值是通过在较小的 K 值处查询模型得到的，例如 $K = 1$，2，4。在 K处评估一次，就会隐式地产生所有 $K' \leq K$对应的预测，因此在 $K = 4$ 处查询即可免费获得 $K = 1$，2，4对应的有效结果。利用该公式，我们可以外推并预测 K^*——即同时满足预算和准确率约束的最小 K。最优的 K^* 必须满足总浮点运算次数约束：

$$\mathrm{F}_{\mathrm{tr}}[t] + \mathrm{F}_{\mathrm{inf}}[t, K^*] < \mathrm{F}_{\mathrm{tr}}[B] + \mathrm{F}_{\mathrm{inf}}[B, 1] \tag{3}$$

其中，$\mathrm{F}_{\mathrm{tr}}[t]$ 是截至时间步 t的训练成本，$\mathrm{F}_{\mathrm{inf}}[t, K^*]$ 是在该时间步使用 K^* 个样本进行 TTC 推理评估的成本。此外，K^*必须满足预期准确率要求：

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

也就是说，在时间步 t 使用 K^* 个样本时的测试时计算准确率，必须至少达到完整预算 B 下预期的最终验证准确率。若在经过包含 p个检查点的耐心窗口后存在这样的 K^*，则提前终止训练；否则，训练将继续进行至下一个检查点。

3 提前停止的计算开销与盈亏平衡界限

所提出的 TTC 感知训练流程采用提前停止，可能会引入额外的计算开销——

算法1TTC 感知指数拟合提前停止

1：输入：耐心值 p**，训练预算** B**；初始化：** $best\_val\_acc$、$best\_ttc\_val\_acc \leftarrow -\infty$、$patience\_counter \leftarrow 0$ **2：定义拟合函数** $f\ x\ = a\ \ + c\,(\,)\,(1 - e^{-bx})$**：对于** $t = 3\ 4\ \ldots\ \ t > B$ **FLOPs，，直到 do 4：训练一步；评估以获得** val_acc_t **5：使用观测到的 FLOPs 和** $\{val\_acc\}$ **拟合** $f(x)$**；估计基线最终准确率** $f(B)$ **6：找到满足以下条件的最小** K^***：** $F_{\mathrm{tr}}(t) + F_{\mathrm{inf}}(t, K^*) < F_{\mathrm{tr}}(B) + F_{\mathrm{inf}}(B, 1)$ **&** $\mathrm{Acc}_t^{\mathrm{ttc}}(K^*) \geq f(B)$ **7** $patience\_counter \leftarrow 0\quad val\_ttc\_acc_t[K^*] > best\_ttc\_val\_acc$**：更新 if：max(,)** **8** $best\_ttc\_val\_acc \leftarrow\ \ best\_ttc\_val\_acc\ val\_ttc\_acc_t[K^*]$ **9** $patience\_counter \leftarrow 0\quad best\_ttc\_val\_acc < f\ B\quad patience\_counter{+}{+}$**：更新 if ( )** **10** **else：if** $patience\_counter \geq p$ **then 11：停止；加载最佳 TTC 检查点；break 12：end if 13：end for**

，因为在训练和部署期间都会定期进行 TTC 驱动的推理。因此，确定何时 这些额外成本会被 TTC 感知提前停止带来的训练节省抵消，是很有用的。我们将此称为盈亏平衡点：在“采用 TTC 感知训练 + 部署”的总成本超过不采用 TTC 训练同一模型的成本之前，所能进行的下游推理量上限。

这里，总成本明确包括训练成本（其中可能已经计入 TTC 驱动的推理）和部署成本，即为响应终端用户查询所需的浮点运算次数或词元数。现代大语言模型并非训练一次后便永久冻结，而是会通过预训练、微调和持续更新而不断刷新 （Ramaswamy 等人，2019；Gamal 等人，2023；Amer 等人，2024）。因此，刻画这一界限尤为重要：它指明模型应当刷新之前可以消耗的推理令牌数量。

我们用 N_{train} 表示不使用 TTC 的基线计划中的训练 词元数，用 N_{infer} 表示推理 词元数。这里， N_{infer}涵盖所有推理词元，包括 TTC 采样期间使用的词元以及提供给终端用户的词元。使用 TTC 时的训练浮点运算次数相对于基线的比率为 $r \in (0, 1]$,，其中 $r < 1$ 表示提前停止所节省的训练量。使用 TTC 提供模型服务的推理成本相对于不使用 TTC 的基线模型，由乘数 $\lambda > 1$表示。假设每个词元的训练浮点运算次数约为每个词元推理浮点运算次数的 $6\times$ （计入前向传播 +反向传播和优化器更新），则盈亏平衡条件——即在确保 TTC 的总浮点运算次数不超过不使用 TTC 的基线总浮点运算次数的前提下，最多可提供的推理词元数——为：

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

详细推导和讨论见附录 A。

作为说明，表1报告了在我们的界限下允许的推理样本数量，设模型输出序列长度为 1024，训练词元总计为 3T。当$\lambda \approx 1.2\times$时，在需要进行下一次模型刷新之前，采用 TTC 进行推理最多可处理 70B 个样本。这也进一步说明了设计更高效的 TTC推理技术的重要性，我们将其列为未来工作。

r	λ	N_{infer}	推理样本
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	70.3B

表1： r、 λ、 N_{infer} 的示例值，以及推理样本；每个样本包含 1024 个词元。

4 实验结果

4.1 实验设置

模型与检查点 我们使用三个开源模型系列评估所提出的方法：TinyLlama（Zhang 等人，2024）、Pythia（Biderman 等人，2023）、FineMath（Liu 等人，2024b）和 Qwen3（Yang 等人，2025）。选择这些模型是因为它们提供了公开可用的中间检查点；而对于大多数大规模模型，复现训练的成本高得令人难以承受。表 2总结了接受评估的模型。我们使用 TinyLlama 和 Pythia 评估 TTC 感知训练在预训练设置中的影响，而 FineMath 则用于在持续训练场景中进行评估。

数据集 我们在涵盖代码生成、阅读理解和数学推理的多样化基准测试集上评估模型。

	TinyLlama	Pythia	FineMath	Qwen3-A3B-Instruct
规模（十亿参数）	1.1	1 / 2.8 / 6.9	3	30
架构基座	LLaMA-2（分组查询注意力）	GPT风格	LLaMA-3	Qwen3
训练数据	SlimPajama StarCoder	The Pile	FineWeb-Edu FineMath	开放数据
词元	∼2T	300B	160B	36T
优化器	AdamW	Adam + ZeRO	AdamW	–

表 2：我们实验中使用的模型。

具体而言，我们纳入了 HumanEval（Li 等人，2022），这是用于评估基于自然语言提示的代码合成的标准基准测试；DROP（Dua 等人，2019），用于测试对篇章的离散推理和阅读理解；Math-500（Hendrycks 等人，2021），一组经过筛选的高等数学题；GSM8k（Cobbe 等人，2021），一组小学数学应用题，旨在评估多步算术推理能力。

推理硬件与环境 推理实验在一台配备 8 块 NVIDIA Tesla V100 GPU 的设备上进行。我们启用了 Float16 推理，因为它能在保持准确率的同时加快推理速度。我们的推理环境基于 InstructEval 基准测试构建（Chia 等人，2023）。

TTC 超参数设置 我们的实验使用 0.8 的温度、10 的耐心值、最大 1,024 个词元的输入序列长度，以及最大 512 个词元的输出长度。其余配置借鉴自 InstructE-val 仓库（Chia 等人，2023）。

4.2 TTC 感知训练洞见

TTC 感知训练使模型能够以显著更少的训练 FLOPs 或更少的数据达到相同或更高的准确率。本节将介绍相关实验，以展示这一洞见。

TinyLlama 在 HumanEval 和 DROP 上的表现：如图 1所示，与不使用 TTC 的完全训练基线相比，中间检查点的准确率至少相同，甚至更高，同时最多可节省 92.44% 的训练浮点运算量。具体而言，该检查点的准确率提升了 0.45%。计入训练期间 TTC 推理额外消耗的浮点运算次数后，总节省仍高达 85.5%。尽管采用 Pass@8 的最终检查点准确率更高，但与中间检查点相比，这一提升需要付出大幅增加 FLOPs 预算的代价。为进一步验证这一发现，图10 显示，DROP 上也呈现出相同趋势：在训练预算小得多的情况下，较早的检查点也能达到相当或更高的准确率。这证实了 TTC 感知训练现象并非仅限于单一数据集，而是能够泛化到不同任务。

TinyLlama 在 HumanEval 上不同 Pass@K 指标的表现： 在图 3中，我们观察到三个一致的趋势。首先，总是存在一个中间检查点，其 Pass@K 性能至少与不使用 TTC 的完全训练基线相当，同时所需的浮点运算次数显著更少。其次，比较不同的 K 值时，如果中间检查点采用的更大 K 高于完全训练的模型（例如，Pass@8 或 Pass@16 相较于 Pass@4），则通常能够在节省浮点运算量的同时，保持甚至超过完全训练基线的准确率。在 TTC 下，如果第一个检查点采用的 K 小于完全训练模型所采用的值，其准确率就会较低。若要达到相当的性能，训练就必须继续进行，直至用完完整预算。第三，在相同的 K下比较两个模型时，完全训练的基线准确率更高，但代价是需要大得多的 FLOPs 预算。

Pythia 在 HumanEval、GSM8K 上的表现：对于每个 Pythia 1B/2.8B/6.9B 模型，我们将训练步数为 80k 的中间检查点与训练步数为 143k 的完全训练基线进行比较（见表3）。在 HumanEval 和 GSM8k 上，我们都观察到与此前在 TinyLlama 上发现的相同 TTC感知趋势。使用 TTC 的中间检查点，其性能可与完全训练的基线相当或更优，同时所需的浮点运算次数大幅减少。例如，在 HumanEval 上，训练至 80k 步的 1B Pythia 模型取得了 3.8% 的 Pass@8准确率，与其不使用 TTC、训练至 143k 步的基线（3.04%）相当；而训练至 80k 步、Pass@8 为 8.84% 的 2.8B 模型，不仅超过其训练至 143k 步的基线（6.7%），还优于规模大得多、训练至 143k 步的 6.9B 基线（6.8%）。

GSM8k 上也呈现出类似的模式：80k 检查点的 2.8B 模型在所有 Pass@K值上都达到或超过了 143k 基线，而带有 TTC 的 6.9B 模型在 80k 检查点上的表现优于不带 TTC 的完全训练版本。此外，在相同模型规模下比较不同 K 值时，如果中间检查点使用的 K 大于完全训练的基线，其准确率通常更高。例如，带有 Pass@8（11.29%）的 Pythia-2.8B 80k 检查点，优于带有 Pass@4（6.3%）的 Pythia-2.8B 143k 检查点。在不同 Pythia 模型规模之间也观察到了相一致的模 式，

模型	数据集	Step	词元	浮点运算次数	基线	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

表3：不同规模的 Pythia 模型在 HumanEval 和 GSM8K 上的中间（80k）与最终（143k）检查点对比。结果使用 Pass@K 指标报告。**粗体**条目表示中间 TTC 感知检查点在相同或更高的 K下达到或超过完全训练的基线，凸显了 TTC 感知训练的一致性和优势。

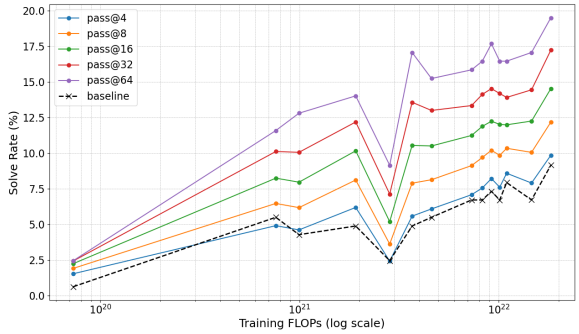


图 3：TinyLlama 在 HumanEval 上的 Pass@K 表现。中间检查点使用少得多的浮点运算次数即可匹配或超过完全训练的基线，有时还能在更高的 K 下胜过基线；而最终检查点只有在付出大幅增加浮点运算次数的代价后才表现更好。

这进一步印证了从 TinyLlama 中得出的结论：TTC 感知效应可推广至不同模型规模和数据集，表明中间检 查点只需付出一小部分训练成本，就能达到相当甚至更好的性能。

使用实际 TTC 技术的 FineMath：在之前的所有实验中，我们都聚焦于 Pass@K，它反映了模型在采样多个输出时的潜在性能。为了将这一研究扩展到 Pass@K之外更先进的 TTC 技术，我们评估了几种方法：(1) 不使用 TTC 的基线，(2) Pass@K，(3) 多数投票，(4) 使用朴素加权的多样化树搜索（DVTS），(5) 使用加权聚合的 DVTS，以及 (6) 计算最优搜索，它会从多数投票和两种 DVTS 变体中选择性能最佳的方法（Beeching 等人，2024；Snell 等人，2024）。
在采用加权聚合的 DVTS 中，相同的回答会被归为一组，并根据所有相同回答的累积奖励选择最终答案。相比之下，采用朴素加权的 DVTS 会独立处理每个生成的回答，并选择奖励模型（RM）评分最高的回答作为最终输出。

我们在数学领域的持续训练设置中，对 FineMath 模型使用 10B、30B、40B 和 50B 个训练词元应用这些技术，并在 Math-500 基准测试上进行评估。表4 中的一个重要观察是，即使较低的训练预算在某个 Pass@K下表现不佳，也往往存在一个更大的 K，使其超越在较小 K下评估的高预算检查点。例如，在 K=2 时，10B 检查点的表现不如 50B 检查点（15.05% 对 18.28%）；但在 K=4时，10B 检查点的表现优于 50B 检查点，达到 K=2 （24.73% 对 18.28 %）。这一规律在多种设置中反复出现（例如，10B 对比 40B），体现了 FineMath 所展示的持续训练场景中更广泛的 TTC 感知洞见。

我们进一步发现，这一 TTC 感知洞见并不局限于 Pass@K，也适用于基于验证器的指标和聚合指标。例如，在 DVTS-WEIGHTED 验证器下，使用 K=16 的 10B 检查点解题率达到 19.35%，超过了未使用 TTC 的 50B 检查点（15.05%）。同样，50B 检查点的解题率从 K=2 时的 13.98% 提升至 K=4 时的 21.51%，表明随着 K 增大，性能持续提升。若要达到或超过 50B 检查点的准确率，也可以选择使用 K=32 的 10B 检查点（33.33%）、使用 K=16 的 30B 检查点（24.73%），或使用 K=32 的 40B 检查点（41.94%）。在评估的所有 TTC 技术中，我们观察到解题率通常会随

着 K 增大而提升。不过，基于验证器的测试时计算分数通常低于原始 Pass@K 值，这凸显了设计强大验证器对于真实评估的重要性。在各项 TTC 技术中，DVTS-WEIGHTED 的整体性能始终最佳，而计算最优搜索则为已测试的方法提供了一个近似上界。这些发现进一步证实，即使采用基于验证器的指标和聚合指标，TTC 感知洞见依然成立，从而进一步验证了测试时计算框架在不同指标类型和模型规模上的普适性。

TinyLlama 在 Math-500 上我们在 Math-500 基准测试上评估 TinyLlama。Math-500 是用于数学推理任务的标准数据集。由于 TinyLlama 最初并未使用足量的数学领域数据进行训练，因此该基准测试对它而言尤其具有挑战性，原始完全训练检查点的得分为 0%。为解决这一问题，尽管计算资源有限，我们仍进行了额外训练，以纳入足量的数学数据。表7详细列出了完整的训练配置。

受计算资源限制，我们只能使用部分中间检查点。尽管覆盖范围有限，不使用 TTC 的基线在可用检查点上的表现仍大致呈单调增长，但得分有所波动。如图4所示，结果呈现出两组点：一组位于左侧，对应于使用不同浮点运算次数、基于 Cosmopedia 数据集从头训练的检查点；另一组位于右侧，对应于使用不同浮点运算次数的微调检查点。

尽管存在这些实际限制，基于 TTC 的洞见依然清晰可见。这既展现了 TTC 提升训练效率的潜力，也揭示了在不同模型与检查点上充分发挥其作用所面临的挑战，凸显了未来工作中配备合适训练资源并做好设置的必要性。

前沿大型模型上的结果：虽然在前沿规模大语言模型上评估我们的方法受到中间检查点可用性的限制，但我们创建了如下设置：我们在 GSM8K 上对 Qwen3-30B-A3B-Instruct 进行了 1 个训练轮次的微调，在训练过程中保存了中间检查点，并在这些检查点中应用了 TTC 选择。如图5所示，即使在这种大模型设置下，我们也观察到了相同的 TTC 感知效应：

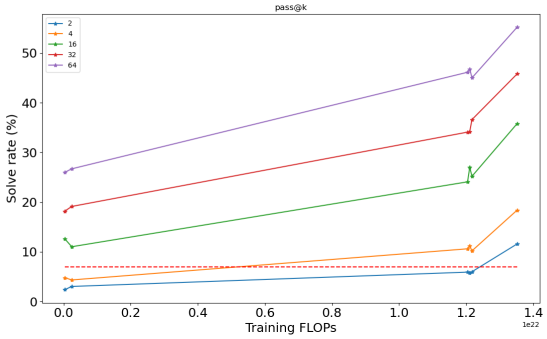


图4：TinyLlama 在 Math-500 数据集上不同 Pass@K 的表现。虚线表示未经 TTC 的 TinyLlama 完成全量训练后的得分。鉴于 TinyLlama 可用检查点在 Math-500 上的准确率以及我们的计算资源，这一设置颇具挑战性。

通过 Pass@2 选择的较早 TTC 中间检查点，其表现可比最终检查点高出 10% 以上，这表明 TTC 在小模型规模之外仍然有益。

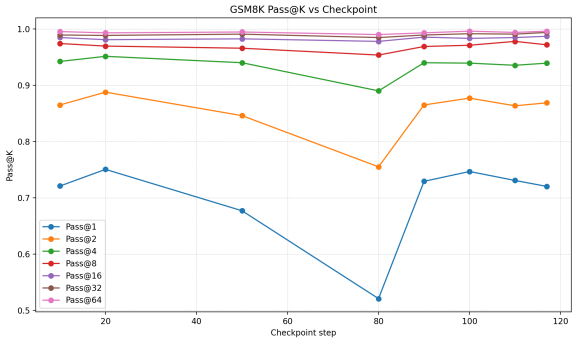


图5：Qwen3-30B-A3B-Instruct 在 GSM8K 上（1 个训练轮次）的表现。中间检查点的 Pass@K 呈现非单调变化；TTC 选择的中间检查点优于最终检查点。

4.3 提前停止分析与结果

如表5所示，所提出的早停算法取得了显著提升。在 TinyLlama 的 HumanEval 数据集上，其 Pass@8 准确率比不使用 TTC 的完全训练检查点高约 0.6%，同时节省了 90.7% 的训练浮点运算量；Pass@16 则提升了 4.3%，同时节省了 90.76% 的训练浮点运算量。采用我们的提前停止方法，Pass@8 在相较于基线的情况下减少了 90.7% 的训练 FLOPs，而总体上实际可节省 92.44% 的浮点运算量。不同数据集和模型系列也呈现出类似趋势，包括 Pythia（Biderman 等人，2023）。总体而言，TTC 感知训练最多可节省 92% 的训练 FLOPs，同时不

训练令牌数	K	基线 Pass@K	多数投票	朴素加权	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

表 4：不同 K 下 *FineMath* 检查点（10B、30B、40B、50B 词元）的解题率（%）。验证器为 Skywork-o1-Open-PRM-Qwen-2.5-1.5B。我们报告了在 Math-500 上采用不使用 TTC 的基线、Pass@k、多数投票、DVTSNaive、DVTS 加权和计算最优搜索（CTS）聚合方法的结果。

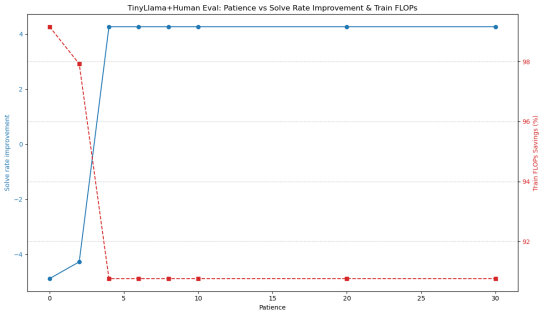


图 6：耐心值对 TinyLlama（HumanEval）的影响。耐心值越大，测试时计算 Pass@16 越高，但训练 FLOPs 也越多；中等范围的值（$\approx$ 10）能实现最佳权衡，而耐心值过大则会削弱提前停止的益处。

牺牲准确率，且往往能提升准确率。

为进一步评估所提出的提前停止算法，我们实现了 PyTorch 风格的朴素早停，并在 TinyLlama–HumanEval（PyTorch-Ignite, 2025）上进行评估。我们将耐心值从 0 调整到 100，使用 $\epsilon = 0$。我们报告 Δ，即早停模型与全量训练基线之间的准确率差异；FLOPs 节省，即节省的训练计算量占比；以及 Δ_{TTC}，即对早停检查点应用 TTC Pass@8 后的准确率差异。如表6所示，朴素早停的权衡效果较差：要么很早就停止，虽然节省了大量浮点运算次数，但会导致准确率大幅下降；要么接近全量训练后才停止，节省的计算量微乎其微。事后应用 TTC 并不能显著改善这种权衡。总体而言，朴素早停不如所提出的方法，后者最多可节省 92% 的浮点运算次数，同时将准确率提高 0.44%。

表 5：验证波动过滤对 TinyLlama 提前停止的影响。

	HumanEval		DROP	
方法	FLOPs 节省	准确率提升	FLOPs 节省	准确率提升
带波动的提前停止	56.8%	+2.05%	72.7%	+2.08%
不带波动的提前停止	90.7%	+0.6%	87.0%	+1.0%

表6：在 HumanEval 上，朴素早停与全量训练的 TinyLlama 检查点在准确率和计算成本（浮点运算次数）方面的相对比较。

耐心值	Δ (%)	FLOPs 节省(%)	Δ_{TTC} (%)
0	-4.27	97.92	-5.31
6	-3.66	95.80	-2.69
8	-3.05	93.28	-1.42
10	-3.05	93.28	-1.42
20	-1.22	70.24	0.19
50	0.00	0.00	3.03
100	0.00	0.00	3.03

5 结论

我们提出了*TTC* 感知训练，通过选择一个中间检查点并搭配最小测试时计算设置 K^*，以最大化每总浮点运算次数所获得的质量，而非默认使用预算为 B 的最终检查点。在多种模型规模和设置下，这一方法在不损失质量的情况下实现了显著节省：训练浮点运算次数减少最多**92.44%**，同时保持（且往往提升）下游性能。例如，配合 TTC 的 Pythia-2.8B**8 万步**检查点在 HumanEval 上达到**8.84%** 的 Pass@8，而完全训练的基线为**6.7%**；在 Math-500 上，基于验证器的 TTC 将成绩从**15.05%**（50B token 基线）提升至**19.35%**，对应的检查点仅训练了**10B token** K=16。我们的 TTC 感知提前停止策略进一步提升了这一结果：在 TinyLlama–HumanEval 上，相较于完全训练的检查点（不使用 TTC），Pass@8 提高约 **+0.6%**，同时节省**90.7%** 的训练浮点运算次数；此外，Pass@16 达到 **+4.3%**，浮点运算量节省**90.76%**。最后，我们的盈亏平衡界限表明，这些训练节省可用于为 ~**70B** 个下游样本提供 TTC（对于 $\lambda \approx 1.2\times$），因此 TTC 感知训练对频繁更新的模型尤其有吸引力。

局限性

模型与检查点。我们的研究仅涵盖参数量不超过 6.9B 的公开检查点。Llama-3.x-405B 等更大的模型并未发布中间检查点，因此无法直接评估。

数据集。 由于领域预训练和计算量有限，某些模型与数据集的组合（例如，TinyLlama 在 Math-500 上）表现不佳，限制了评估范围。

TTC 推理效率。 要在部署中实现 TTC 收益，可能需要高效的推理策略（例如束搜索、多样化搜索），以控制延迟和成本。整合此类优化仍是一个重要的研究方向。

基于验证器的评估。 我们主要在持续预训练中评估基于验证器的方法。未来值得在多个验证器和架构上开展更广泛的研究，但本文重点在于展示如何在保持准确率的同时减少浮点运算量。

缩放定律。 完整的 TTC 感知训练缩放定律——用于刻画最优停止点——仍有待研究，也是未来研究的一个重要方向。

参考文献

Josh Achiam、Steven Adler、Sandhini Agarwal、Lama Ahmad、Ilge Akkaya、Florencia Leoni Aleman、Diogo Almeida、Janko Altenschmidt、Sam Altman、Shyamal Anadkat及其他 1 位作者。2023。GPT-4 技术报告。*arXiv* 预印本 *arXiv:2303.08774*。

Meta AI。2024。Llama 3.1 405B。https://huggingface.co/meta-llama/Llama-3.1-405B。访问日期：2025-08-21。

Hossam Amer、Joe Osborne、Michael Zaki 和 Mohamed Afify。2024。通过基于 GPT 的数据增强训练的移动键盘设备端表情符号分类器。*arXiv* 预印本 *arXiv:2411.05031*。

Rohan Anil、Andrew M Dai、Orhan Firat、Melvin Johnson、Dmitry Lepikhin、Alexandre Passos、Siamak Shakeri、Emanuel Taropa、Paige Bailey、Zhifeng Chen及其他 1 位作者。2023。PaLM 2 技术报告。*arXiv* 预印本 *arXiv:2305.10403*。

Edward Beeching、Lewis Tunstall 和 Sasha Rush。2024 年。使用开放模型扩展测试时计算量。Hugging Face Space。https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute。

Stella Biderman、Hailey Schoelkopf、QuentinGregory Anthony、Herbie Bradley、Kyle O'Brien、Eric Hallahan、Mohammad Aflah Khan、Shivanshu Purohit、USVSN Sai Prashanth、Edward Raff 等另外 1 位作者。2023。《Pythia：用于分析训练和规模扩展过程中大语言模型的一套工具》。载于国际机器学习大会，第 2397–2430 页。PMLR。Bradley Brown、Jordan Juravsky、Ryan Ehrlich、Ronald Clark、Quoc V Le、Christopher Ré 和 Azalia Mirhoseini。2024。《大型语言模型猴子：通过重复采样扩展推理计算量》。*arXiv* 预印本 *arXiv:2407.21787*。Lingjiao Chen、Jared Quincy Davis、Boris Hanin、Peter Bailis、Ion Stoica、Matei A Zaharia 和 James Y Zou。2024。《更多大语言模型调用就是你所需要的一切吗？迈向复合人工智能系统的扩展特性》。神经信息处理系统进展，37：45767–45790。

Yew Ken Chia 等 1 人。2023 年。InstructEval：迈向对指令微调大语言模型的全面评估。*arXiv* 预印本 *arXiv:2306.04757*。

Karl Cobbe、Vineet Kosaraju、Mohammad Bavarian、Jacob Hilton、Reiichiro Nakano、Christopher Hesse 和 John Schulman。2021 年。训练验证器解决数学文字题。*arXiv* 预印本 *arXiv:2110.14168*。

Dheeru Dua、Yizhong Wang、Pradeep Dasigi、Gabriel Stanovsky、Sameer Singh 和 Matt Gardner。2019 年。DROP：一项要求对段落进行离散推理的阅读理解基准。收录于《*2019 年北美计算语言学协会会议论文集：人类语言技术*》（*NAACL-HLT*），第 2368–2378 页。计算语言学协会。

Karim Gamal、Ahmed Gaber 和 Hossam Amer。2023。基于联邦学习的多语言表情符号预测，涵盖干净场景和攻击场景。*arXiv* 预印本 *arXiv:2304.01005*。

Habib Hajimolahoseini、Walid Ahmed 和 Yang Liu。2023。利用顺序冻结和秩量化加速低秩分解网络的训练。*arXiv* 预印本 *arXiv:2309.03824*。

Dan Hendrycks、Collin Burns、Saurav Kadavath、Akul Arora、Steven Basart、Eric Tang、Dawn Song 和 Ja-cobSteinhardt。2021。使用 MATH 数据集衡量数学问题求解能力。*arXiv* 预印本 *arXiv:2103.03874*。

Jordan Hoffmann、Sebastian Borgeaud、Arthur Men-sch、Elena Buchatskaya、Trevor Cai、Eliza Ruther-ford、Diego de Las Casas、Lisa Anne Hendricks、Johannes Welbl、Aidan Clark 和另外 1 位作者。2022。训练计算最优的大语言模型。*arXiv* 预印本 *arXiv:2203.15556*。

Jared Kaplan、Sam McCandlish、Tom Henighan、TomB Brown、Benjamin Chess、Rewon Child、Scott Gray、Alec Radford、Jeffrey Wu 和 Dario Amodei。2020年。《神经语言模型的缩放定律》。*arXiv* 预印本 *arXiv:2001.08361*。

Junyou Li、Qin Zhang、Yangbin Yu、Qiang Fu 和 Deheng Ye。2024。《更多智能体就够了》。*arXiv* 预印本 *arXiv:2402.05120*。

X Li、DG Wang、S Wang、S Wang、Y Wang、Y Wang、Y Wang、Y Wang、Z Wang、Z Wang和其他1位作者。2022。评估基于代码训练的大语言模型。载于*2022*年自然语言处理实证方法会议论文集（*EMNLP*），第12345–12356页。

Aixin Liu、Bei Feng、Bing Xue、Bingxuan Wang、Bochao Wu、Chengda Lu、Chenggang Zhao、Chengqi Deng、Chenyu Zhang、Chong Ruan和其他1位作者。2024a。Deepseek-v3技术报告。*arXiv*预印本 *arXiv:2412.19437*。

Yan Liu、Renren Jin、Ling Shi、Zheng Yao和Deyi Xiong。2024b。FineMath：面向中文大语言模型的细粒度数学评估基准测试。*arXiv*预印本 *arXiv:2403.07747*。

Mohammad Mahdi Moradi、Hossam Amer、Sudhir Mudur、Weiwei Zhang、Yang Liu 和 Walid Ahmed。2025 年。通过采用验证器驱动的样本选择进行测试时训练，实现大语言模型的持续自我改进。*arXiv* 预印本 *arXiv:2505.19475*。PyTorch-Ignite。2025 年。提前停止——pytorch-ignite 文档。
h
t
t
p
s://
d
o
c
s.
pytorch.org/ignite/generated/ignite.handlers.early_stopping.EarlyStopping.html。访问日期：2025-12-16。Swaroop Ramaswamy、Rajiv Mathews、Kanishka Rao 和 Françoise Beaufays。2019 年。移动键盘中的表情符号预测联邦学习。*arXiv* 预印本 *arXiv:1906.04329*。Nikhil Sardana、Jacob Portes、Sasha Doubov 和 Jonathan Frankle。2023 年。超越 Chinchilla 最优：在语言模型缩放定律中考虑推理。*arXiv* 预印本 *arXiv:2401.00448*。Charlie Snell、Jaehoon Lee、Kelvin Xu 和 Aviral Ku- mar。2024 年。以最优方式扩展大语言模型的测试时计算，可能比扩展模型参数更有效。*arXiv* 预印本 *arXiv:2408.03314*。Emma Strubell、Ananya Ganesh 和 Andrew McCal- lum。2020 年。现代深度学习研究的能耗与政策考量。载于*AAAI* 人工智能会议论文集，第 34 卷，第 13693–13696 页。Youssef Tawfilis、Hossam Amer、Minar El-Aasser 和 Tallal Elshabrawy。2025 年。在数据共享约束下，面向异构多领域环境的分布式生成式人工智能方法。*arXiv* 预印本 *arXiv:2507.12979*。Ashish Vaswani、Noam Shazeer、Niki Parmar、Jakob Uszkoreit、Llion Jones、Aidan N. Gomez、Lukasz Kaiser 和 Illia Polosukhin。2017 年。注意力机制就够了。预印本，arXiv:1706.03762。

Yangzhen Wu、Zhiqing Sun、Shanda Li、Sean Welleck 和 Yiming Yang。2024 年。推理缩放定律：对使用语言模型解决问题的计算量最优推理进行实证分析。*arXiv* 预印本 *arXiv:2408.00724*。

An Yang、Anfeng Li、Baosong Yang、Beichen Zhang、Binyuan Hui、Bo Zheng、Bowen Yu、Chang Gao、Chengen Huang、Chenxu Lv 等另外 1 位作者。2025。Qwen3 技术报告。*arXiv* 预印本 *arXiv:2505.09388*。

Peiyuan Zhang、Guangtao Zeng、Tianduo Wang 和 Wei Lu。2024。Tinyllama：一个开源小型语言模型。预印本，arXiv:2401.02385。

Yaowei Zheng、Richong Zhang、Junhao Zhang、Yanhan Ye、Zheyan Luo、Zhangchi Feng 和 Yongqiang Ma。2024。Llamafactory：语言模型的统一高效微调 100+语言模型。载于计算语言学协会第 *62* 届年会论文集（第 *3* 卷：系统演示），泰国曼谷。计算语言学协会。

推导：持续刷新下测试时计算（TTC）的盈亏平衡条件

定义以下各量的单位时间值（例如，每年）：

- FLOPstrain：基线（不使用 TTC）的训练 FLOPs。
- FLOPsinfer：基线（不使用 TTC）的推理浮点运算次数。
- $r \in (0, 1]$：TTC 下训练 FLOPs 与基线训练 FLOPs 的比值，即 $r =$ FL~~OPsTTCt~~rain FLOPstrain 。
- λTTC > 0：使用 TTC 时的推理成本乘数（每令牌推理成本相对于“不使用 TTC”的基线）。
- λbase ≥ 1：基线部署中已有的推理成本乘数（若基线没有额外推理开销，则等于 1）。
- $f > 0$：刷新频率（单位时间内训练+部署周期的次数）。例如，如果模型每年重新训练并部署三次，那么 f =3。对于下面的代数推导，f 会抵消，因此不会影响界限。

基线和 TTC 感知运行模式下单位时间内的浮点运算次数总计为

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

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

这里假设，我们可以通过 λ_{base}或 λ_{TTC}来控制底层模型的 TTC 训练和/或部署。

要求 TTC 的成本不高于基线，可得

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

约去 f（假设 $f > 0$）：

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

因此，浮点运算次数层面的盈亏平衡条件为

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

接下来，用令牌数量来表示这个界限。设 N_{train} 为每次刷新（基线）所使用的训练词元数，N_{infer} 为下次刷新前处理的推理词元数。按照常见的近似，即一个训练词元的成本约为一个推理词元浮点运算次数的 6×倍（计入前向计算、 + 反向传播和 + 优化器的开销）（Kaplan 等人，2020年；Anil 等人，2023），

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

将此比值代入浮点运算次数不等式：

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

消去浮点运算次数$_{\text{infer}} > 0$并求解 N_{infer}：

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

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

设置 $\lambda_{\text{base}} = 1$即可得到基线没有额外推理开销这一特殊情况：

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

备注：

- 分母 λTTC $- \lambda$base 必须为正，该界才有意义。如果λTTC $\leq \lambda$base，则带有 TTC 的中间检查点的准确率不会优于完全训练检查点。
- 对于 λbase > 1，分母 (λTTC $- \lambda$base) 会减小，从而使 Ninfer 的上界增大。这意味着，随着基线产生的推理开销增加，TTC 感知训练在达到盈亏平衡点之前允许使用的推理令牌也会按比例增加。因此，我们将 λ base 设为 1，以尽量减少基线的开销。
- 该界定给出了两次刷新之间可提供的最大推理令牌数，同时将总浮点运算次数（训练加部署）控制在非 TTC 基线以内。

B 曲线拟合结果与分析

图 7 和 8 展示了对检查点进行指数曲线拟合的结果，拟合误差极小。由于我们在曲线拟合中排除了非单调点，因此消除了验证准确率中预期的波动。如表 5所示，我们观察到，移除这些波动后，TinyLlama 在 HumanEval 上的浮点运算量节省效果更佳——从 56.8% 提升至 90.7%。因此，我们用于训练的曲线拟合过程更加稳定，在至少保持准确率的同时实现了更高的浮点运算量节省。此外，图 9 展示了 Sigmoid 曲线预测结果，在不同浮点运算量级下，这些预测结果都能紧密跟踪各检查点的 TTC 准确率。

为进一步防范潜在的拟合失败，我们引入了耐心值，这是提前停止中常用的超参数。图 6 展示了不同耐心值对 TinyLlama 在 HumanEval 数据集上的测试时计算 Pass@16 解题率提升（相对于全量训练检查点）以及相应训练 FLOPs 节省的影响。可以看出明显的权衡关系：提高耐心值通常会提升解题率，但也会减少训练 FLOPs 的节省比例。较大的耐心值带来的提升略高，但 FLOPs 节省会随之减少，导致计算效率的收益递减。约为 10 的中间耐心值似乎最为理想，既能显著提高解题率，又能将训练 FLOPs 控制在合理范围内。这凸显了仔细调整耐心值、平衡性能与效率的重要性。

为进一步验证稳健性，我们还在温度为 0.7、0.8 和 0.9 的条件下重复了相同实验，每种温度使用三个随机种子。各检查点的平均标准差仍接近 0.1，这也证实了模型在不同解码设置下的稳定性。

C TTC 延迟开销

一个常见的顾虑是，测试时计算（TTC）会增加部署延迟，因为解码会重复进行 K 次（例如 Pass@K、多数投票或基于验证器的搜索）。在我们的框架中，这种开销通过两种方式得到明确控制。首先，早停过程会估算出满足目标准确率、同时仍符合公式*3*所述总计算量约束的最 K^* 小测试时计算预算。实证结果表明，K^* 这一开销并不大：在 TinyLlama/HumanEval 和 Qwen3-30B/GSM8k 上，中间检查点已能达到或超过全量训练基线的表现，且只需较小的 K （图3）；在 FineMath/Math-500 上，使用基于验证器的 DVTS、具有**100 亿 token**的检查点在 $K{=}4$ 时达到**24.73%**，优于具有**500 亿 token**、采用 $K{=}2$ 的**检查点**（18.28%）（表4）。这些结果表明，部署并不需要非常大的 K；在低到中等水平的K下即可实现 TTC 增益。

其次，我们通过公式 端到端盈亏平衡界限来量化开销，见公式 5，该界限保证了何时“TTC 感知训练 +部署”的成本仍低于训练并部署完全训练的基线。该界限使用 λ （TTC 引起的推理成本乘数）和 r（提前停止后的训练 FLOPs 比率）。表 1 显示，即便 TTC 带来不可忽略的开销，训练节省仍可在下次刷新前支撑大量下游推理：例如，开销适中时 $\lambda{\approx}1.2\times$, 该界限允许进行 **54T–72T** 个推理令牌，相当于 **52.7B–70.3B** 个长度为 1024 个词元的样本。即使开销更大（例如 $\lambda{=}8$ 或$\lambda{=}16$），盈亏平衡点仍约为 **0.96T–1.54T** 个推理令牌（表1）。

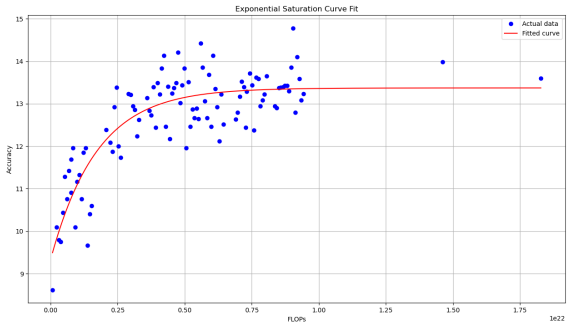
最后，训练期间的实际开销微不足道：曲线拟合（基于少数几个点拟合指数函数/ sigmoid 函数）可以通过闭式解完成，且仅在耐心机制触发检查时才运行；TTC 采样的开销则已由盈亏平衡分析全面计入。总体而言，TTC 感知提前停止以小幅、可控的推理时间乘数为代价，大幅减少训练计算量，从而加快模型开发并提高刷新周期的频率。从经济角度来看，即使在我们的 **1.1B** TinyLlama 实验中，TTC 感知提前停止也将有效训练量从**1,657GPU 天** 降至至少**127 GPU 天**（Pass@8），按 Azure V100 定价，成本约从**135k** 降至**10k**；由于训练成本与模型规模大致呈线性关系，在**70B–400B** 规模下，相对节省将更为可观。

D 最终准确率与 TTC 的拟合

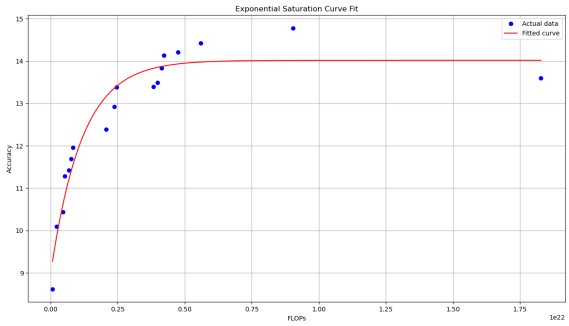
图 8 和 9分别展示了对指数饱和函数和 sigmoid 函数进行拟合的成功结果。

E TTC 感知指数拟合提前停止的详细信息

算法1 实现一种部署对齐的提前停止规则，用于决定何时停止训练

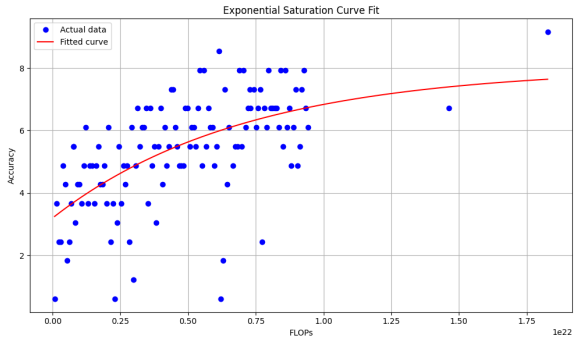


(a) 使用包含波动的所有检查点进行曲线拟合。

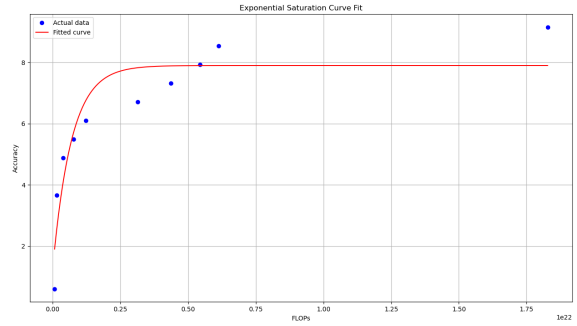


(b) 使用不含波动的所有检查点进行曲线拟合。

图7：**TinyLlama 在 DROP 上的指数饱和曲线拟合结果**。(a) 展示了使用所有可用检查点时的拟合结果。(b) 展示了使用所有可用检查点并忽略波动时的拟合结果。在所有情况下，红色曲线表示拟合函数，蓝色点表示实际数据，其中每个点对应一个在 DROP 数据集上评估的 TinyLlama 检查点。经过波动过滤后，拟合检查点与最终观测检查点之间的相对误差低至 +0.4%



（a）包含波动的所有检查点曲线拟合。



（b）不包含波动的所有检查点曲线拟合 .

图8：指数饱和曲线拟合结果。（a）展示了对**TinyLlama 在 HumanEval** 上使用所有可用检查点时的拟合结果。（b）展示了使用所有可用检查点并忽略波动时的拟合结果。在所有情况下，红色曲线表示拟合函数，蓝色点表示实际数据，每个点对应一个在 HumanEval 数据集上评估的 TinyLlama 检查点。经过波动过滤后，拟合检查点与最后一个实际检查点之间的相对误差为 = -1.8%

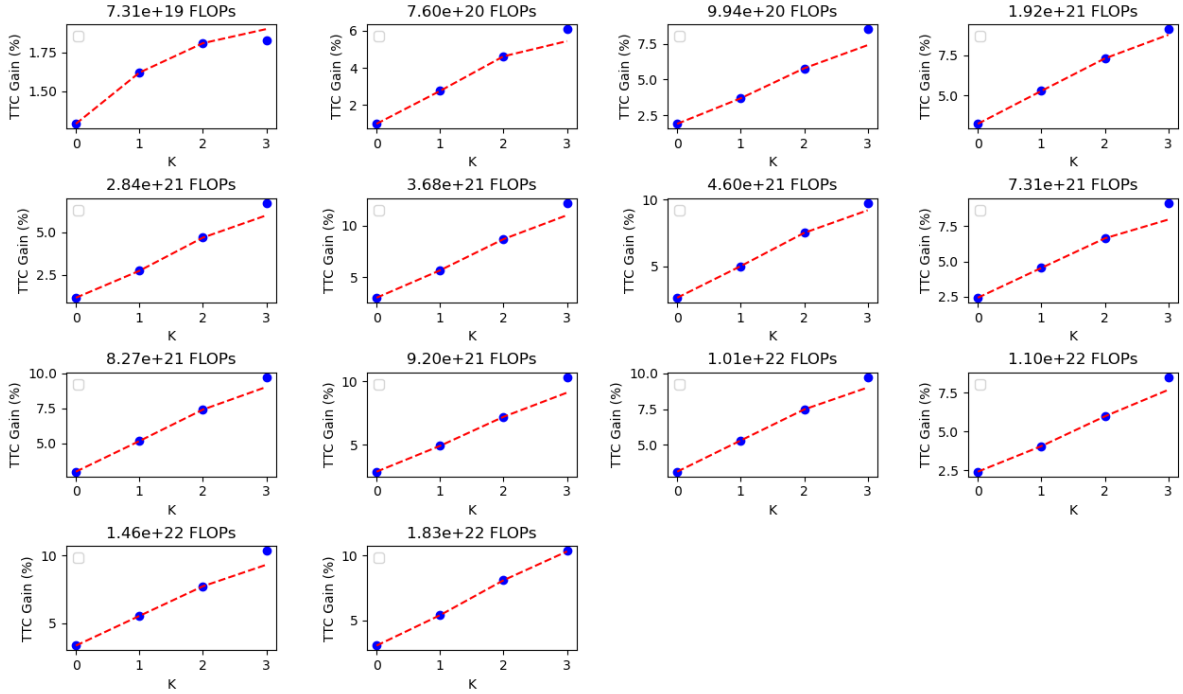


图9：不同浮点运算次数下 TTC 中参数 K 的 Sigmoid拟合。记录 $K = 1$、2、4 时的 TTC 增益，并使用拟合曲线估算在 $K = 8$ 时 TinyLlama-1B 在 HumanEval 数据集上的准确率。

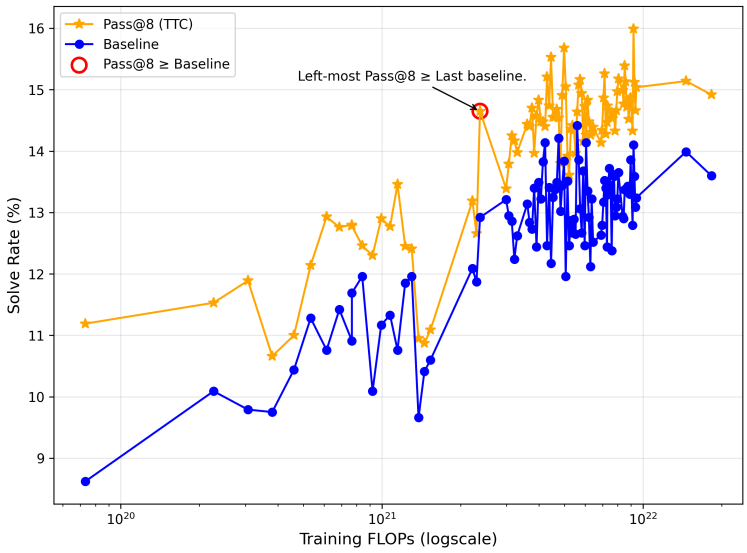


图10：使用 TinyLlama-1B 在 DROP 数据集上进行 TTC 感知训练的结果。与图1类似，应用 TTC 可实现提前停止，在不牺牲最终准确率的情况下节省高达 92% 的训练 FLOPs。这表明，TTC 感知能力能够推广到不同数据集，并非仅适用于单一基准测试。

训练，以及在选定检查点使用多少测试时计算量。该算法不再针对全量训练预算 B 下的最终检查点进行优化，而是搜索 FLOPs 为 t 的中间检查点，并为其配备最低限度的 TTC 预算 K^*，使其 (i) 达到或超过预测的完全训练后准确率，且 (ii) 总计算量低于训练至 B 并采用标准解码进行部署的总计算量。

符号与目标。在每个训练时间点 t，我们考虑两种成本：训练 FLOPs $F_{\text{tr}}(t)$，以及使用 TTC 时的推理 FLOPs $F_{\text{inf}}(t,\ K)$，其中 K 表示 TTC 方法使用的样本数/解码尝试次数（例如 Pass@K、多数投票、基于 DVTS 的搜索）。我们还将时间点 t、预算为 K 时经测试时计算验证的准确率记为 $\text{Acc}_t^{\text{ttc}}(K)$，该值可在留出验证集上估算。核心目标是找到一个计算量占优的TTC 检查点，即，

$$F_{\text{tr}}(t) + F_{\text{inf}}(t, K) < F_{\text{tr}}(B) + F_{\text{inf}}(B, 1), \quad \textit{and} \quad \text{Acc}_t^{\text{ttc}}(K) \geq Acc(B). \tag{8}$$

其中，$Acc(B)$ 表示（未知的）完全训练检查点的准确率。由于在完成训练之前无法获得 $Acc(B)$，我们通过学习曲线拟合对其进行估计。

第 1 步：用于预测的指数学习曲线拟合 $f(B)$。第 2 行定义了一个指数饱和模型 $f(x) = a(1 - e^{-bx}) + c$，其中 x 是累计训练浮点运算量（或映射到浮点运算次数的步数）。在每次迭代 t（第 4–6 行），算法将 $f(\cdot)$ 拟合到截至目前收集的验证准确率$\{val_acc\}$，并使用拟合曲线预测完整预算下的基线最终准确率 $f(B)$。该预测值可作为完全训练检查点性能的替代指标，并会随着观测到新的检查点而在线更新。

步数 2：选择最小的 TTC 预算 K^*。给定 t 处的当前检查点，算法选择满足以下两个约束（第 8 行）的最小 TTC预算 K^*：

$$F_{\text{tr}}(t) + F_{\text{inf}}(t, K^*) < F_{\text{tr}}(B) + F_{\text{inf}}(B, 1), \tag{9}$$

$$\text{Acc}_t^{\text{ttc}}(K^*) \geq f(B). \tag{10}$$

公式（9）中的计算量约束确保“在 t + 处停止并使用 TTC 部署”严格低于“训练至 B + 处并使用标准解码部署”的成本。公式（10）中的准确率约束确保检查点 t 处的 TTC 达到（或超过）预测的完全训练准确率。通过最小化 K^*，算法避免了不必要的 TTC 开销，并直接针对能够实现所需质量的最小部署时成本。

步骤 3：跟踪最佳 TTC 检查点。确定 K^* 后，算法会评估（或查找）该 K^* 对应的测试时计算验证准确率，记为 $val\_ttc\_acc_t[K^*]$。如果该值有所提升

超过目前为止观察到的最佳 TTC 验证准确率 $best\_ttc\_val\_acc$，则重置耐心计数器（第 10 行）并更新 $best\_ttc\_val\_acc$（第 12 行）。这样可确保我们保留在不等式和目标准确率所规定的 TTC 预算下最佳的可部署检查点。

步骤 4：基于耐心值的停止准则。当算法连续 p 次检查都未观察到能提升 TTC 表现的检查点时，停止条件（第 14–17 行）就会触发，前提是 TTC 表现已经达到预测基线之后。具体而言，如果 $best\_ttc\_val\_acc \geq f(B)$，耐心计数器就会递增；否则将其重置（第 14 行）。当 $patience\_counter \geq p$ 时，训练停止，算法返回迄今找到的最佳 TTC 检查点（第 16 行）。这种设计可以避免在 TTC 表现追上预测的最终基线之前过早停止，同时又能在训练轨迹从部署相关的角度趋于饱和后尽快终止。

实践说明。(i) 拟合基于少量观测点进行，并在线更新；实际中，仅在执行耐心值检查时才会调用。(ii) $F_{\text{inf}}(t,\ K)$ 可以直接测量（例如墙钟时间或浮点运算次数），也可以近似建模为在基于采样的测试时计算中随 K 线性变化；这足以用于选择 K^*。(iii) 该算法不依赖于特定的 TTC 流程：$\text{Acc}_t^{\text{ttc}}(K)$ 可以来自 Pass@K、多数投票、验证器引导搜索或其他测试时计算策略，只要准确率随 K 单调（或近似单调）提升即可。

表7：LLaMA-Factory 中 TinyLLaMA 的预训练配置（Zheng等人，2024）。

类别	配置
模型	量化_方法：无 训练 _from_从头训练：是 禁用_梯度_检查点：否
方法	阶段：预训练 do_训练：是 微调_类型：全量 deepspeed: examples/deepspeed/ds_z3_config.json
数据集	数据集：Cosmopedia-v2，来自 HuggingFaceTB/smollm-corpus 截断 _长度：2048 预处理_数量_工作进程：120 流式处理：False，打包：True 覆盖 _缓存：True 数据加载器 _num_工作进程：8
训练	每_设备_训练_批次_大小：16 梯度_累积_步数：4 学习_率：2.0e-5 学习率_调度器_类型：余弦 预热_比例：0.005 半精度浮点数：True，脑浮点数：False ddp_超时：180000000 num_训练_轮次：1
评估	val_大小：5000 每_设备_评估_批次_大小：1 评估_策略：步数 评估_步数：200
输出	日志记录_步数：1，保存_步数：200 绘制_损失：True 覆盖_输出_目录：True save_总计 _限制：500 报告_至：TensorBoard