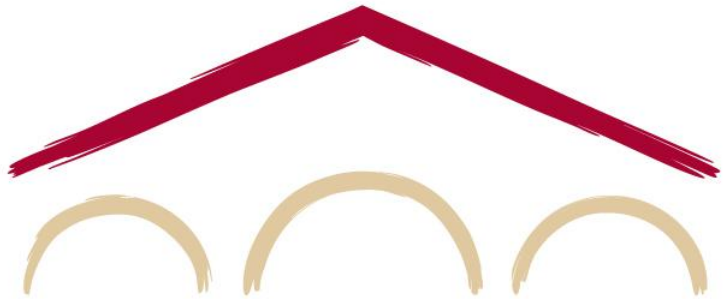
自然语言处理与深度学习

CS224N/Ling284



Yejin Choi

第 13 讲：推理 2 / 2

公告

- A4 is due this Thursday. We highly recommend to start working on it **now**, since it involves querying APIs. If everyone is working on it all right before the deadline, we'll 遇到速率限制问题。

- The 项目里程碑 instructions are out now, and we will be doing our best to get 今晚 / 明天上午完成项目提案评分和反馈。

课程计划

推测解码 （20 分钟）

Off-policy 漂移与 on-policy 蒸馏（20 分钟）

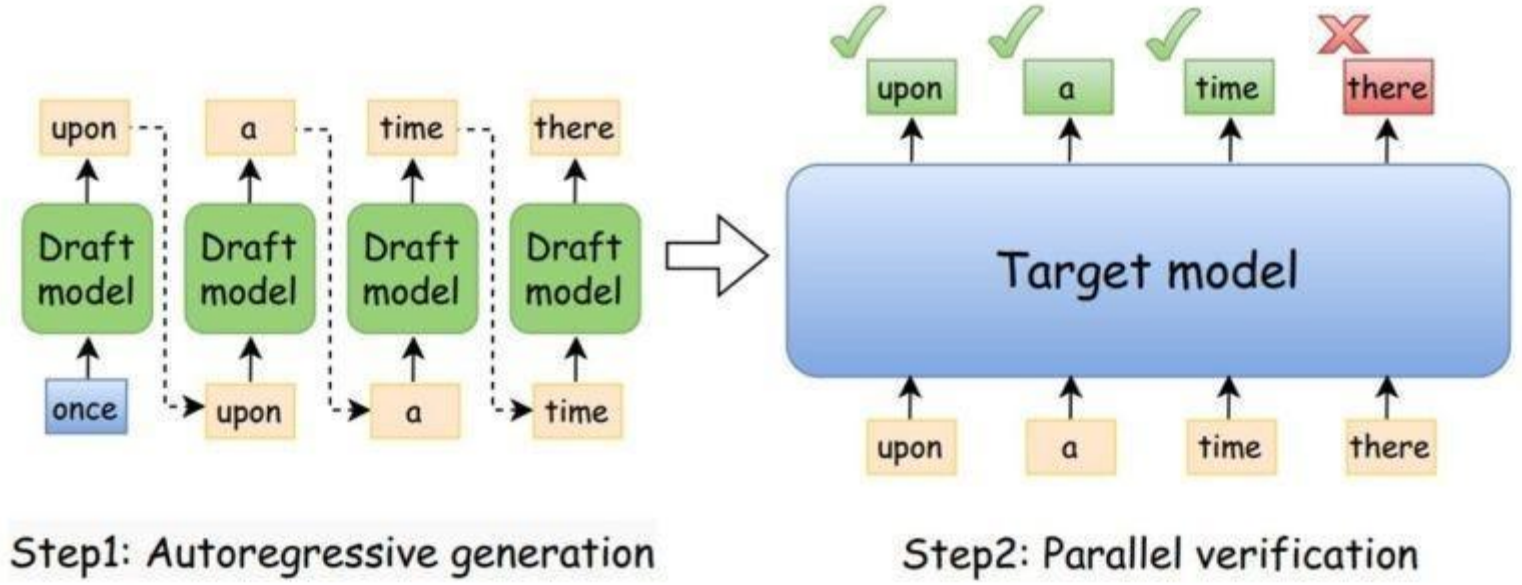
Off-policy、on-policy、在线蒸馏 基础设施与 off-policy 漂移
On-policy 蒸馏

长上下文扩展 （25 分钟）

推理时间扩展 （15 分钟）

推测解码

- Problem: Generating with a large LM takes a long time
  Intuition: Not all tokens are equally hard to generate!



- 思路：使用小　LM　的生成来辅助大　LM　的生成
  - Same idea independently proposed from Google Research (Leviathan et al., Nov 2022) and DeepMind (Chen et al., Feb 2023)

Image credit: https://medium.com/@genai.works/speed-up-llm-inference-with-speculative-decoding-1fc79701e9d6

推测解码

- First, sample a draft of length K (= 5 in this example) from a small LM M_p

$$y_1 \sim p(\cdot\,|x), y_2 \sim p(\cdot\,|x, y_1), \cdots, y_5 \sim p(\cdot\,|x, y_1, y_2, y_3, y_4)$$

- Then, compute the token distribution at each time step with a large target LM M_q

$$q(\cdot\,|x), q(\cdot\,|x, y_1), q(\cdot\,|x, y_1, y_2), \cdots, q(\cdot\,|x, y_1, \cdots, y_5)$$

  - Note: This can be computed in a *single forward pass* of M_q (Why?)

- Let's denote $p_i = p(\cdot\,|x, y_1, \cdots, y_{i-1})$ and $q_i = q(\cdot\,|x, y_1, \cdots y_{i-1})$
  e.g., $q_2 = q(\cdot\,|x, y_1)$, *i.e. next token distribution predicted by the target model* M_q, *when given* x *and* y_1

推测解码

- Now, we can compare the probability of each token assigned by draft model M_p and target model M_q

		y_1	y_2	y_3	y_4	y_5
		dogs	love	chasing	after	cars
草稿模型（1B）	p_i	0.8	0.7	0.9	0.8	0.7
目标模型（100B）	q_i	0.9	0.8	0.8	0.3	0.8

- Starting from y_1, decide whether to accept the tokens generated by the draft model.

推测解码

- Now, we can compare the probability of each token assigned by draft model M_p and target model M_q

		y_1	y_2	y_3	y_4	y_5
		dogs	love	chasing	after	cars
草稿模型 （1B）	p_i	0.8	0.7	0.9	0.8	0.7
目标模型 （100B）	q_i	0.9	0.8	0.8	0.3	0.8

- Starting from y_1, decide whether to accept the tokens generated by the draft model.
- Case 1: $q_i \geq p_i$
  The target model (100B) likes this token, even more than the draft model.
  => Accept this token!

第 1 步后的生成结果：
dogs

推测解码

- Now, we can compare the probability of each token assigned by draft model M_p and target model M_q

		y_1	y_2	y_3	y_4	y_5
		dogs	love	chasing	after	cars
草稿模型 （1B）	p_i	0.8	0.7	0.9	0.8	0.7
目标模型 （100B）	q_i	0.9	0.8	0.8	0.3	0.8

- Starting from y_1, decide whether to accept the tokens generated by the draft model.
- Case 2: $q_i < p_i$ (accept)
  Target model doesn't like this token as much as the draft model...
  => Accept it with the probability $\frac{q_i}{p_i}$

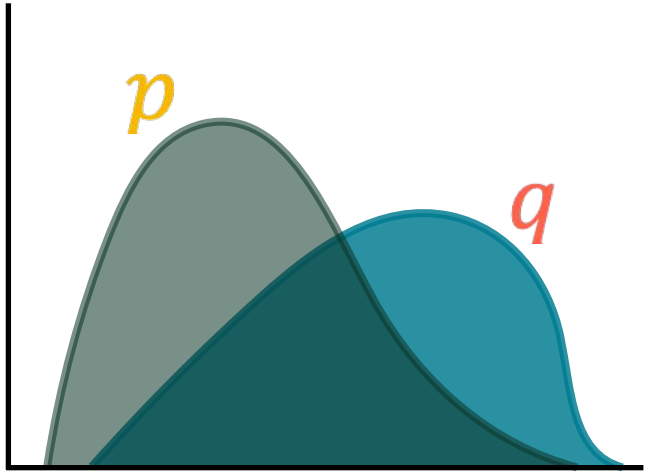
第 3 步后的生成结果：
dogs love chasing
（假设我们以概率 0.8 / 0.9 接受追逐）

推测解码

- Now, we can compare the probability of each token assigned by draft model M_p and target model M_q

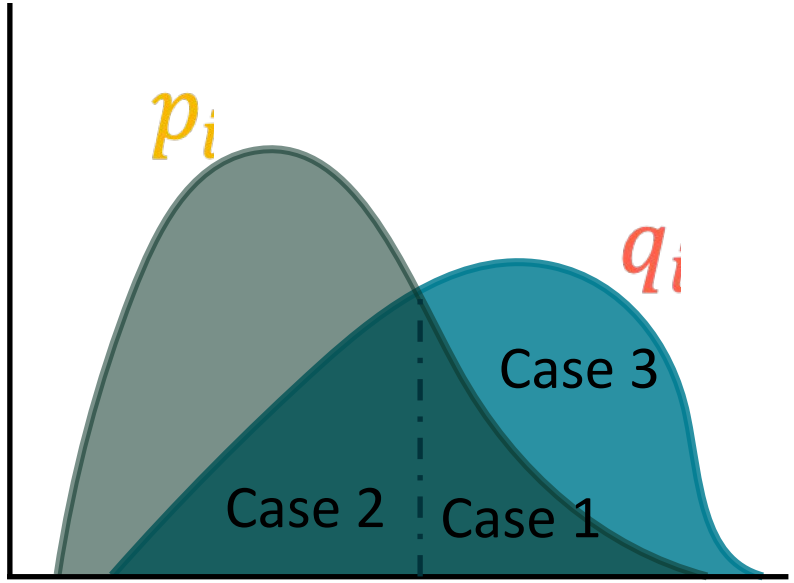
		y_1	y_2	y_3	y_4	y_5
		dogs	love	chasing	after	cars
草稿模型 （1B）	p_i	0.8	0.7	0.9	0.8	0.7
目标模型 （100B）	q_i	0.9	0.8	0.8	0.3	0.8

- Starting from y_1, decide whether to accept the tokens generated by the draft model.
- Case 3: $q_i < p_i$ (reject)
  If $q_i <<< p_i$, we likely would have rejected it.
  In this case, we sample a new token from target model
  => Specifically, we sample from $(q_i - p_i)_+$



推测解码

- But why specifically $(q_i - p_i)_+$?
  - because our goal: to cover target LM distribution q_i
- Case 1: $q_i \geq p_i$
  Accept this token.
- Case 2: $q_i < p_i$ (accept)
  Accept it with the probability $\frac{q_i}{p_i}$
- Case 3: $q_i < p_i$ (reject)
  If $q_i <<< p_i$, we likely would have rejected it.
  In this case, we sample a new token from target model
  => Specifically, we sample from $(q_i - p_i)_+$



Note: This sampling procedure,
though sampling from small LM (p_i), has
the same effect as sampling from target LM (q_i).
Formal proof in Appendix I of (Chen et al., 2023)

推测解码

- 推测采样是拒绝采样的一种形式。
  - To sample from an easy-to-sample distribution p (small LM), in order to ample from a 更复杂的分布 q（大 LM）中采样。
- Using 60M LM (T5-small) as a draft model and 11B (T5-XXL) LM as a target model, we get 2~3x acceleration with identical outputs!

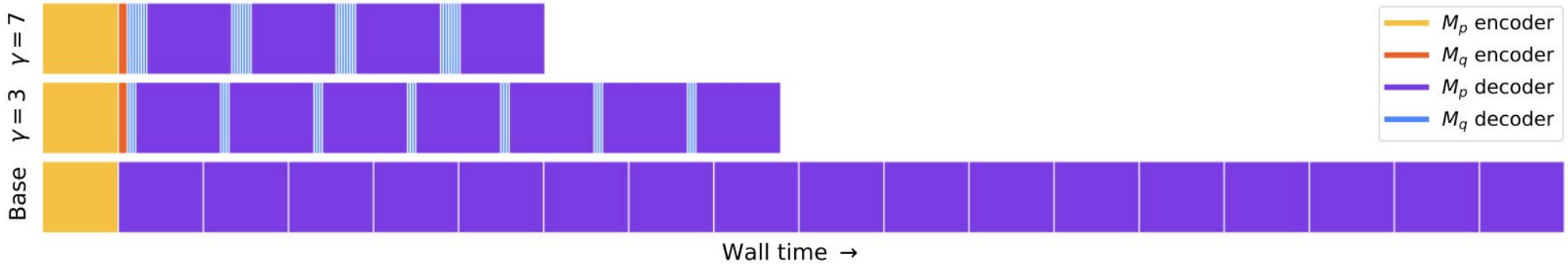


Figure 5. A simplified trace diagram for a full encoder-decoder Transformer stack. The top row shows speculative decoding with $\gamma = 7$ so each of the calls to M_p (the purple blocks) is preceded by 7 calls to M_q (the blue blocks). The yellow block on the left is the call to the encoder for M_p and the orange block is the call to the encoder for M_q. Likewise the middle row shows speculative decoding with $\gamma = 3$, and the bottom row shows standard decoding.

动态 speculative decoding

- adaptively adjusts the "lookahead" size (the number of candidate tokens) at each iteration by using a lightweight classifier or confidence threshold to decide on-the-fly whether the 草稿模型应继续起草还是切换到目标模型进行验证。
- See more: Mamou et al., 2024 & https://huggingface.co/blog/dynamic_speculation_lookahead

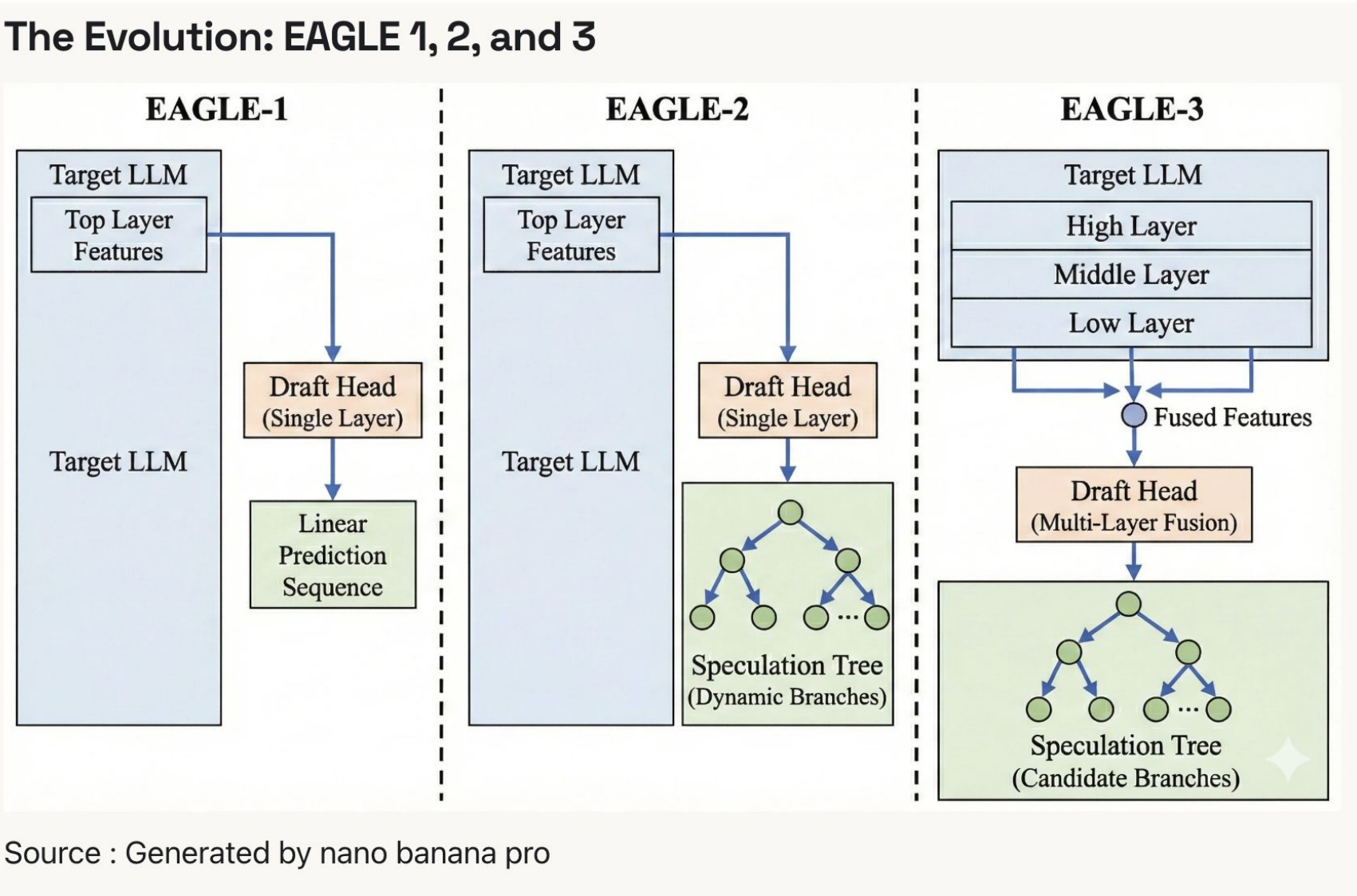
Universal speculative decoding

- Original spec decoding requires 草稿模型和目标模型之间相同的分词
- Models with 可以通过重新编码支持异构分词 and alignment techniques.
- 查看更多：https://huggingface.co/blog/universal_assisted_generation

Even better speculative decoding algorithms are rapidly developed, 而且你喜欢的推理引擎都支持多种选项！

算法	vLLM	TRT-LLM	HF TGI	HF Transformers	SGLang
EAGLE-3	✅ (Native)	✅ (Opt)	❌	⚠️ (Manual 头)	✅
SuffixDecoding	✅ (Arctic)	❌	❌	❌	✅ (Beta)
Medusa	✅	✅	✅ (SOTA)	✅ (Pipeline)	✅
Draft-Target	✅	✅	✅	✅ (Universal)	✅
DFlash (2026)	⚠️ (Fork)	✅ (Custom)	❌	❌	⚠️ (Fork)

EAGLE-3（更高 LM 效率的外推算法）



Key ideas:

Eagle-1: let the draft model "读取思维 " of the target model by sneaking into its internal representation ("features-level" speculation)

Eagle-2: context-aware 动态草稿树 !

Eagle-3: "fused features" 在不同层之间

Image credit: https://docs.jarvislabs.ai/blog/speculative-decoding-vllm-faster-llm-inference & nano banana pro

EAGLE-3（更高 LM 效率的外推算法）

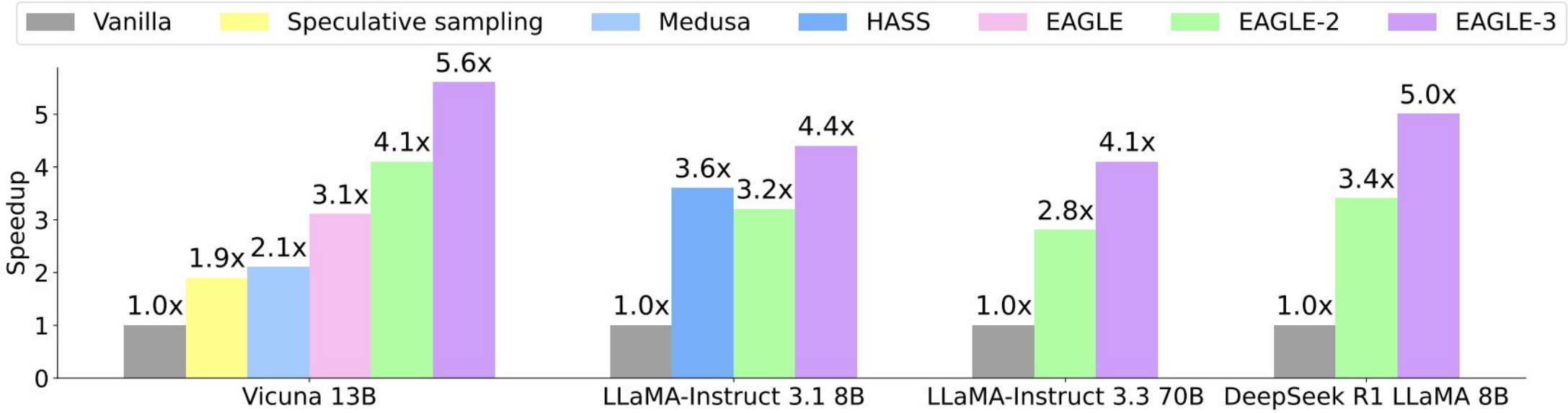


Figure 2: Speedup ratios of different methods at temperature=0. For the standard speculative sampling, Vicuna-13B uses Vicuna-68M as the draft model. In Table 1, we present comparisons with additional methods, but this figure only showcases a subset. Chat model's evaluation dataset is MT-bench, and the reasoning model's evaluation dataset is GSM8K. DeepSeek R1 LLaMA 8B refers to DeepSeek-R1-Distill-LLaMA 8B.

Suffix Decoding: Extreme Speculative Decoding (Oliaro et al., 2025)

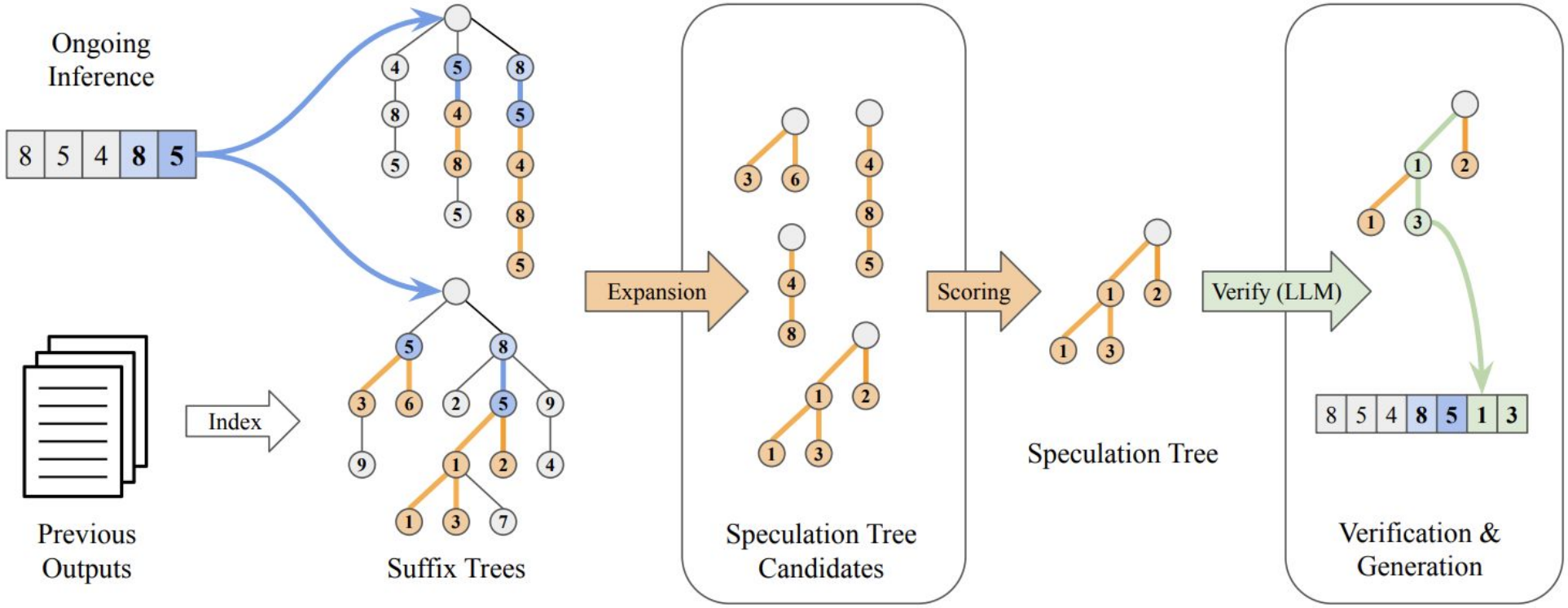
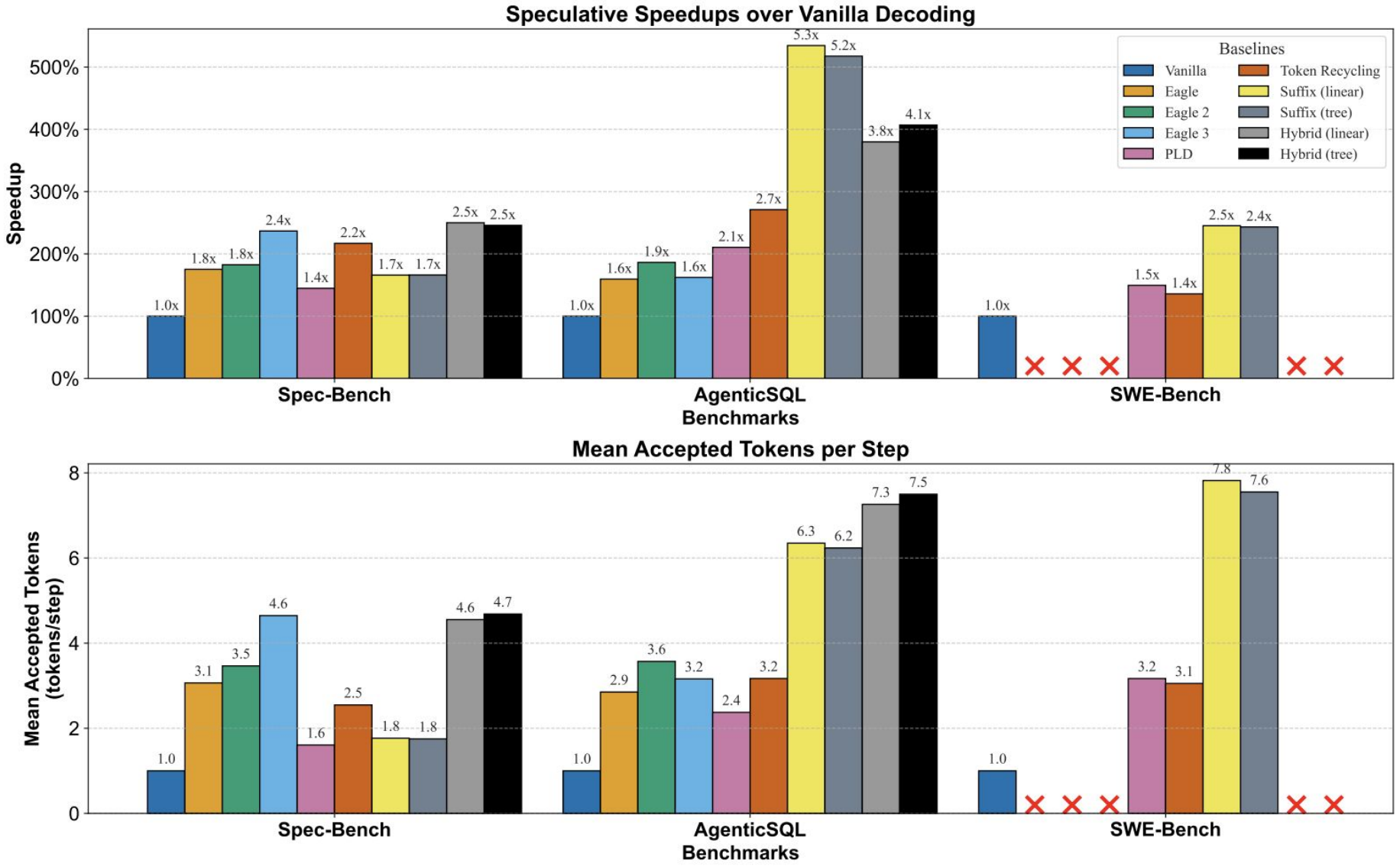


Figure 1: Overview of SuffixDecoding's algorithm. Two suffix trees track ongoing inference (top-left) and previous outputs (bottom-left). SuffixDecoding uses these trees to find matching patterns based on recently generated tokens. It constructs a speculation tree (middle) by selecting the most likely continuations, scoring them based on frequency statistics. Finally, the best candidate is verified by the LLM in a single forward pass (right), with accepted tokens (shown in green) being added to the output and used for the next round of speculation.

Suffix Decoding: Extreme Speculative Decoding (Oliaro et al., 2025)



Suffix Decoding vs Eagle-3

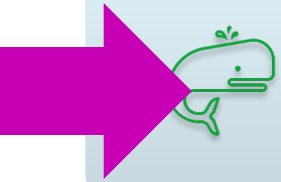
特性	**SuffixDecoding (2025/2026)**	**EAGLE-3 (2025/2026)**
机制	**Model-Free:** Uses a Suffix Tree to cache and match repetitive sequences 在提示和过去输出中。	**Model-Based:** Uses a small, trained Transformer head to predict future 目标模型的隐藏特征。
计算位置	**CPU-Bound:** Runs speculation on the CPU while the GPU handles the target 模型的验证。	**GPU-Bound:** The draft head runs on the GPU, sharing VRAM with the main 模型。
起草速度	每 token ~20 μs（极快）	较慢（需要 GPU 前向传递）。
最佳性能	高度重复性任务 (Coding, Agentic 循环、RAG、SQL 生成）。	开放式任务 (Creative writing, general chat, unpredictable 推理）。
训练需求	Zero. It is a "plug-and-play" data 结构。	Requires training a small auxiliary head on the target model’s feature 空间。

Even better speculative decoding algorithms are rapidly developed, 而且你喜欢的推理引擎都支持多种选项！

算法	vLLM	TRT-LLM	HF TGI	HF Transformers	SGLang
EAGLE-3	✅ (Native)	✅ (Opt)	❌	⚠️ (Manual 头)	✅
SuffixDecoding	✅ (Arctic)	❌	❌	❌	✅ (Beta)
Medusa	✅	✅	✅ (SOTA)	✅ (Pipeline)	✅
Draft-Target	✅	✅	✅	✅ (Universal)	✅
DFlash (2026)	⚠️ (Fork)	✅ (Custom)	❌	❌	⚠️ (Fork)

课程计划

推测解码 （20 分钟）



Off-policy 漂移与 on-policy 蒸馏（20 分钟）

Off-policy、on-policy、在线蒸馏 基础设施与 off-policy 漂移
On-policy 蒸馏

长上下文扩展 （25 分钟）

推理时间扩展 （15 分钟）

Online RL： 智能体可以在训练期间与环境交互

Offline RL (Batch RL): Learning happens strictly from a pre-recorded dataset (human logs or 之前的智能体）。智能体无法 " 探索 " 或测试新动作。

On-Policy RL: The data used for training is generated exactly by the *current* policy. Once the 策略更新后，旧策略的数据被丢弃为 " 过时的 " 。

REINFORCE, PPO, GRPO

Off-Policy RL: The agent learns from data generated by any policy (older policy, different policy, 甚至人类）。它将经验保存在 * 回放缓冲区 * 中以供重复使用。

DQN (Deep Q-Network)

Online vs **Offline RL**: 智能体是否可以与环境交互

On-policy vs **Off-policy RL**: rollout 训练数据是否来自最新的策略

Offline On-Policy RL?

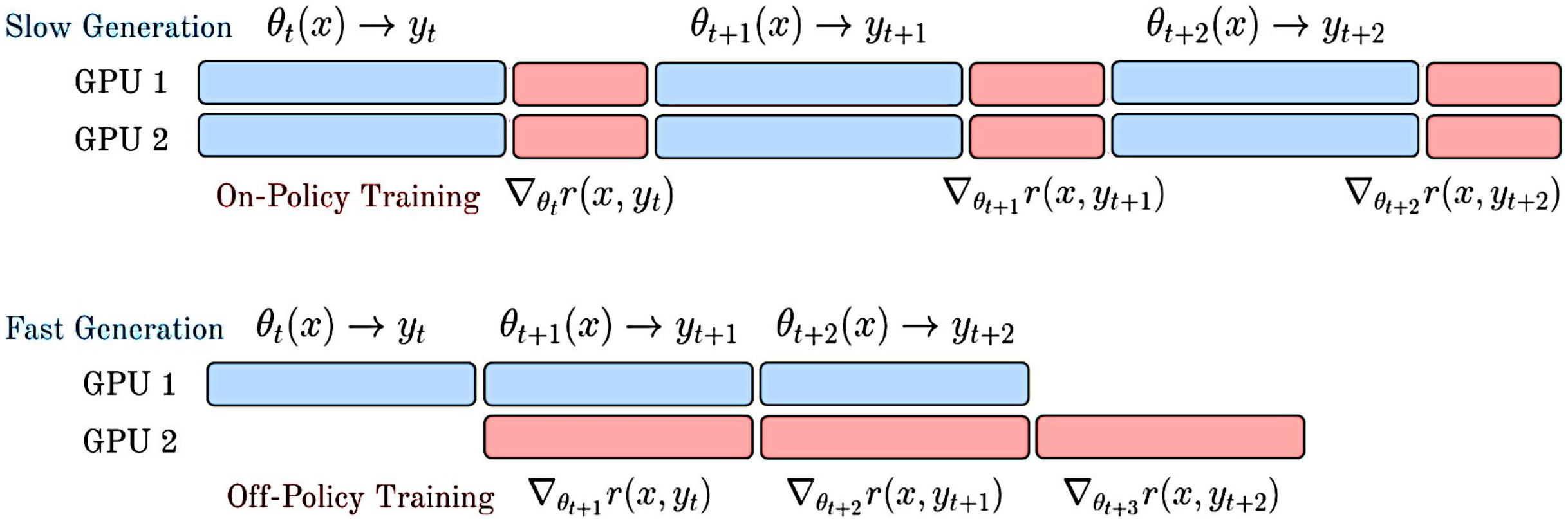
- no true offline on-policy RL (unless the interpretation is 略有拉伸 …)

Online Off-Policy RL?

- This is very common. An agent interacts with a simulator (Online) but stores everything in a 回放缓冲区 to learn from later (Off-Policy), like **DQN**.
- Asynchronous RL
- PPO with "off-policy" due to RL infra 漂移优化

异步 RLHF (Noukhovitch et al., 202

- "原因"： 经典 RLHF 是同步的，浪费了 GPU 吞吐量



PipelineRL: in-flight weight updates ✈ (Piché et al., 2025)

- The generation engine receives new model weights mid-generation — briefly pausing, loading fresh weights, then continuing in-progress sequences. This creates mixed-policy sequences where early tokens are generated by a staler policy and later tokens by a fresher one.

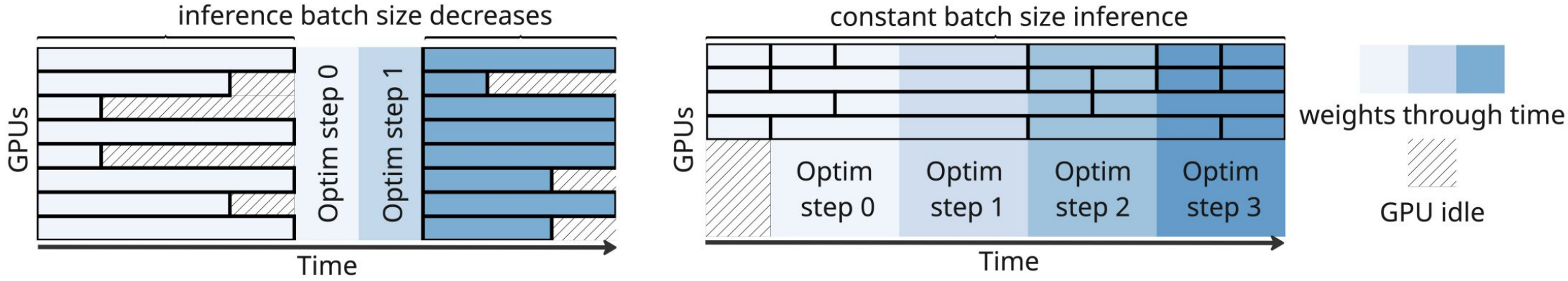


Figure 1: **a)** Conventional RL alternates between using all the GPUs for generation and then training. **b)** PipelineRL runs generation and training concurrently, always using the freshest model weights for generations thanks to the in-flight weight updates.

为什么 off-policy 情况在实践中出现

- The Two-Engine Architecture

1. The Rollout Generator

   Input: The current policy (π_θ)

   Output: Rollouts

2. The Learner

   Input: Batches of rollouts from the generator.

   Output: Updated policy ($\pi_{\theta+1}$)

- Why rollouts become ”stale"
  - Deliberate asynchronous RL
  - Multiple gradient steps per batch
  - Large replay buffers
  - Separate generation and training clusters (one cluster optimized for inference, while another cluster optimized for gradient computation) require weight transfer between

    集群之间，引入延迟

Off-policy 缓解措施

为什么 off-policy 是个问题

- 有偏梯度估计
- 重要性权重爆炸
- 奖励黑客放大效应

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

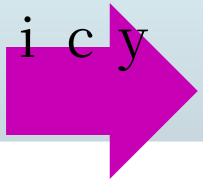
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

- 缓解策略
  - PPO clipping
  - KL penalty against the reference policy
  - Algorithms without critiques/advantages (reducing the window of off-policy drifts)
    如 GRPO 或 REINFORCE
  - Less epochs
  - SGLang / vLLM's continuous batching with 权重流式传输
  - Pipeline RL's in-flight weight update

课程计划

推测解码 （20 分钟）

Off-policy 漂移与 on-policy 蒸馏（20 分钟）



Off-policy、on-policy、在线蒸馏 基础设施与 off-policy 漂移
On-policy 蒸馏

长上下文扩展 （25 分钟）

推理时间扩展 （15 分钟）

On-policy 蒸馏（又称广义知识蒸馏）

- First introduced by (Gu et al, 2023) and (Agarwal et al, 2023), and later amplified by Qwen3's Tech report and ThinkingMachine's blog (https://thinkingmachines.ai/blog/on-policy-distillation/)
- 先前的蒸馏方法以教师为中心（因此对学习者而言是 off-policy 的）
- On-policy 蒸馏以学生为中心（因此对学习者而言是 on-policy 的）

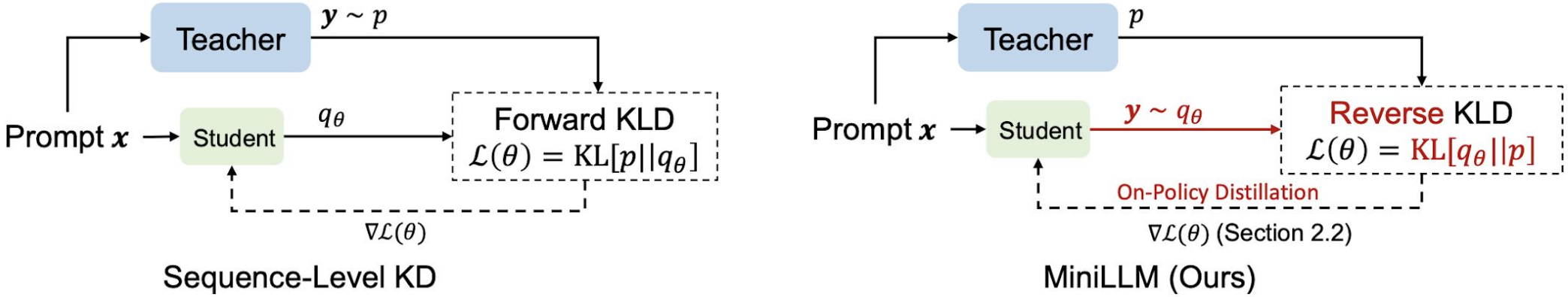


Figure 3: Comparison between sequence-level KD (left) and MINILLM (right). Sequence-level KD forces the student to memorize all samples generated by the teacher model, while MINILLM improves its generated texts with the teacher model's feedback.

Gu et al. 2023

KL Divergence: how Q differs from the target P

Forward KL — **"Mean-seeking" or "mass-covering"**

$$\mathrm{KL}(p \,\|\, q) = \mathbb{E}_{x\sim p}\left[\log \frac{p(x)}{q(x)}\right] = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

- Weighted by P(x): 惩罚 P 有质量但 Q 无质量处的 Q
- Q covers 所有模式 of P → over-estimates support
- Used in variational inference (ELBO)

Reverse KL — **"Mode-seeking" or "mode-collapsing"**

$$\mathrm{KL}(q \,\|\, p) = \mathbb{E}_{x\sim q}\left[\log \frac{q(x)}{p(x)}\right] = \sum_x q(x) \log \frac{q(x)}{p(x)}$$

- Weighted by Q(x)： 惩罚 Q 在 P 无质量处的质量
- Q concentrates on **one mode** of P → under-estimates support
- Used in RLHF: KL penalty keeps policy close to reference model

On-policy 蒸馏 vs 标准知识蒸馏

Let p_T = teacher, p_θ= student, $y*$ = ground truth, x = input

- Knowledge distillation (Hinton et al., 2015)

loss = forward KL

off-policy

$$\mathcal{L}_{\mathrm{KD}} = -\sum_{t=1}^{T}\sum_{v\in\mathcal{V}} p_T^{(\tau)}(v \mid y^*_{<t}, x)\ \log p_\theta^{(\tau)}(v \mid y^*_{<t}, x)$$

- Sequential knowledge distillation (Kim & Rush, 2016)

loss = NLL

off-policy

$$\mathcal{L}_{\mathrm{SeqKD}} = -\sum_{t=1}^{T} \log\, p_\theta(\hat{y}_t \mid \hat{y}_{<t}, x) \qquad \text{where} \quad \hat{y} \sim p_T(\cdot \mid x)$$

- On-policy knowledge distillation (Gu et al., 2023; Agarwal et al, 2023)

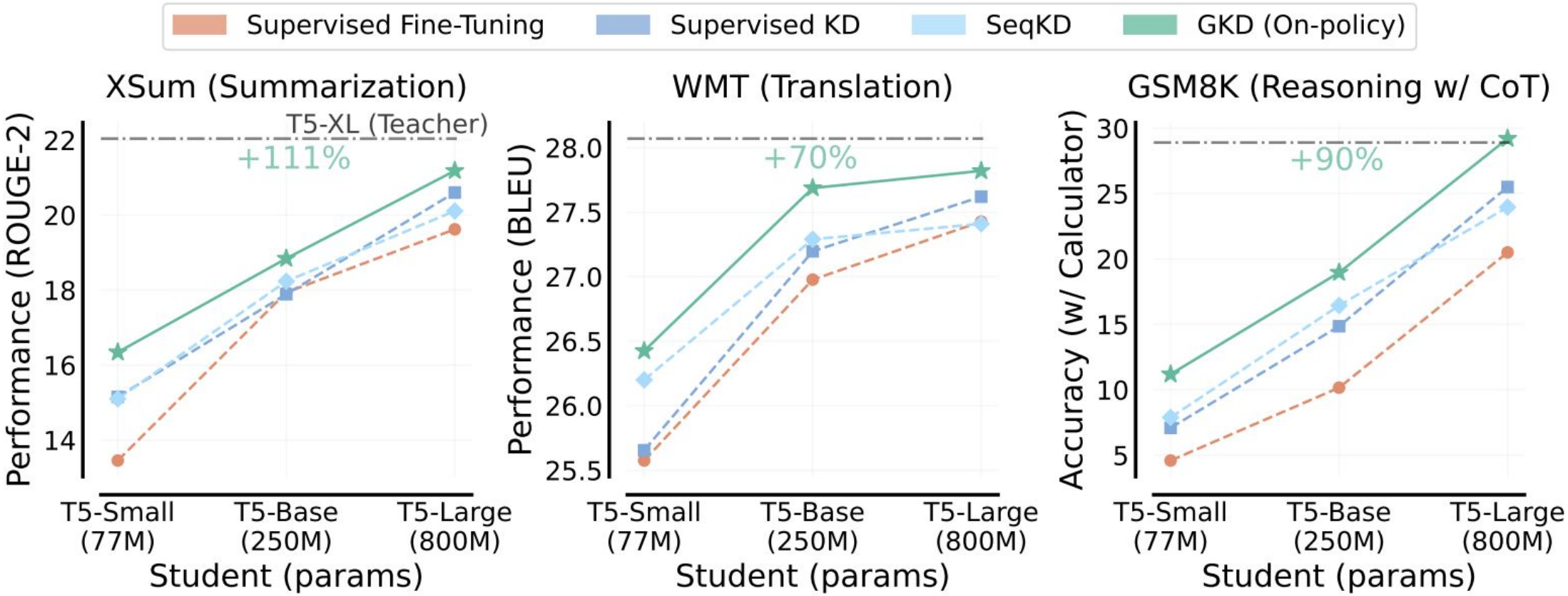
loss = reverse KL

on-policy

$$\mathcal{L}_{\mathrm{GKD}} = \sum_{t=1}^{T}\sum_{v\in\mathcal{V}} p_\theta(v \mid \tilde{y}_{<t}, x)\left[\log p_\theta(v \mid \tilde{y}_{<t}, x) - \log p_T(v \mid \tilde{y}_{<t}, x)\right]$$

$$\text{where} \quad \tilde{y} \sim p_\theta(\cdot \mid x)$$

	损失	上下文	硬标签 vs 软标签	习方式
SFT	金标准输出上的 NLL	真实上下文	硬标签	
KD	Forward KL	真实上下文	软分布	Mass-covering
SeqKD	NLL on teacher's output	教师生成的上下文	硬标签	Mass-covering
GKD	Reverse KL	学生生成的上下文	软分布	Mode-seeking



Agarwal et al. 2023

On-policy 蒸馏

- The implication of reverse-KL: the context comes from the student's own generation.
- This eliminates train-test mismatch (exposure bias): the student learns to recover from its 自己的错误 .
- The mode-seeking property encourages the student to be sharp and confident on its best behaviors rather than diffusely covering all of 教师的模式。

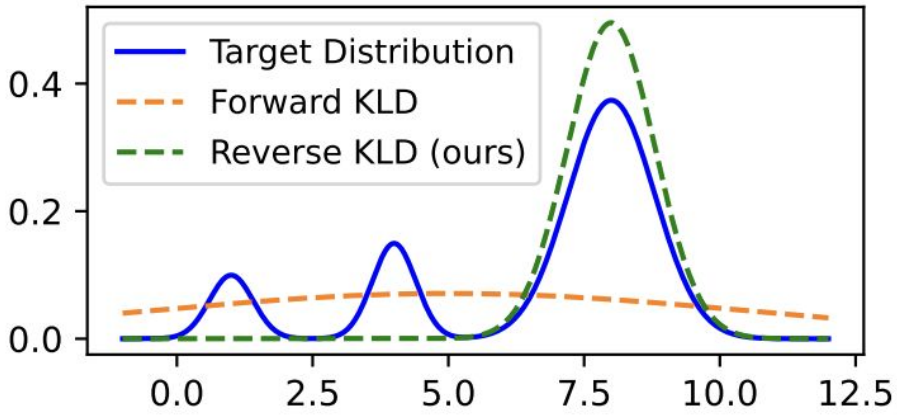


Figure 2: The toy experiment. We fit a Gaussian mixture distribution with a single Gaussian distribution using *forward* KLD and *reverse* KLD.

Gu et al. 2023

On-policy 蒸馏 vs RL

- On policy distillation can be viewed as the best of both worlds of SFT and RL ---

	采样	奖励信号	GPU 需求
有监督微调	Off-policy	密集	轻量
强化学习	On-policy	稀疏	重量级
On-policy 蒸馏	On-policy	密集	轻量

Table 21: Comparison of reinforcement learning and on-policy distillation on Qwen3-8B. Numbers in parentheses indicate pass@64 scores.

Method	AIME'24	AIME'25	MATH500	LiveCodeBench v5	MMLU -Redux	GPQA -Diamond	GPU Hours
Off-policy Distillation	55.0 (90.0)	42.8 (83.3)	92.4	42.0	86.4	55.6	-
+ Reinforcement Learning	67.6 (90.0)	55.5 (83.3)	94.8	52.9	86.9	61.3	17,920
+ On-policy Distillation	**74.4 (93.3)**	**65.5 (86.7)**	**97.0**	**60.3**	**88.3**	**63.3**	1,800

On-policy distillation for domain-specific adaptation 🥳

- One very cool use case of on-policy distillation is domain-specific adaptation of OSS models, e.g., fine-tuning Qwen-8B on a domain-specific corpus (i.e., internal documents 公司的内部文档），可以视为一种中期训练形式。
- Doing this in a naïve way (applying midtraining on top of the off-the-shelf OOS model which has been already post-trained) will cause degradation of capabilities acquired 通过之前的后训练（如指令遵循）
- On-policy distillation (on an instruction-tuning dataset, 30%) recovers the instruction following capability after acquiring the new domain-specific knowledge thru mid-training (7(

Model	Internal QA Eval (Knowledge)	IF-eval (Chat)
Qwen3-8B	18%	**85%**
+ midtrain (100%)	**43%**	45%
+ midtrain (70%)	36%	79%
+ midtrain (70%) + distill	**41%**	**83%**

课程计划

推测解码 （20 分钟）

Off-policy 漂移与 on-policy 蒸馏（20 分钟）

Off-policy、on-policy、在线蒸馏 基础设施与 off-policy 漂移
On-policy 蒸馏

长上下文扩展 （25 分钟）

推理时间扩展 （15 分钟）

扩展推理能力需要扩展长上下文

- 当应用需要非常长的上下文窗口（１００ｋ　到　２００　万　ｔｏｋｅｎ）时：
  - Software engineering tasks that demand understanding the entire repository
  - Legal analysis that involves careful review of documents spanning hundreds of pages
  - Personalized chat interactions conditioned on prolonged interaction histories
  - Solving extremely challenging math problems that require elaborate sequences of 在不同的解题策略之间进行试错

扩展推理能力需要扩展长上下文

是什么使长上下文对　ＬＬＭ　具有挑战性

- Data limitation: most internet documents aren’t long enough to support pre-training

  具有极长的上下文窗口
- Compute/memory limitation: standard attention requires quadratic computation
- Generalization limitation of positional embeddings: models trained on shorter sequences doesn’t generalize well on longer sequences

互联网数据的典型文档长度

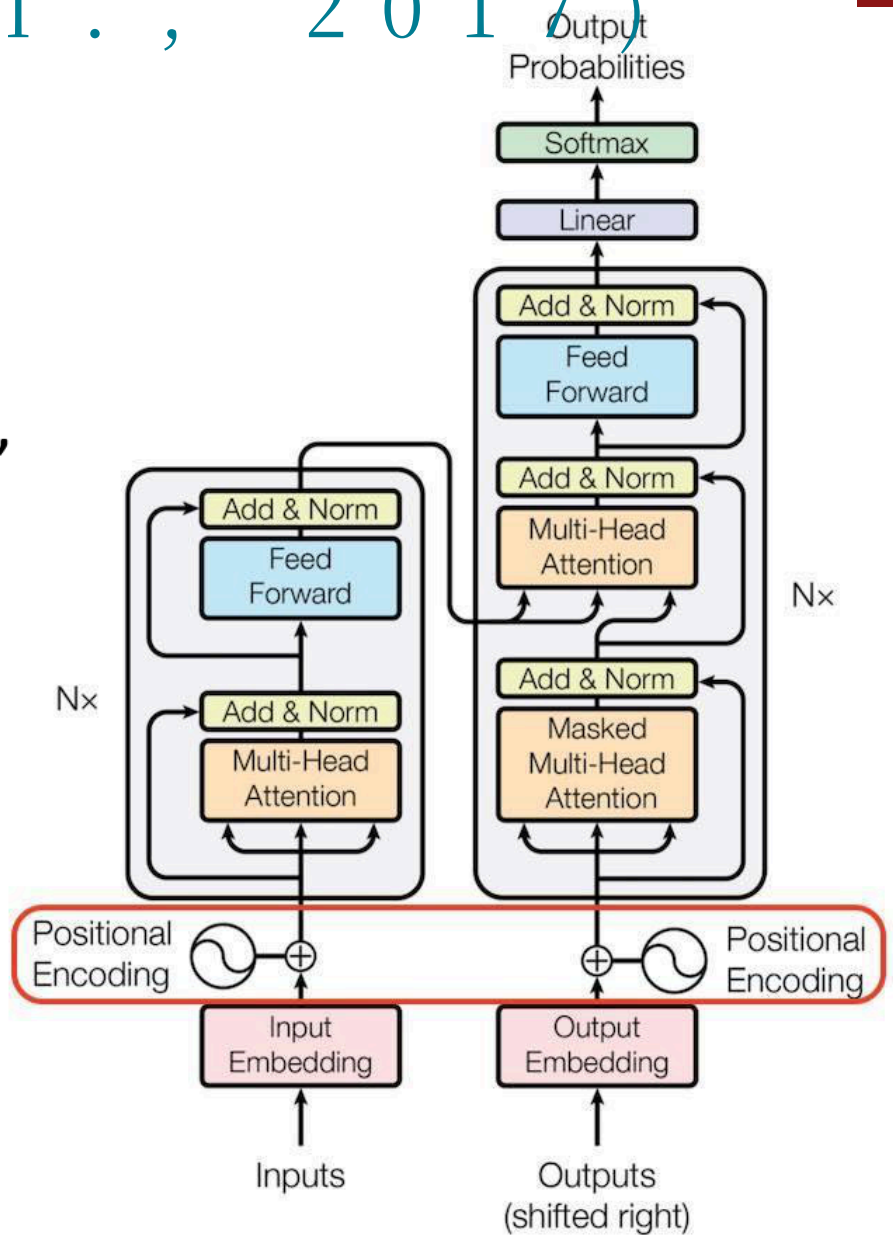
来源类型	典型长度（约 Token 数）	描述
Common Crawl（网络）	600 – 1,200	Includes blog posts, news, and landing pages. 许多 " 文档 " 高度碎片化。
Wikipedia	500 – 1,500	While some entries are long, the median article is 相对简洁。
科学论文 （arXiv）	5,000 – 10,000+	These are among the longest "natural" documents 在大多数集合中。
书籍 （Project Gutenberg）	50,000 – 100,000+	The "long-tail" of the data; rare but critical for 长距离依赖。
GitHub（代码）	100 – 5,000	Code files vary wildly; many scripts are quite 较短，而库则很长。

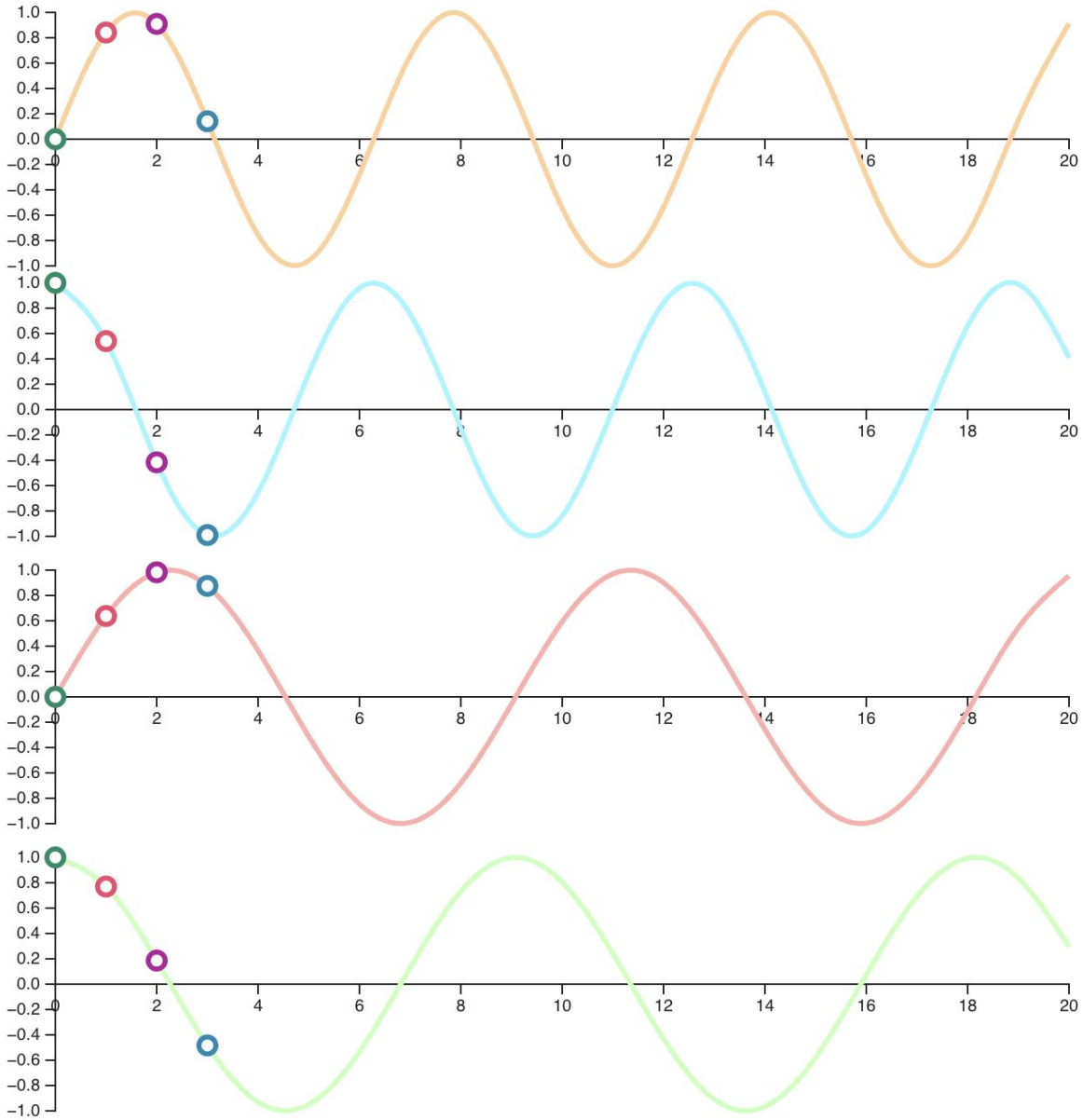
正弦位置嵌入（Vaswani et al., 2017）

$$PE(pos, 2i) = \sin(pos/10000^{2i/d})$$

$$PE(pos, 2i+1) = \cos(pos/10000^{2i/d})$$

- Where pos is the position index, i is the dimension index, and d is the model dimension.
- Used in the original Transformer (Vaswani et al., 2017)
- We are revisiting this because it helps to learn RoPE (Rotary positional embedding) that is recently important for LLMs and long-context extension





p0	p1	p2	p3	
0.000	0.841	0.909	0.141	i=0
1.000	0.540	-0.416	-0.990	i=1
0.000	0.638	0.983	0.875	i=2
1.000	0.770	0.186	-0.484	i=3

Positional Encoding

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{2i/d_{\text{model}}}})$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{2i/d_{\text{model}}}})$$

Settings: d = 50

The value of each positional encoding depends on the *position* (*pos*) and *dimension* (*d*). We calculate result for every *index* (*i*) to get the whole vector.

https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers

正弦位置嵌入（Vaswani et al., 2017）

$$PE(pos, 2i) = \sin(pos/10000^{2i/d})$$
$$PE(pos, 2i+1) = \cos(pos/10000^{2i/d})$$

- Where pos is the position index, i is the dimension index, and d is the model dimension.
- Why this particular sinusoidal form?
  - Unique absolute positional embedding pre-defined for *any* position (whether seen during training or not)
  - Captures relative distance! For any offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}
- Two important things to note:
  - Positional embedding is “added” to the token embedding
  - Previously unseen positional embeddings during training, while can be defined, are still unseen to the model, thus the model can’t interpret them

可学习的位置嵌入

- Randomly initialized, and then learned via backprop
- Used broadly in early days of LLMs, such as BERT, Roberta, GPT-2, GPT-3, Albert, Electra, BART
- It is *not* possible to define the positional embedding for a previously unseen position.
- Also doesn't generalize to positions beyond those seen during training.
- 但当模型自己学习位置编码时性能更好
- This became a major bottleneck for long-context extension however, thus no longer used in recent LLMs

RoPE：旋转位置嵌入 （Su et al., 2021）

- 回忆注意力

$$Q = W_Q X, \qquad K = W_K X, \qquad V = W_V X$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

- 带有 RoPE 的注意力

$$\tilde{q}_m = R_\Theta^{(m)} q_m, \qquad \tilde{k}_n = R_\Theta^{(n)} k_n$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{\tilde{Q}\tilde{K}^\top}{\sqrt{d_k}}\right)V$$

After query and key vectors are computed, multiply them with the rotation matrix!

Rotations are applied to just query and key vectors, 不是 value 向量！

RoPE：旋转位置嵌入（Su et al., 2021）

$$\begin{pmatrix} x'_{2i} \\ x'_{2i+1} \end{pmatrix} = \begin{pmatrix} \cos(m\theta_i) & -\sin(m\theta_i) \\ \sin(m\theta_i) & \cos(m\theta_i) \end{pmatrix} \begin{pmatrix} x_{2i} \\ x_{2i+1} \end{pmatrix}$$

Where m is the position index, i is the dimension index, $\theta_i = b^{-2i/d}$ is the frequency for dimension pair i, and b is the base frequency (typically $b = 10{,}000$)

RoPE 的完整旋转矩阵如下所示：

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

RoPE：旋转位置嵌入 （Su et al., 2021）

$$\begin{pmatrix} x'_{2i} \\ x'_{2i+1} \end{pmatrix} = \begin{pmatrix} \cos(m\theta_i) & -\sin(m\theta_i) \\ \sin(m\theta_i) & \cos(m\theta_i) \end{pmatrix} \begin{pmatrix} x_{2i} \\ x_{2i+1} \end{pmatrix}$$

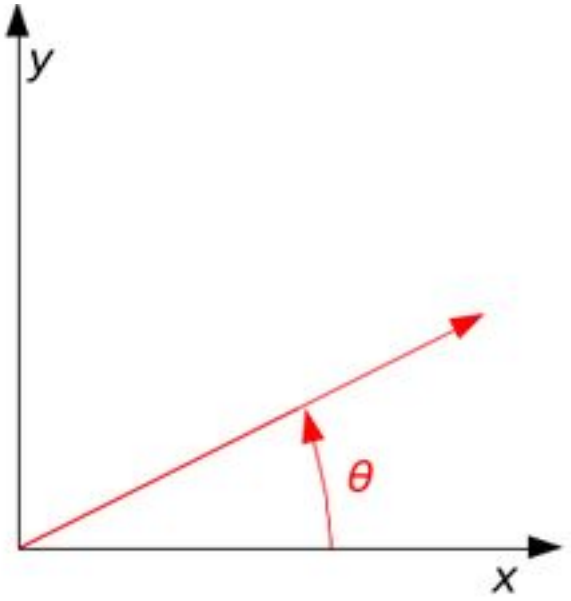
Where m is the position index, i is the dimension index, $\theta_i = b^{-2i/d}$ is the frequency for dimension pair i, and b is the base frequency (typically $b = 10{,}000$)

- RoPE 通过在 2D 平面中*旋转* query 和 key 向量来编码位置。

回顾旋转矩阵：

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



RoPE：旋转位置嵌入 （Su et al., 2021）

- RoPE encodes position by *rotating* the query and key vectors in the 2D plane.

- **High-frequency dimensions** (small i): rotate rapidly, encoding fine-grained local positional information. These dimensions cycle through many full rotations over the training length.
- **Low-frequency dimensions** (large i): rotate slowly, encoding broad/global positional information. These dimensions complete very few full rotations even over very long sequences.

- Dimension pair 0: $\theta_0 = 1.0$ — highest frequency, one full rotation every ~ 6.28 tokens
- Dimension pair 31: $\theta_{31} = 0.01$ — one rotation every ~ 628 tokens
- Dimension pair 63: $\theta_{63} = 0.0001$ — one rotation every $\sim 62{,}832$ tokens

RoPE：旋转位置嵌入 （Su et al., 2021）

- 带有 RoPE 的注意力

$$Q = W_Q X, \qquad K = W_K X, \qquad V = W_V X$$

$$\tilde{q}_m = R_\Theta^{(m)} q_m, \qquad \tilde{k}_n = R_\Theta^{(n)} k_n$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{\tilde{Q}\tilde{K}^\top}{\sqrt{d_k}}\right)V$$

- 相对距离
  - because the transposed R rotates backward, we have

$$R^{(m)\top} R^{(n)} = R^{(n-m)}$$

$$\tilde{q}_m^\top \tilde{k}_n = q_m^\top \left(R_\Theta^{(m)}\right)^\top R_\Theta^{(n)} k_n = q_m^\top R_\Theta^{(n-m)} k_n$$

"Taylor" and "Swift" at position 10 and 11 will get the exact same result as "Taylor and "Swift" at 位置 100 和 101！

While their corresponding rotation matrices are all different, by the time q and k are multiplied, the rotational angle becomes identical! 😍v

ＲｏＰＥ：旋转位置嵌入 （Ｓｕ ｅｔ ａｌ．，２０２１）

特性	原始正弦 （２０１７）	**RoPE (LLaMA, PaLM, etc.)**
集成	加性 : Added to the input 第一层之前的嵌入	乘性 : Applied to Q and K at 每个注意力层
Norm 保持	增加 the token embedding’s norm	保持 the token embedding’s norm
Relative 距离	Mathematically captured, but due to the additive integration, the information gets muddled up by the ＱＫＶ 注意力计算时	Mathematically captured via the rotation angle and the information is cleanly available 当 ＱＫＶ 注意力被计算时
外推	显著困难	Allows for additional long context extension 如 " 位置插值 ”, "NTK scaling”, and YARN

Long context extension practices 🧵

１．　位置编码修改

ＲｏＰＥ　缩放似乎是最常见的方法族（截至　２０２６　年）：

- Linear interpolation (Position Interpolation): Scales position indices down by a factor so the model sees familiar relative positions. Introduced by Meta for extending LLaMA from 2K→３２Ｋ＋。需要轻量微调。
- NTK-aware interpolation: Adjusts the rotary base frequency rather than linearly compressing positions. Better preserves high-frequency components. Has "dynamic NTK" 根据推理时序列长度自适应调整缩放的变体。
- YaRN (Yet another RoPE extensioN): Combines NTK scaling with an attention temperature correction and a ramp function that treats different frequency bands 不同方式处理——对低频进行插值而对高频进行外推。

Long context extension practices 🧵

２．　渐进式 / 分阶段训练

一个常见方案（ＬｏｎｇＬＬａＭＡ、ＬｏｎｇＡｌｉｇｎ、Ｌｌａｍａ　３．１　等使用）：

- 从基础模型开始（如　４Ｋ - ８Ｋ　上下文）。
- 应用位置编码修改。
- 持续预训练　　on long documents with progressively increasing sequence lengths (e.g., 8K → 32K → 64K → 128K), often with a relatively small amount of data (billions of ｔｏｋｅｎ，而非万亿级）。
- 在长上下文指令数据上微调。
- The key insight is that you don't need the full pretraining budget—typically 0.1–1% of 的原始预训练　ｔｏｋｅｎ　就足以适应。

Long context extension practices 🧵

3．　数据工程

- 上采样长文档　　in the continued pretraining mix (books, code repos, 长篇文章、拼接的相关文档）。
- 合成长上下文任务　　: Needle-in-a-haystack retrieval, long-range QA, 多文档摘要。
- **LongAlign-style** instruction tuning with tasks specifically requiring the model to use 分布在完整上下文中的信息。
- 长上下文自指令　　: Using a capable model to generate long-context instruction-response pairs.

Long context extension practices 🧵

4．　注意力架构修改

- 稀疏 / 滑动窗口注意力 combined with global attention tokens (Longformer-style).
- **Flash Attention** (and FlashAttention-2/3) to reduce memory from $O(n^2)$ to $O(n)$ during 训练。
- **Grouped Query Attention (GQA)** or **Multi-Query Attention (MQA)**: Reduces KV cache 内存，在推理时启用更长的上下文。

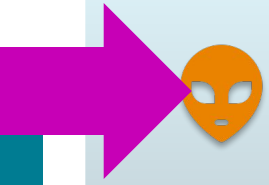
课程计划

推测解码 （20 分钟）

Off-policy 漂移与 on-policy 蒸馏（20 分钟）

Off-policy、on-policy、在线蒸馏 基础设施与 off-policy 漂移
On-policy 蒸馏

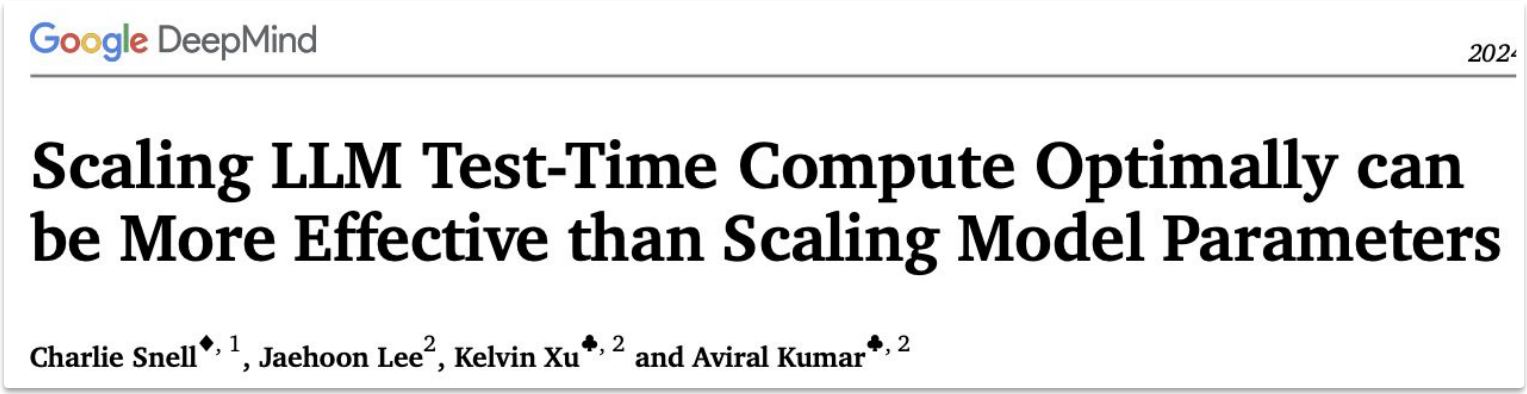
长上下文扩展 （20 分钟）



推理时间扩展 （10 分钟）

Test time compute scaling 🚀

- This notion was intensely popularized by OpenAI's O1 release in Sep 2024
- 尽管 GDM 在 2024 年 8 月的论文更早提出
- And it also appeared less explicitly in the "Let's verify step by step" paper from May 2023

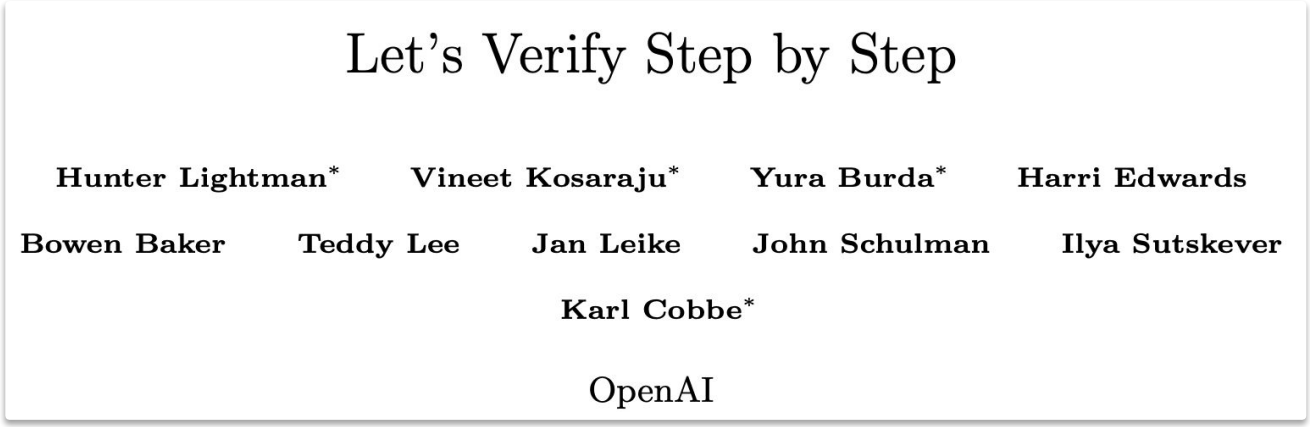


Google DeepMind

202

Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters

Charlie Snell[♦, 1], Jaehoon Lee[2], Kelvin Xu[♣, 2] and Aviral Kumar[♣, 2]



Let's Verify Step by Step

Hunter Lightman* Vineet Kosaraju* Yura Burda* Harri Edwards

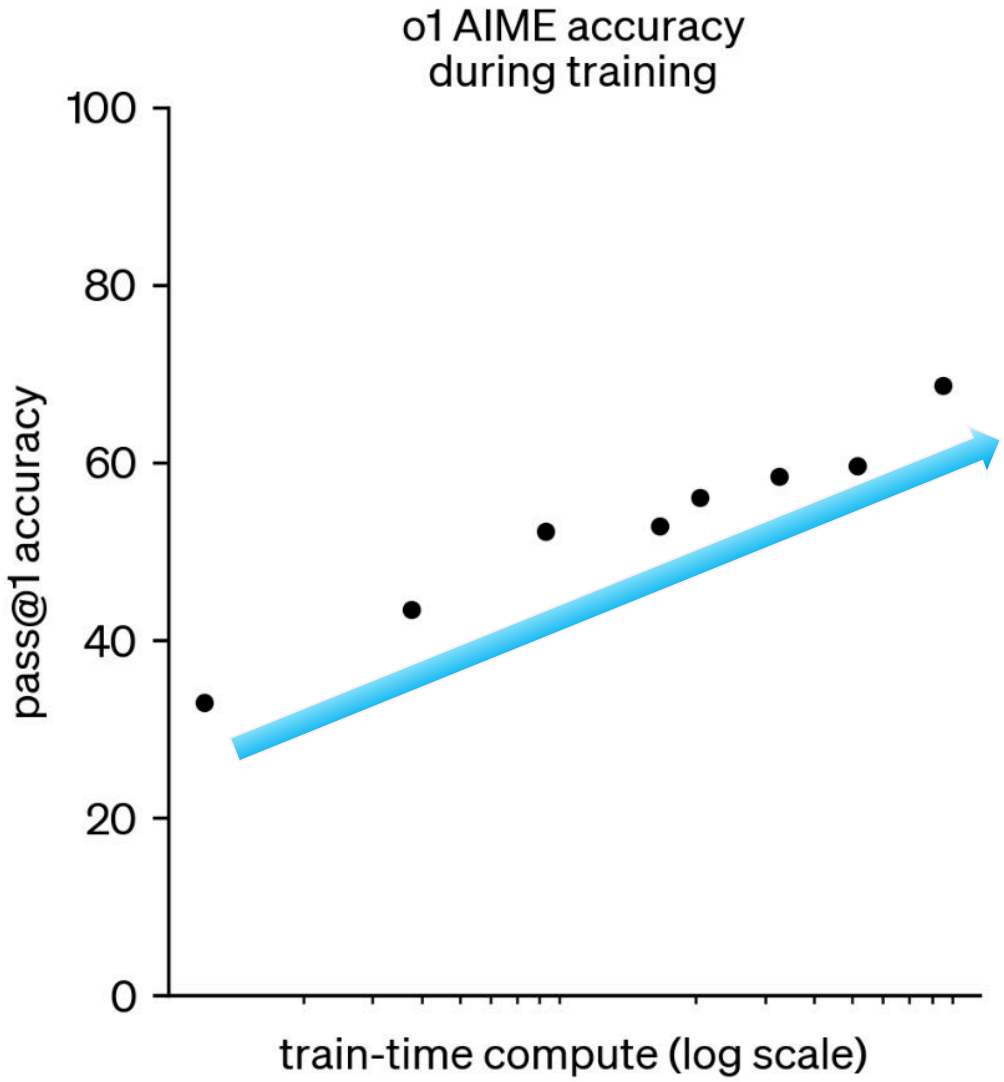
Bowen Baker Teddy Lee Jan Leike John Schulman Ilya Sutskever

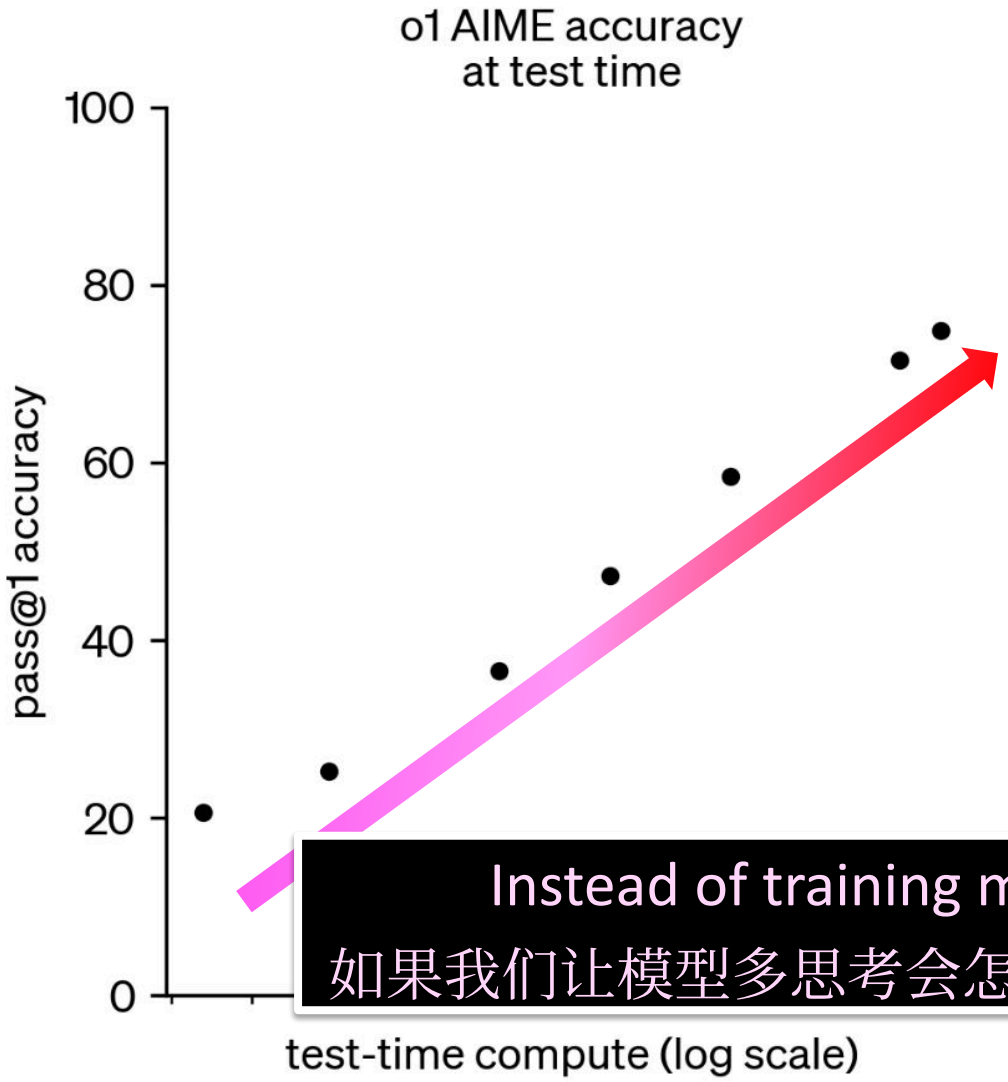
Karl Cobbe*

OpenAI

Test time compute scaling 🚀

Instead of scaling only training time compute,
如果我们扩展测试时间计算会怎样？





Instead of training more,
如果我们让模型多思考会怎样？

Test time compute scaling 🚀

Google DeepMind 2024

Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters

Charlie Snell[♦, 1], Jaehoon Lee[2], Kelvin Xu[♣, 2] and Aviral Kumar[♣, 2]

- This paper challenges the dominant paradigm in LLM development by demonstrating that intelligently scaling test-time compute can yield larger performance gains than simply scaling model parameters
- For a fixed inference budget, smaller models using smart test-time compute strategies can 超过使用标准解码的更大模型
- For compute-optimal allocation, seek an optimal balance between
  - Model size (pretraining FLOPs)
  - Number of generated samples or revision steps (test-time FLOPs)

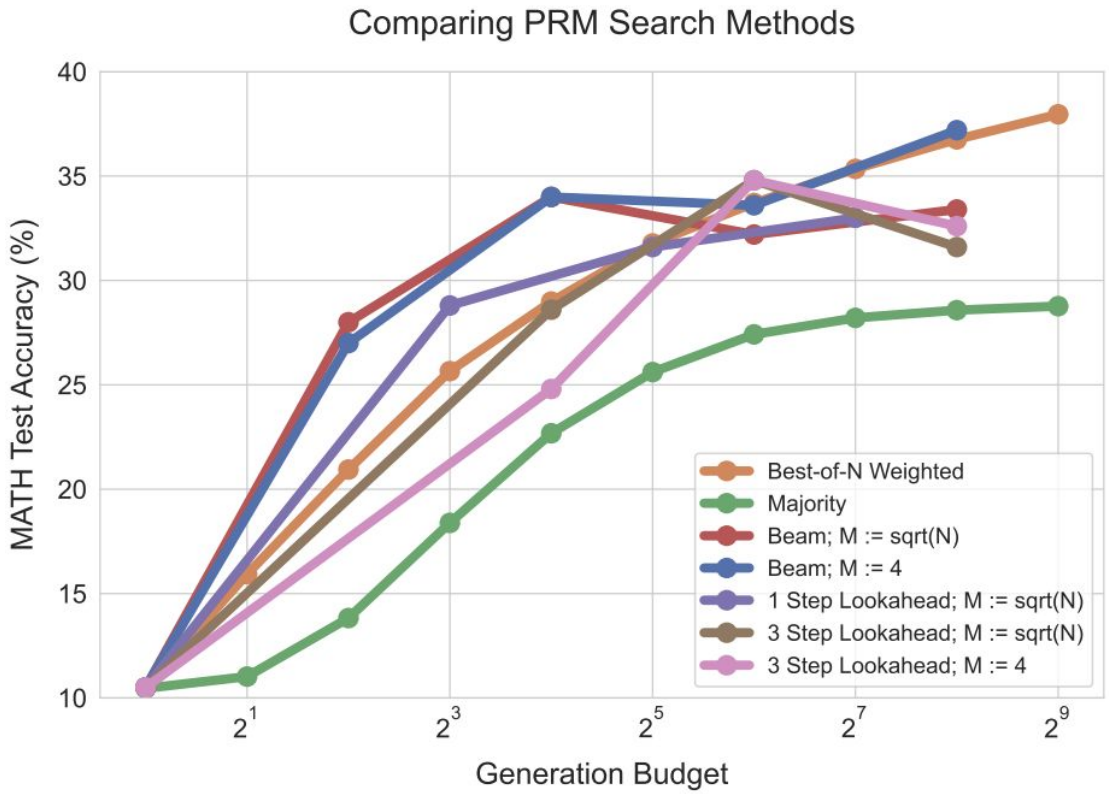
Test time compute scaling 🚀

Google DeepMind 2024

Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters

Charlie Snell[♦, 1], Jaehoon Lee[2], Kelvin Xu[♣, 2] and Aviral Kumar[♣, 2]

- 评估的测试时间计算扩展策略：
  - **Best-of-N sampling**： 生成 N 个独立解，通过验证器选最优
  - **Weighted Best-of-N**： 从计算最优的温度分布中采样
  - **Sequential revisions**： 使用模型自我批评迭代改进单个解
  - **Beam search with PRMs**: Maintain multiple solution candidates, pruning based on 步级奖励
  - **Diverse beam search**： 鼓励探索不同的解题方法
- 评估的验证方法：
  - Outcome-supervised reward models (ORMs): Trained to predict solution correctness
  - Process-supervised reward models (PRMs): Trained to evaluate correctness at each 推理步骤
  - Domain-specific verifiers: For problems with checkable solutions (e.g., code execution)



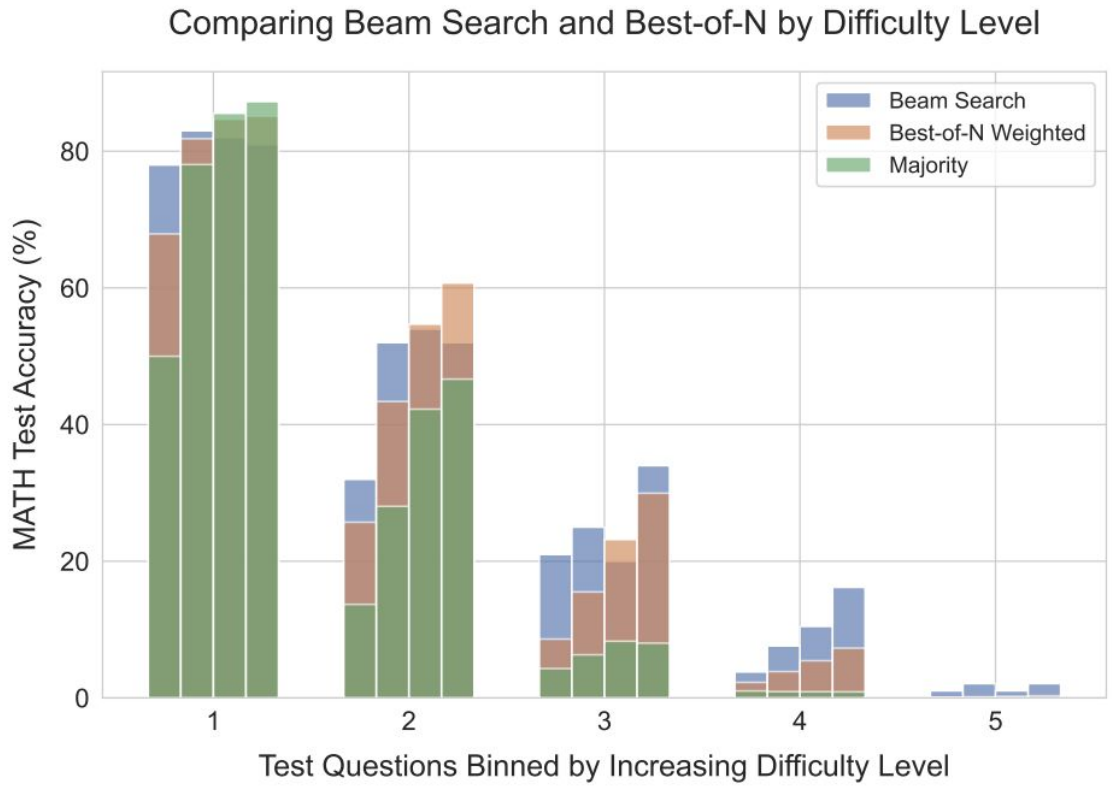


Figure 3 | **Left:** ***Comparing different methods for conducting search against PRM verifiers***. We see that at low generation budgets, beam search performs best, but as we scale the budget further the improvements diminish, falling below the best-of-N baseline. Lookahead-search generally underperforms other methods at the same generation budget. **Right:** ***Comparing beam search and best-of-N binned by difficulty level***. The four bars in each difficulty bin correspond to increasing test-time compute budgets (4, 16, 64, and 256 generations). On the easier problems (bins 1 and 2), beam search shows signs of over-optimization with higher budgets, whereas best-of-N does not. On the medium difficulty problems (bins 3 and 4), we see beam search demonstrating consistent improvements over best-of-N.

Test time compute scaling 🚀

Google DeepMind 2024

Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters

Charlie Snell[♦, 1], Jaehoon Lee[2], Kelvin Xu[♣, 2] and Aviral Kumar[♣, 2]

- 关键发现：
  - On MATH-500, a 14B parameter model with optimized test-time compute matched or 超过了 4 倍大模型的性能
  - The compute-optimal strategy allocated roughly equal FLOPs to pretraining and 在他们的实验设置中进行推理
  - Sequential revision strategies showed particularly strong scaling on tasks requiring 改进和错误修正
  - PRMs enabled 4-8× more efficient compute usage compared to ORMs w/ beam search
- 实用要点：
  - Don't default to the largest model!
  - Invest in verification!
    - High-quality reward models, especially PRMs, dramatically improve test-time scaling