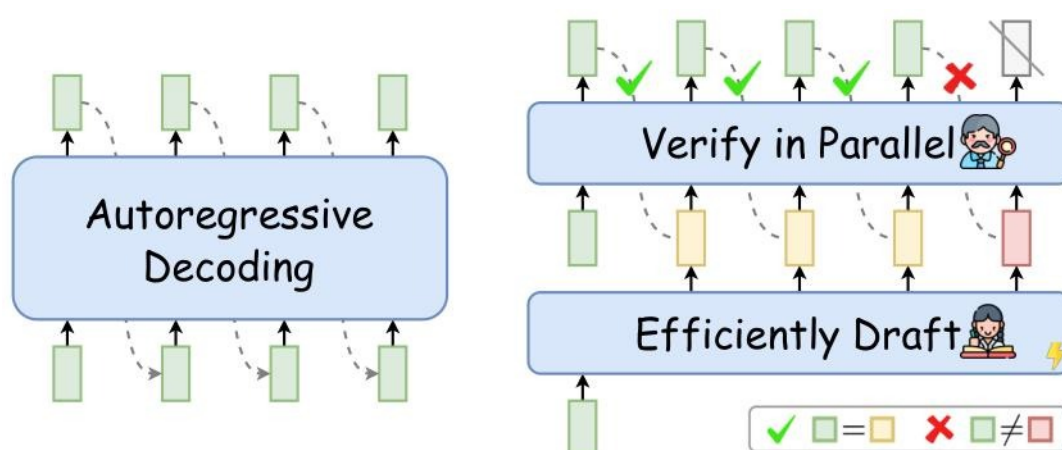


投机解码Speculative Decoding

大家好，我是居丽叶。

今天来看一篇综述《Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding》。这篇文章系统梳理了投机解码（Speculative Decoding）的核心思想——“先预测（draft）后验证（verify）”。一起来看看吧！



题图来自论文Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding

背景：为什么需要加速 Decode

大模型推理一般分为两个阶段：**Prefill** 和 **Decode**。前者并行度高，效率问题不大；真正的瓶颈出在后者——自回归解码。传统的做法是每次解码只生成一个 token，推理延迟随生成序列长度和模型规模线性增长，算力利用率不高，显存带宽压力又大，即便有 KV cache 也治标不治本。

于是人们开始尝试各种加速方案，但这些方法又各有各的局限，例如：

- **Continuous batching**：即已经解码完毕的任务可以退出batch，允许新的请求插入到batch中，减少了因等待导致的算力浪费。但在用户请求少时无法发挥作用。
- **Parallel Decoding**：通过**Blockwise Decoding**，在Transformer解码器顶部添加额外的FFN头，使得每步推理可以同时生成多个token。这些token随后由原始LLM并行验证，接受第一个不一致的token之前的所有token，确保输出与原始LLM的输出一致。但额外的 FFN 头预测能力有限，效果一般。

在这些尝试之上，**投机解码（Speculative Decoding）** 带来了更灵活的思路：

先预测（draft）、再验证（verify）。具体做法是用轻量的 drafter 模型预测多个未来的 token，然后使用 LLM 并行验证，只有符合 LLM 验证标准的 token 才会被接受为最终输出。

之所以可以采用投机解码，主要基于两个观察：

- 许多简单的 token 使用简单的模型就能预测
- 解码是**通信密集型任务**，每次解码都需要将大量参数从 HBM 加载到 SRAM 中。利用投机解码可以减少解码次数，从而减少通信次数，提升推理效率

这种方法保证了生成质量，同时通过减少大模型的解码次数，大幅提升了推理效率。

公式化定义

在继续介绍解码方法的细节之前，先给出两种解码方式的公式化定义：

自回归解码

给定输入序列为 x_1, \dots, x_t ，目标自回归模型为 M_q ， q 为 M_q 模型给出的条件概率分布， x_{t+1} 从 q_{t+1} 中采样：

$$x_{t+1} \sim q_{t+1} = \mathcal{M}_q(x \mid x_{<t+1}), \quad (1)$$

自回归解码每次只生成一个 token，不能充分利用GPU的算力，受到内存带宽限制。

投机解码

在投机解码的每一步中，先用轻量的drafter模型 M_p 并行预测K个token，然后由verify模型 M_q 并行验证这些预测token，从而提升推理吞吐量。草稿模型 M_p 可以是与目标模型 M_q 相互独立、专门训练的小模型；也可以在 M_q 内部结构中构造，通过添加轻量的前馈预测头（FFN heads）或在浅层提前终止计算（early exit），以较低代价自举生成候选 token 序列。

如下右侧的 **Algorithm 2** 给出了完整的伪代码实现，可以清晰看到 **draft** \rightarrow **verify** \rightarrow **correct** 的循环流程。

核心步骤

- **Drafting**：给定输入序列 x_1, \dots, x_t ，目标大模型为 M_q ，利用轻量的模型 M_p 并行预测 K 个 token：

$$\begin{aligned} p_1, \dots, p_K &= \text{DRAFT}(x_{\leq t}, \mathcal{M}_p), \\ \tilde{x}_i &\sim p_i, \quad i = 1, \dots, K, \end{aligned} \quad (2)$$

其中 p 是由 M_p 给出的概率分布， \tilde{x}_i 表示从 p_i 中采样的drafted token。

- **Verification**：给定输入序列 x_1, \dots, x_t 和预测的drafted token $\tilde{x}_1, \dots, \tilde{x}_K$ ，使用 M_q 并行计算K+1个条件分布：

$$q_i = \mathcal{M}_q(x \mid x_{\leq t}, \tilde{x}_{<i}), i = 1, \dots, K + 1. \quad (3)$$

drafted token使用验证准则

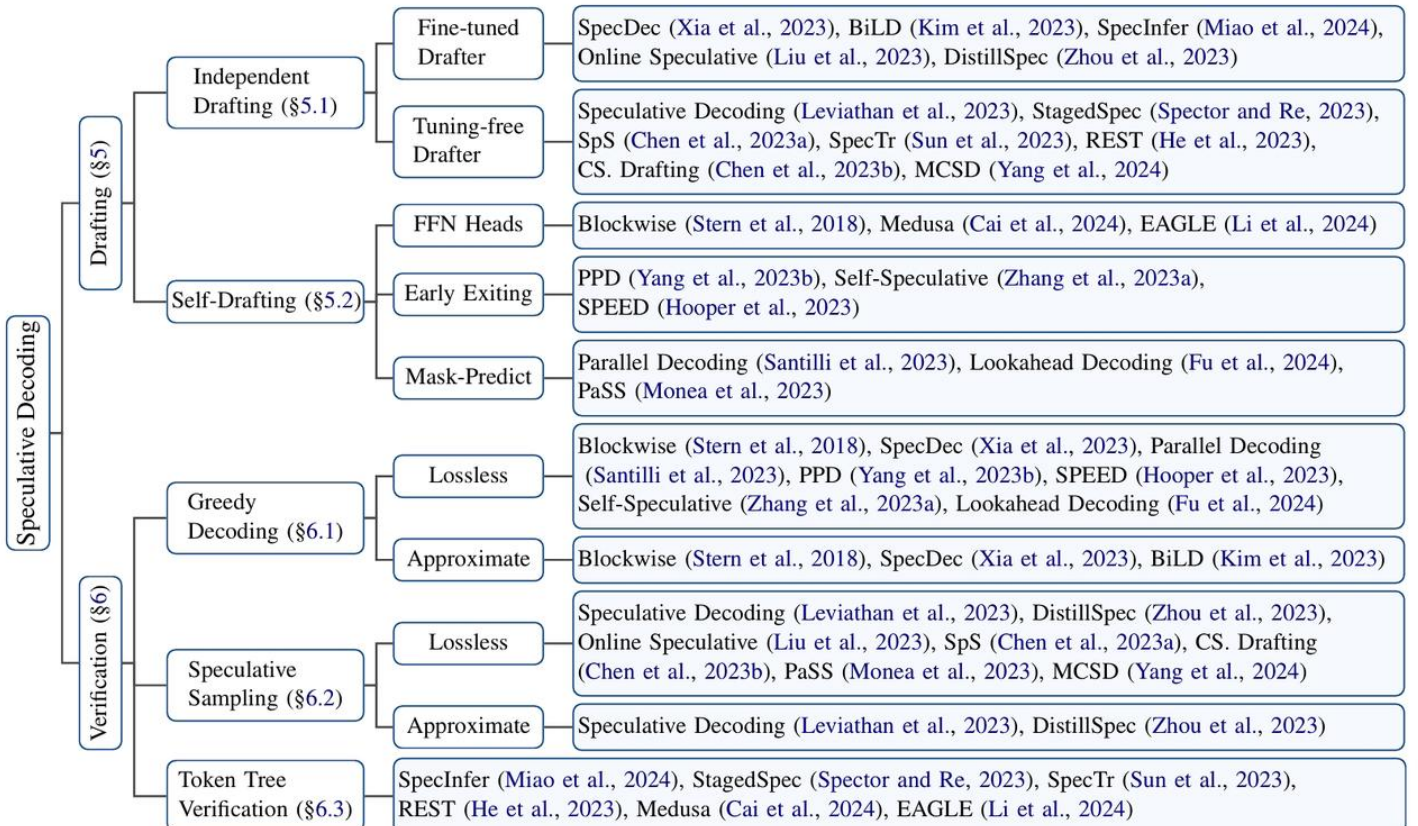
$VERIFY(\tilde{x}_i, p_i, q_i)$ 验证是否与目标模型的输出一致，假设drafted token \tilde{x}_c 不正确，则使用 $CORRECY(p_c, q_c)$ 纠正这个token， \tilde{x}_c 之后的token被丢弃以保证输出的高质量。如果所有token都验证成功，第K+1个token从分布 q_{K+1} 中得到。这样就确保每次都生成至少一个新的token，运行次数就不会大于自回归解码时运行的次数。

Algorithm 2 Speculative Decoding

Require: Target language model \mathcal{M}_q , draft model \mathcal{M}_p , input sequence x_1, \dots, x_t , block size K , target sequence length T , drafting strategy DRAFT, verification criterion VERIFY, and correction strategy CORRECT;

- 1: **initialize** $n \leftarrow t$
- 2: **while** $n < T$ **do**
 - // Drafting: obtain distributions from \mathcal{M}_p efficiently*
 - 3: **Set** $p_1, \dots, p_K \leftarrow \text{DRAFT}(x_{\leq n}, \mathcal{M}_p)$
 - // Drafting: sample K drafted tokens*
 - 4: **Sample** $\tilde{x}_i \sim p_i, i = 1, \dots, K$
 - // Verification: compute $K+1$ distributions in parallel*
 - 5: **Set** $q_i \leftarrow \mathcal{M}_q(x \mid x_{\leq n}, \tilde{x}_{< i}), i = 1, \dots, K+1$
 - // Verification: verify each drafted token*
 - 6: **for** $i = 1 : K$ **do**
 - 7: **if** $VERIFY(\tilde{x}_i, p_i, q_i)$ **then**
 - 8: **Set** $x_{n+i} \leftarrow \tilde{x}_i$ and $n \leftarrow n + 1$
 - 9: **else**
 - 10: $x_{n+i} \leftarrow \text{CORRECT}(p_i, q_i)$
 - 11: **and Exit for loop.**
 - 12: **end if**
 - 13: **end for**
 - 14: **If** all drafted tokens are accepted, sample next token $x_{n+1} \sim q_{K+1}$ and set $n \leftarrow n + 1$.
- 15: **end while**

上面介绍的是投机解码的基本范式。实际上，已有研究在 **Drafting（草稿生成）** 和 **Verification（验证）** 两个环节都提出了大量改进方法。下图对这些工作做了分类整理，帮助我们从全局把握这一领域的研究脉络。



投机解码的加速效果取决于 **接受率**，即 drafter 预测的 token 有多少能通过大模型验证。接受率越高，推理速度越快。它受到以下因素影响：

- drafter 的精度与推理速度；
- 验证标准是严格匹配还是近似匹配；
- drafter 与目标模型分布的一致性。

为了进一步理解这些因素，我们接下来将从 Drafting、Verification、Alignment 三个环节来做进一步研究。

Drafting：如何生成草稿

在 Drafting 阶段，研究主要有两条路线：**独立 drafter** 和 **自举 drafter**。

Independent Drafting

- **核心思想**：用一个与目标 LLM 不同、但更高效的小模型来并行生成候选 token，典型方式包括：
 - a. **专用非自回归 Transformer**：例如 SpecDec 提出的 deep-shallow encoder-decoder 结构（深层编码器 + 少量解码器），需要额外训练，但能高效生成多步候选。
 - b. **同系列小模型**：直接使用与目标模型同系列的轻量模型（共享 tokenizer 和预训练数据，结构相似），无需额外训练，天然更易对齐；若结合 **知识蒸馏**，可进一步提高接受率。

Self-Drafting

- **核心思想**：直接在目标 LLM 内部生成草稿，避免额外训练和调度开销。代表方法有：
 - a. **FFN Heads**：在 Transformer 解码器上添加并行 FFN 头（如 **Blockwise Decoding**、**Medusa**），一次性生成多个 token，开销小，适合分布式部署。
 - b. **Early Exiting / Layer Skipping**：在推理时提前退出或跳过中间层（如 **Self-Speculative**），加速生成后续分布。
 - c. **Mask-Predict & N-grams**：在输入末尾追加多个 [PAD]，通过并行掩码填充预测多个 token，并将低质候选转化为 n-gram 片段以提升质量。
 - d. **Learnable Tokens**：引入可训练的特殊 token（如 [LA]），通过小规模微调改善并行预测效果。

Verification：如何批改草稿

在 Verification 阶段，目标模型需要判断 drafter 生成的候选 token 是否可以被接受。不同方法在严格程度和灵活性上有所不同，大体可以分为三类：

贪婪解码

早期的投机解码多采用贪婪解码，以保证输出与目标模型的贪婪解码完全一致。

- **严格匹配 (Lossless)**：要求 drafter 的预测 token 必须等于目标模型 M_q 的 Top-1 结果，即 $\tilde{x}_i = \arg \max q_i$ 。

若第 c 个 token 不符，则用目标模型的 Top-1 结果替换，并舍弃后续 token。代表方法如 **Blockwise Decoding**。缺点是可能拒绝掉一些“质量高但不是 Top-1”的预测，限制加速效果。

- **近似匹配 (Approximate)**：为提高接受率，可以放宽条件。例如 **SpecDec** 允许 drafter 的预测落在目标模型的 Top-k 内；**BiLD** 则只在连续不匹配的 token 超过阈值时才拒绝。

投机采样

相比硬性匹配，投机采样引入概率接受机制。第 i 个预测 token 的验证标准为：

$$r < \min \left(1, \frac{q_i(\tilde{x}_i)}{p_i(\tilde{x}_i)} \right), \quad r \sim U[0, 1]$$

其中 $q_i(\tilde{x}_i)$ 和 $p_i(\tilde{x}_i)$ 分别表示目标模型和 drafter 对该 token 的预测概率。

- 如果通过，则接受该 token；
- 如果拒绝，则按 $x_{t+c} \sim \text{norm}(\max(0, q_c - p_c))$ 的分布重新采样。

这样做理论上可以保证最终分布与目标模型一致。

Token 树验证

为进一步提升并行度和接受率，部分方法（如 **SpecInfer**、**Medusa**）利用共享前缀将多条候选序列合并为一棵 **token 树**。目标模型通过设计 **树注意力掩码** 来并行验证整棵树，大幅减少重复计算。后续会通过 Medusa 这篇论文讲解树注意力。

Alignment：如何提升接受率

投机解码的加速效果在很大程度上取决于 **接受率**——也就是 drafter 的预测分布与目标模型分布的一致性。接受率越高，通过验证的 token 就越多，推理速度也就越快。因此，研究者提出了多种 **对齐 (alignment) 策略** 来缩小两者差距，提升接受率。

主要方法包括：

- **序列级知识蒸馏 (Seq-KD)**：在目标模型生成的序列上训练 drafter，使其输出更贴近大模型分布。
- **集体增强微调 (Col-BT)**：对多个小模型应用 Seq-KD，并利用聚合输出进行预测，从而提升整体准确性。
- **在线知识蒸馏 (Online KD)**：在推理过程中基于实时查询动态更新 drafter，无需预训练，即时对齐目标模型。

下表对现有方法做了进一步总结，展示了不同算法在 **Drafting**、**Verification**、**Alignment** 等方面的设计选择及加速效果。

Methods	Drafting			Verification			Target LLM	Speedup (reported)
	Approach	Alignment	Tuning-free	Greedy	Sampling	Token Tree		
<i>Independent-D</i>	SpecDec (Xia et al., 2023)	Non-Auto LM	Seq-KD	✗	✓	✗	Transformer-base (65M)	3.9× ~ 5.1×
	SpS (Chen et al., 2023a)	Small LM	-	✓	✓	✗	Chinchilla (70B)	1.9× ~ 2.5×
	SpecInfer (Miao et al., 2024)	Boost-tuned LMs	Col-BT	✗	✓	✓	LLaMA (30B-65B)	2.0× ~ 2.4×
	DistillSpec (Zhou et al., 2023)	Small LM	KD	✗	✓	✗	T5-XL (3B)	-
	Online Speculative (Liu et al., 2023)	Small LM	Online-KD	✗	✓	✗	Vicuna (7B)	-
	CS. Drafting (Chen et al., 2023b)	Cascaded LMs	-	✓	✓	✗	FLAN-T5-xxl (11B)	-
	REST (He et al., 2023)	Context Retrieval	-	✓	✓	✓	Vicuna (7B-13B)	1.6× ~ 1.8×
<i>Self-D</i>	Blockwise Decoding (Stern et al., 2018)	FFN Heads	Seq-KD	✗	✓	✗	Transformer-big (213M)	1.7× ~ 3.0×
	Medusa (Cai et al., 2024)	FFN Heads	Seq-KD	✗	✓	✓	Vicuna (7B-13B)	2.2× ~ 2.3×
	PPD (Yang et al., 2023b)	Early Exiting	-	✗	✓	✗	Vicuna (13B)	1.1× ~ 1.5×
	Self-Speculative (Zhang et al., 2023a)	Layer Skipping	-	✓	✓	✗	LLaMA-2 (13B-70B)	1.4× ~ 1.7×
	Parallel Decoding (Santilli et al., 2023)	Mask-Predict	-	✓	✗	✗	MBart50 (610M)	1.0× ~ 1.1×
	Lookahead Decoding (Fu et al., 2024)	Mask-P & N-grams	-	✓	✗	✗	LLaMA-2 (7B-70B)	1.5× ~ 2.3×
	EAGLE (Li et al., 2024)	Auto-regression Head	KD	✗	✓	✓	Vicuna (7B-33B)	2.9× ~ 3.1×

Table 3: Summary of Speculative Decoding methods. “*Independent-D*” and “*Self-D*” denote independent drafting and self-drafting, respectively. “*Greedy*”, “*Sampling*”, and “*Token Tree*” denote whether the method supports greedy decoding, speculative sampling, and token tree verification, respectively. We list the most representative target LLMs for each method and the speedups in the original paper (if reported), which is obtained with a batch size of 1.

总结与展望

投机解码通过“草稿 + 批改”的策略，在不牺牲生成质量的前提下显著提升推理效率。它已经展现出巨大的应用潜力，但仍面临挑战：

- 如何在预测精度与延迟之间找到最佳平衡？
- 如何与批量推理结合？
- 是否能推广到多模态等更复杂场景？

这些问题为未来研究留下了空间，也让投机解码成为大模型推理优化的一个重要方向。