arXiv:2603.08088v1 [cs.LG] 9 Mar 2026

EAGLE-Pangu：昇腾 NPU 上的加速器安全树式推测解码

Chang Han,Yijie Hu,Jingling Liu重庆大学计算机科学学院

摘要

自回归解码仍是大语言模型（LLM）服务中的主要瓶颈，这促使人们采用推测解码方法，通过在每一步验证多个候选词元来减少代价高昂的教师模型调用。树式推测进一步提升了并行性，但在跨异构后端和加速器软件栈移植时往往较为脆弱，因为注意力掩码、KV 缓存布局和索引语义并不可互换。

我们提出 EAGLE-PANGU，这是一个可复现系统，将 EAGLE-3 风格的树式推测解码移植到昇腾 NPU 上的 Pangu 教师后端。EAGLE-PANGU 的贡献包括：（i）基于 Cache API 构建的显式分支/提交缓存管理器；（ii）加速器安全的树张量化，从构造上消除未定义的负索引并验证结构不变量；（iii）与融合内核兼容的教师验证路径，并提供可调试的即时执行回退。

在来自 MT-Bench 和 HumanEval 风格提示的 240 轮对话中，在融合内核性能路径下，与仅教师模型贪心解码相比，EAGLE-PANGU 的端到端解码吞吐量平均提升 $1.27\times$（p99 下最高提升 $2.46\times$）。我们还提供了一条不依赖融合内核的参考路径，包含结构化跟踪记录和不变量检查，支持在不同执行模式和树预算下进行可复现的调试与消融实验。

1 引言

大语言模型服务的主要成本来自自回归解码，因为每生成一个新词元，都需要对大型“教师”模型额外执行一次前向传播。这种序列依赖从根本上限制了吞吐量，并在实际部署约束下（例如长上下文、高并发和严格的尾延迟预算）增加了延迟。推测解码通过使用较小的“草稿模型”提出候选词元，再由教师模型进行验证，从而减少生成固定长度文本所需的教师模型步数 [1]。树结构推测解码进一步推广了这一思路：每一步验证多个候选延续，从而提高并行性，并有望使吞吐量超越线性草稿提议所能达到的水平 [2, 3]。

尽管概念上很简单，树式推测解码在实践中往往十分脆弱。将树解码实现移植到不同的模型后端和加速器软件栈，并非只是“重新实现内核”这么简单：(i) 教师模型可能会采用非标准的 KV缓存布局和专用的注意力掩码接口，尤其是在支持树模式执行时；(ii) 与即时执行实现相比，融合注意力内核对掩码形状、对齐和边界条件的要求更为严格；(iii) 不同设备运行时的索引和收集语义可能各不相同，其中负索引或越界索引可能不受支持、处理方式不一致，或被静默地错误计算。在树解码中，这些差异会被放大，因为候选评估依赖结构化索引（例如，沿多条推测路径检索节点状态），还依赖

预印本。

精确的掩码处理，以防止跨分支信息泄漏。因此，即便“能够运行”，简单的移植也可能导致质量下降、偶发故障或无法复现的性能。

在本工作中，我们致力于让树式推测解码在部署于昇腾 NPU 上的盘古教师后端中具备可移植性、正确性和可复现性。我们采用 EAGLE-3 风格的树式推测解码框架及其后验接受规则 [2], ，并不声称提出了新的解码算法。相反，我们指出了这一场景下关键的系统级故障模式，并提出一系列设计原则和抽象，在满足加速器约束的同时保留预期的解码语义。我们的评估重点关注：(a) 端到端解码吞吐量；(b) 在解码配置一致的条件下的答案质量；(c) 借助结构化跟踪记录和执行模式控制实现的稳定性与可调试性。

具体而言，我们的贡献如下：

- **可分支 KV 缓存抽象**。我们提出一种缓存接口，将已接受的前缀状态与逐分支推测状态分离，从而能够正确执行缓存克隆和接受状态更新，同时使树解码与特定后端的 KV 表示解耦。
- **加速器安全的树张量语义**。我们提出一种保持正确性的索引方案，用有效的虚拟索引替代未定义的负填充索引，并在执行收集操作之前应用安全映射；同时辅以不变量检查，以检测形状和边界违规。
- **采用融合注意力的树掩码教师模型执行**。 我们将 4D 树形注意力掩码形式化并集成到教师模型执行路径中，在保留用于验证和调试的即时执行回退机制的同时，支持使用融合注意力内核来提升吞吐量。
- **可复现的分布式流水线（辅助贡献）**。 我们提供分布式预处理和缓存机制，避免重复工作，并减少大规模运行中的同步故障。
- **数据驱动的词汇子集映射（辅助贡献）**。 我们实现了可复用的草稿模型词汇子集构建与缓存工作流，以支持草稿建模中可控的速度与质量权衡。

我们使用 MT-Bench 风格的对话提示词对 EAGLE-PANGU 进行基准测试，采用一致的提示模板和匹配的解码设置，以确保比较公平。我们报告相对于标准贪心生成的每秒令牌数和端到端实际运行时间加速比，并依据相应的基准测试评分协议评估质量。为单独考察各项设计的影响，我们针对以下方面开展了专项消融实验：(i) 缓存管理策略，(ii) 安全索引与不变量强制检查，以及 (iii) 融合注意力与即时注意力执行；此外，还分析了树形超参数和草稿词汇子集大小。

本文其余部分安排如下：第2节回顾推测解码和树形验证语义；第3节介绍我们的缓存抽象、加速器安全索引和融合式教师验证；第4节介绍实现和评估框架；第5节报告实验结果并进行分析；最后，第6节总结全文并讨论局限性。

2 背景

本节回顾推测解码及其树形变体，并确立全文使用的符号。我们还重点介绍执行层面的语义——KV 缓存、注意力掩码和张量索引；当树解码需要移植到异构后端和加速器软件栈时，这些语义便成为首要关注事项。

2.1 自回归解码与 KV 缓存

令 $x = (x_1, \ldots, x_T)$ 表示一个词元序列。自回归大语言模型定义了一个分布

$$p_\theta(x) = \prod_{t=1}^{T} p_\theta(x_t \mid x_{<t}),$$

而解码则通过采样或选择的方式依次生成词元

$$x_t \sim p_\theta(\cdot \mid x_{<t}) \quad \text{or} \quad x_t = \arg\max_v p_\theta(v \mid x_{<t}).$$

这种序列依赖意味着，生成 T 个词元需要 T 个教师前向步骤，这通常是端到端服务延迟的主要来源，并限制了吞吐量。

现代 Transformer 解码器通过键值（KV）缓存分摊计算。对于前缀 $x_{\le t}$，缓存会存储各层的键/值张量，用以概括与注意力相关的历史信息，使模型无需对所有过去的词元重新计算注意力，便可计算 $p_\theta(\cdot \mid x_{\le t})$。我们将教师模型的状态转移抽象地写作

$$(\ell_t, \mathcal{C}_t) = \mathrm{TeacherStep}(x_t, \mathcal{C}_{t-1}),$$

其中，$\ell_t \in \mathbb{R}^{|\mathcal{V}|}$ 是下一个词元的对数几率，$\mathcal{C}_t$ 是更新后的 KV 状态。

2.2 推测解码

推测解码通过配合使用大型教师模型 p_θ 与成本较低的草稿模型 q_ϕ [1]，减少教师模型调用。简单来说，草稿模型会提出多个候选词元，教师模型则以批处理方式对其进行验证。如果提出的词元与教师模型的分布一致，每个教师模型步骤就可能接受多个词元，从而提高吞吐量。

具体而言，在给定前缀 $x_{\le t}$ 时，草稿模型使用 $q_\phi(\cdot \mid \cdot)$生成一个提议序列 $y_{1:K} = (y_1, \ldots\ldots, y_K)$。随后，教师模型会评估所提议的后续序列，从而得到沿该提议序列各位置的教师概率。一个通用的推测解码步骤可以视为生成一个接受长度 $A \in \{0, \ldots, K\}$，并将前缀更新为

$$x_{\le t+A} \leftarrow x_{\le t} \,\|\, y_{1:A},$$

其中 $\|$ 表示拼接。具体的接受规则（例如拒绝采样方式）经过选择，使所得边缘分布在预期的解码模式 [1]下与教师模型分布相匹配（或近似匹配）。

2.3 树结构推测解码

树结构推测解码将线性提议推广为分支结构，使草稿模型能够并行探索多个后续词元 [2, 3]。草稿模型不再提议单条长度为-K的链，而是构建一棵有根树，其节点表示基于各自祖先上下文的候选下一个词元。

设 $\mathcal{T} = (\mathcal{N}, \pi)$ 是一棵有根树，其节点集为 $\mathcal{N}$，父节点函数为 $\pi\colon \mathcal{N} \setminus \{r\} \to \mathcal{N}$，其中$r$ 为根节点。每个节点 $u \in \mathcal{N}$（根节点除外）对应一个提议词元 $\tau(u) \in \mathcal{V}$。深度 $d(u)$ 是从 r 到 u 的边数。从 r 到 u 的唯一路径定义了一个提议词元序列

$$\mathrm{path}(u) = \big(\tau(u_1), \tau(u_2), \ldots, \tau(u_{d(u)})\big),$$

其中 u_i 是 u的深度为-i 的祖先。给定当前已接受前缀 $x_{\le t}$，每个节点都对应一条候选后续路径 $x_{\le t} \| (u)$。

教师验证步骤会评估这棵树，以获取教师模型在各节点对应的祖先上下文下的对数几率（或概率）。我们将其抽象地表示为

$$\{\ell(u)\}_{u \in \mathcal{N}} \;=\; \mathrm{TeacherTreeEval}(x_{\le t}, \mathcal{T}, \mathcal{C}_t),$$

其中，$\ell(u)$ 提供教师模型在节点 u 处预测 $\tau(u)$ 的评分，$\mathcal{C}_t$ 是已接受前缀的 KV 状态。随后，树接受过程会选出一组符合有效前缀扩展的已接受节点（通常是单条根到叶路径的前缀），并提交相应的词元和缓存更新。

我们的设定与 EAGLE-3 式树式推测解码一致：草稿模型提出一组结构化候选项，教师模型通过一次（或少量几次）批量评估对其进行验证 [2]。我们将在第3节讨论特定于系统的设计选择，尤其是在目标技术栈上确保正确性和可移植性所需的选择。

2.4 树形注意力掩码

正确进行树验证的一项核心要求是防止分支之间发生信息泄漏。在批量张量中评估多个推测节点时，不同分支中的词元

除非彼此存在祖先关系，否则不得相互关注。这通过树注意力掩码来实现。

设批量推测词元排列成长度为 $M = |\mathcal{N} \setminus \{r\}|$（不包括根节点）的张量，并采用固定的节点顺序。对于分别对应节点 u 和 v的两个推测位置，当且仅当 v 位于 u 的祖先路径上（包括其自身）时，词元 u 才可以关注词元 v。将祖先谓词记为

$$\mathrm{Anc}(v, u) = \begin{cases} 1, & \text{if } v \text{ is an ancestor of } u \text{ (or } v = u), \\ 0, & \text{otherwise.} \end{cases}$$

随后可以定义一个概念上的注意力掩码 M_{tree}：将 $-\infty$ 赋给不允许关注的词元对，其他情况赋值为 0。实际中，加速器内核通常需要可广播的四维掩码（例如 $[B, H, M, M]$或 $[B, 1, M, M]$），实现时必须严格保留上述谓词，以确保语义正确。

2.5 树解码中的张量索引与收集语义

树解码高度依赖结构化索引：将每个节点映射到其父节点、获取与祖先对齐的隐藏状态，以及组装逐路径表示以进行评分和缓存更新。这些操作通常会编译为加速器上的收集/索引原语。值得注意的是，在某些框架中无害的索引语义（例如将负索引用作填充哨兵，或隐式回绕）在其他框架中可能未定义或不受支持，并可能导致静默的错误计算。

一种常见模式是将父指针表示为整数数组 parent$[u]$，其中根节点使用哨兵值（通常为 -1）表示“无父节点”。构造批量张量时，哨兵值可能会传播到收集索引中，除非显式进行清理。在负索引无效的加速器运行时上，这可能触发运行时错误；更糟的是，在某些技术栈中还可能产生未指定的值。因此，要确保正确性，索引方案既要忠实体现树结构的语义，又必须由设备定义所有可能的索引值。

2.6 问题设定与评估视角

我们考虑这样一种部署环境：教师模型通过盘古后端提供服务，并在昇腾 NPU 上运行。这一环境代表了更广泛的一类生产环境技术栈，其特点包括：(i) 注意力内核经过高度融合；(ii) KV 缓存以特定于后端的布局存储；(iii) 掩码处理和索引行为必须满足比即时执行的 GPU 实现更严格的约束。我们的目标是以一种能够 (a) 保持预期解码语义、(b) 实现显著吞吐量提升，并且 (c) 在分布式执行下保持稳定且可复现的方式，实现树式推测解码。

下一节将把这些背景要求转化为具体的系统设计：可分支 KV 缓存抽象、用于索引和 gather 的加速器安全树张量语义，以及兼容融合内核的树掩码教师模型执行路径。

3 方法

本节介绍使树式推测解码能够在我们的目标技术栈上实现可移植且正确运行的核心系统设计。我们重点介绍两个构建模块：（i）一种可分支 *KV* 缓存 抽象，可清晰地区分已提交前缀状态与推测分支状态；（ii）加速器安全的树张量语义，可消除未定义的索引行为，同时保留验证与接受率所需的树结构。第 3.3 节（下节）将在这些抽象的基础上，实现兼容融合内核的树掩码教师模型执行。

3.1 可分支 KV 缓存抽象

树式推测解码对 KV 缓存语义提出了严峻考验：单个已接受前缀必须支持多个推测性延续，且系统必须能够 (a) 在评估分支时不修改已提交状态，以及 (b) 高效提交 与最终接受的词元相对应的缓存。

在实践中，特定后端的缓存布局和内核原地更新会让简单克隆变得脆弱：不正确的状态共享可能在不知不觉中污染分支，而完整深拷贝则会抵消推测解码带来的吞吐量优势。

我们在 HuggingFace Cache 接口之上实现了一个可分支缓存管理器，用于维护已提交缓存 $\mathcal{C}^\star$ （主 _ 缓存）和一组按候选项划分的分支缓存 $\{\mathcal{B}_i\}$ （分支 _ 缓存）。在每次推测迭代中，我们通过复制已提交缓存的状态来创建彼此隔离的分支缓存：

$$\mathcal{B}_i \leftarrow \mathrm{Replicate}(\mathcal{C}^\star), \quad i \in \{1, \ldots, N\},$$

然后，将相应的 $\mathcal{B}_i$ 传入教师模型前向路径，为不同的候选延续执行教师验证；该路径会就地扩展相应分支的缓存。在当前代码库中，为确保稳健性和隔离性，Replicate(·) 通过深拷贝实现。

接受后，我们会通过以下两种与实现相对应的提交模式之一更新已提交缓存：

- **基于长度的提交**。给定接受长度 A，我们通过保留原有的已提交前缀，并仅采用所选分支中新生成的前 A 个步骤，来更新已提交缓存。
- **基于路径索引的提交**。当树接受阶段生成明确的路径_索引映射时，我们会对所选分支缓存重新排序，以重建已提交缓存，从而使下一步的前缀缓存与接受路径完全一致。

我们强制执行以下不变量：

1. **隔离性**。 分支缓存是已提交缓存的独立副本。扩展一个分支不会修改 $\mathcal{C}^\star$ 或其他分支。2. **提交等价性**。 接受后，已提交缓存等价于教师模型在已接受前缀上顺序执行所得到的缓存（使用相同的后端和缓存格式）。3.**后端透明性（通过 Cache API）**。实现仅依赖与后端无关的 Cache 原语（例如 get_seq_length 和 to_legacy_cache/from_legacy_cache），以支持在不同 KV 布局间安全地重建缓存。

按路径索引提交并进行前缀共享快速重排。 实际上，树解码通常只接受一个较短的后缀，同时保留较长的已提交前缀。朴素的基于路径索引的提交会重排整个前缀缓存，造成不必要的内存移动。因此，我们针对路径_索引实现了两种提交路径：

- **完整重排（通用）**。 我们使用 to_legacy_cache/from_legacy_cache，沿序列维度根据路径_索引收集 KV 状态，从而重建缓存。
- **前缀共享快速重排（常见情况）**。当路径_索引保留了已提交前缀的顺序时，我们将前缀部分作为连续切片保留，只对新接受的部分进行收集，然后将两者拼接起来，形成新的已提交缓存。此快速路径由执行标志控制（例如 EA_FAST_CACHE_REORDER），遇到任何边界或形状不一致时，都会安全地回退到完整重排。

缓存管理器本身不会构建树索引。相反，树张量化和接受逻辑会计算路径_索引，将下一步接受前缀序列映射到所选分支缓存中的位置。缓存管理器仅负责应用此映射，为后续解码迭代生成正确的已提交缓存。

3.2 加速器安全的树张量语义

树验证和接受需要频繁对节点索引张量执行收集操作（例如，获取父节点/祖先对齐的隐藏状态、组装逐节点路径特征，以及将教师输出映射回节点 ID）。在某些加速器运行时中，负索引未定义，且某些融合内核要求所有索引在静态上均处于有效范围内。因此，我们设计了一种树

张量化方案，该方案 (i) 保留算法所需的祖先关系，且 (ii) 从构造上保证设备端收集操作使用的每个索引均有效。

节点线性化与基础数组。将推测节点（不包括根节点）按$\{u_1, \ldots, u_M\}$排序，其整数 ID 为 $\mathrm{id}(u_k) = k$。我们存储：

- 父节点$[k] = \mathrm{id}(\pi(u_k))$，其中 $\pi(u_k) \in \{r, u_1, \ldots, u_M\}$；
- 深度$[k] = d(u_k)$；
- $\tau[k] = \tau(u_k)$，节点 u_k 处的候选词元。

许多操作都可以表示为对节点索引张量 $Z \in \mathbb{R}^{M \times D}$ 执行聚集，例如：

$$Z_{\mathrm{par}}[k] = Z[\mathrm{parent}[k]].$$

难点在于，对于父节点为根节点 r 的节点，$\mathrm{parent}[k]$ 未定义。

哑根索引（无需哨兵值的 gather 操作）。我们通过在索引 0 处引入一行哑根，消除所有哨兵父节点。具体而言，我们为节点索引张量分配额外的一行，$\widetilde{Z} \in \mathbb{R}^{(M+1) \times D}$，其中第 0 行表示根状态（已提交前缀）。我们定义一个偏移后的 ID 映射：

$$\widetilde{\mathrm{id}}(r) = 0, \qquad \widetilde{\mathrm{id}}(u_k) = k \ \ (k \in \{1, \ldots, M\}),$$

以及一个有效的父节点数组 $\widetilde{\mathrm{parent}}[k] \in \{0, 1, \ldots, M\}$，使得

$$\widetilde{\mathrm{parent}}[k] = \begin{cases} 0, & \text{if } \pi(u_k) = r, \\ \widetilde{\mathrm{id}}(\pi(u_k)), & \text{otherwise.} \end{cases}$$

随后，所有 gather 操作都使用 $\widetilde{\mathrm{parent}}$ 执行，从而确保设备上的索引均在有效范围内。哑行填入与当前已接受前缀（即树根上下文）对应的表示，确保深度为 1 的节点能正确引用前缀状态。

用于路径结构操作的祖先表。除了直接父节点外，树解码通常还需要数量有限的祖先节点（例如，用于构造逐节点路径特征或生成掩码）。令 $D_{\max} = \max_k$深度$[k]$。我们构建祖先索引表 $A \in \mathbb{Z}^{(D_{\max}+1) \times (M+1)}$，其中：

$$A[0, k] = k, \qquad A[\ell + 1, k] = \widetilde{\mathrm{parent}}(A[\ell, k]).$$

根据构造，$A[\ell, k] \in \{0, \ldots, M\}$ 对所有 ℓ，k均成立，因此后续的 gather（例如 $Z[A[\ell, k]]$）可在加速器上安全执行。

填充与批次语义。对于批次大小 B，每个样本 b 的推测节点数量 M_b可能不同。我们将其填充至 $M_{\max} = \max_b M_b$，并维护一个布尔有效性掩码 $\mathrm{valid}_b[k]$，用于指示节点槽 k 是否处于活动状态。所有数组（parent、depth τ）均使用设备定义的值填充（例如，parent $= 0$、depth $= 0$, $\tau = \mathrm{pad}$），并在后续的掩码处理和评分中通过 valid 进行保护，从而确保填充槽不会影响接受率。

结构不变量（可进行单元测试）。为防止发生未被察觉的结构损坏（例如，树格式错误、后端整数溢出），我们会在启动融合内核之前验证以下不变量：

1. **范围**。对所有 $k \in [1, M]$，$\mathrm{parent}[k] \in [0, M]$ 均满足范围约束。2. **无环性 / 深度一致性**。对于 $k > 0$，$\mathrm{depth}[\mathrm{parent}[k]] < \mathrm{depth}[k]$，并且重复应用 parent 后可在 $\mathrm{depth}[k]$ 步内到达 0。3. **有效性闭包**。若 $\mathrm{valid}[k] = 1$，则 $\mathrm{valid}[\mathrm{parent}[k]] = 1$（根索引 0 除外）。

与教师前向传播相比，这些检查开销很小，并显著提升了异构运行时上的可调试性和可复现性。

复杂度。移位索引增加了一行哑行，并以常数时间预处理取代哨兵处理。构建祖先表需要 $\mathcal{O}(MD_{\max})$ 的时间和内存。在我们的设置中， $D_{\max}$ 很小（树深度受推测预算限制），因此与教师侧计算相比，这部分开销可以忽略不计。

3.3 在盘古+Ascend 上进行融合树掩码教师模型执行

给定一棵推测树 $\mathcal{T}$和已提交前缀缓存 $\mathcal{C}^{\star}$，教师模型必须在正确的祖先上下文中评估所有推测节点，且不得发生跨分支泄漏。在我们的目标技术栈上，此评估还必须与融合注意力内核及后端特定的 KV 缓存布局保持一致。我们将教师验证实现为对推测词元进行的一次批处理前向传播，其中包括：(i) 从已提交前缀读取缓存；(ii) 为推测词元进行分支局部的缓存写入；以及 (iii) 使用树注意力掩码来强制执行祖先谓词。

执行视图：前缀缓存 + 推测词元。设 t 为当前已接受前缀的长度。从概念上讲，每个推测节点 u_k 都对应其自身从根节点到该节点路径上的一个位置，深度为 $t + \mathrm{depth}[k]$。我们向教师模型输入一个由推测词元 ID $\tau[1{:}M]$组成的张量（批处理时填充至 $M_{\max}$），同时提供 $\mathcal{C}^{\star}$ 作为前缀 KV 缓存。教师模型会计算推测位置的隐藏状态，并生成每个节点的对数几率 $\ell[k]$，以便在适当的上下文中验证 $\tau[k]$。

树注意力掩码（仅祖先可见性）。在推测段内，当且仅当 j 是推测树（第 2.4 节）中 k 的祖先（包括其自身）时，词元 k 才可以关注词元 j。利用第 3.2 节中的祖先表 A，我们可以将此谓词表示为：

$$\mathrm{Anc}(j,k) = 1 \quad \Leftrightarrow \quad \exists \ell \in [0, D_{\max}] \text{ s.t. } A[\ell, k] = j.$$

我们针对推测位置构造掩码 M_{tree}，使得

$$M_{\mathrm{tree}}[k,j] = \begin{cases} 0, & \mathrm{Anc}(j,k) = 1 \text{ and } \mathrm{valid}[k] = \mathrm{valid}[j] = 1, \\ -\infty, & \text{otherwise.} \end{cases}$$

随后，将此掩码与后端所需的掩码（例如填充掩码）合并，并广播至融合内核所需的形状（通常为 $[B, 1, M_{\max}, M_{\max}]$ 或 $[B, H, M_{\max}, M_{\max}]$）。

不泄漏到填充槽。对于填充节点槽，我们将 $\mathrm{valid}[k] = 0$ 设为 false，并强制屏蔽所有指向 k 或来自 k 的注意力。这样可确保任意填充词元标识和哑索引不会影响有效对数几率。

KV缓存写入与提交。在教师验证期间，随着前向传播就地扩展缓存，推测位置的 KV 块会写入所选的 分支缓存（第 3.1 节）。接受步骤之后，我们通过以下任一方式更新已提交缓存：(i) 基于长度的提交，仅保留所选分支中的前 A 个新步骤；或 (ii) 基于路径索引的提交，重新排列所选分支缓存，使其与接受路径前缀相匹配；若已提交前缀的顺序保持不变，则采用前缀共享快速路径。

将教师输出映射为节点验证分数。 融合的教师前向传播会在每个推测槽位上生成用于下一词元预测的对数几率。对于每个节点 k，我们在掩码上下文下提取所提议词元 $\tau[k]$ 的教师分数，得到接受规则所使用的 $\ell(k)$。该提取实现为沿词表维度的范围内 gather，并由 valid 保护，因此填充槽会被忽略。

正确性保证。 上述构造强制满足两个关键性质：

> 1. **上下文正确性**。 对于每个节点 k，用于为 $\tau[k]$ 评分的隐藏状态仅依赖于已提交前缀以及 k 的唯一祖先链上的推测词元。 2. **状态安全性**。 推测性 KV 写入不能改变 $\mathcal{C}^{\star}$；提交更新是仅在接受之后执行的显式操作。

结合第3.1节中的缓存不变量，提交接受路径后得到的教师状态，与从已提交前缀开始，按顺序解码那些已接受的词元后得到的状态相同。

实现说明（稠密掩码与结构化掩码）。直接使用稠密 $M_{\text{tree}} \in \mathbb{R}^{M_{\max} \times M_{\max}}$ 最简单，也符合常见的融合内核接口。当 $M_{\max}$ 增大时，可以通过预先计算紧凑的祖先编码（例如借助 A）并在设备端生成掩码来降低开销；我们的实现会根据推测预算选择掩码构造策略，以平衡掩码开销和内核简洁性。

4 实现与评估框架

我们的首要目标是让 EAGLE-3 风格的树式推测解码能够在运行于昇腾 NPU 上的盘古教师后端中部署并诊断。因此，我们将执行路径控制和结构化追踪视为系统的一等组成部分，而非临时的调试工具。

4.1 双模式执行协议（参考模式与性能模式）

我们的研究反复发现，“树解码语义”与后端执行路径（KV 布局、注意力内核和掩码处理）密不可分。因此，在整个实现和评估过程中，我们采用明确的双模式协议：

- **参考模式（关闭融合注意力）**。用于隔离语义错误，并在便于调试的执行路径下运行不变量检查。在此模式下，我们通过后端环境标志关闭融合注意力（例如 PANGU_DISABLE_NPU_FUSED=1 和 PANGU_DISABLE_NPU_FUSED_TREE=1）。
- **性能模式（开启融合注意力）**。用于吞吐量基准测试和预算扫描。在此模式下会启用融合注意力（例如 PANGU_DISABLE_NPU_FUSED=0 和 PANGU_DISABLE_NPU_FUSED_TREE=0，或者根据运行时默认设置取消设置）。

我们还提供一种**与即时执行对齐的分析变体**，它会强制使用即时注意力（例如 PANGU_FORCE_EAGER_ATTN=1），并在可能的情况下，让基线与 EA 遵循相同的高层生成路径，以支持对不同执行路径进行受控比较。

4.2 供应商覆盖与精简的模块边界

为保持系统的可移植性并保留生产代码路径，我们遵循“供应商覆盖”原则：追踪、性能分析和受控消融实验所需的改动，都在相关模块的供应商副本中实现，并通过将供应商目录置于 sys.path 的前面来导入。这样既能保持第3节中的核心契约稳定，也能在不修改上游运行时的情况下，按实验进行插桩。

4.3 结构化跟踪记录与调试产物

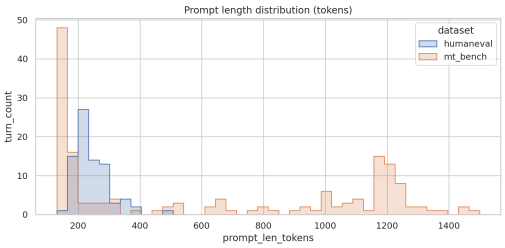
每次运行都会生成一份清单 （包含超参数、环境标志和版本标识符）以及结构化的逐提示/逐轮跟踪记录。这些记录会捕获面向执行和分析的关键信号（例如解码配置、推测树统计信息、接受率摘要和分阶段计时），从而支持可复现基准测试和事后诊断，而无需依赖临时日志。当运行异常终止时（例如发生运行时错误或不变量检查失败），我们会生成一份精简的故障转储，记录复现问题所需的最少上下文（提示标识符、输入以及相关的树/缓存元数据）。

4.4 分布式评估与确定性分片

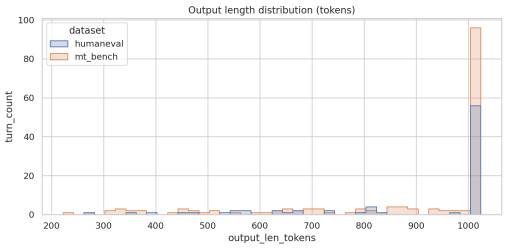
我们使用多进程分布式执行（通过 torchrun 在 8 个 NPU 上运行）进行实验。为确保确定性覆盖，提示词按照稳定规则进行分片（例如，prompt_id mod world_size）。每个 rank 分别写入跟踪文件，然后由 rank 0 将其合并为全局排序输出，以生成汇总 CSV/JSON表格。

4.5 昇腾 NPU 上的计时方法

吞吐量测量报告一次完整生成调用的端到端挂钟时间，并通过设备同步避免异步计时伪影。我们报告：(i) 每秒生成词元数（词元/秒），



提示长度分布。



输出长度分布。

图1：评估集中的输入/输出长度分布（来自结构化跟踪记录）。

(ii) 每个输出词元的时间（每输出词元时间），以及 (iii) 运行时可测量时的首词元时间（首词元时间）。性能分析实验（阶段分解和注意力统计采样）单独进行，因为插桩会影响端到端计时；我们仅将其用于解释，不将其作为主要吞吐量结果。

5 实验

我们报告的结果涉及：(i) 在融合性能路径上的**端到端吞吐量**，(ii) **预算敏感性**和实际的参数选择，以及 (iii) **解释性分析**（开销分解、负面结果和注意力证据）。

5.1 设置

数据集与样本量。 我们使用 160 条提示词进行评估（80 条 HumanEval 风格提示词和 80 条 MT-Bench 提示词），共计 240 轮（MT-Bench 有 2 轮）。提示长度和输出长度统计数据根据结构化跟踪记录计算得出（平均提示长度为 ≈ 501 词元；平均输出长度为≈ 891词元；每次实验的输出上限为 max_new_词元）。

教师后端与硬件。 我们通过运行于昇腾 NPU 上的盘古后端调用教师模型，遵循《盘古嵌入式》 [4]中描述的部署环境。

解码配置。 除非另有说明，我们使用贪心解码（温度=0）和 max_new_词元=1024。预算扫描使用 max_new_词元=256以加快扫描速度。我们比较：**基线**（仅教师模型的贪心解码）与**EA**（使用同一教师模型进行树式推测解码）。

指标。 我们报告词元/秒和加速比（Tok/s_{EA}/Tok/s_{base}）。为解释加速比，我们报告每个验证步骤接受的草稿模型长度 L_k，并将其汇总为 accept_L（对所有步骤和轮次展平统计），同时报告逐位置接受曲线（accept_pos）。

5.2 主要结果与分析

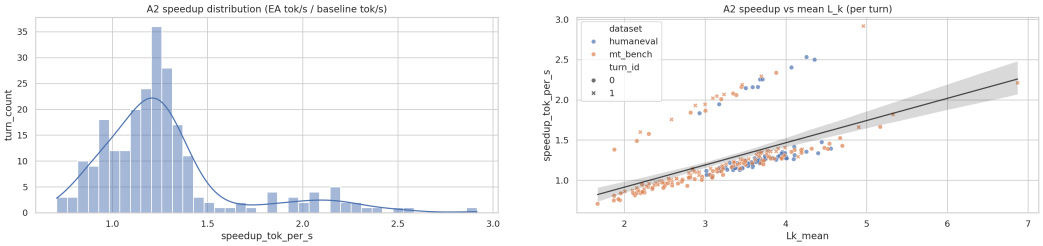
总结。 在 240 轮中，我们发现树形推测能持续提升端到端吞吐量，而预算扫描中表现最佳的配置达到了 $1.48\times$ 平均加速比；草稿生成器上下文的简单截断会大幅降低接受率，甚至可能抵消加速效果。

E1：端到端吞吐量（批次大小为 1，启用融合）。目标。 衡量融合内核性能路径中的端到端解码吞吐量提升。**设置。** 我们在 240 轮中运行批次大小为 1 的生成，并报告每轮的词元/秒以及相较于仅教师模型贪心解码的加速比。

结果。 EAGLE-PANGU 将平均吞吐量从 17.65 词元/秒提升至 22.42 词元/秒，对应 $1.27\times$ 平均加速比；尾部的提升更大（第 90 百分位： $1.84\times$，p99： $2.46\times$；表1）。每个验证步骤的接受长度平均为 3.17（中位数为 3；表1），且与加速比呈正相关（图2）。逐位置接受率会随草稿位置增加而下降（图3），这解释了更深草稿位置的收益递减。

表1：吞吐量微基准测试（240轮次，已启用融合注意力）。汇总各轮次的词元/秒和加速比；accept_L 汇总所有 EA 验证步骤中的扁平化 L_k 个样本。

指标	mean	p50	p90	p99
基线词元/秒	17.65	17.72	18.69	19.37
EA 词元/秒	22.42	21.72	31.65	42.09
加速比（×）	1.27	1.21	1.84	2.46
accept_L（L_k）	3.17	3	6	8



(a) 加速比分布。 (b) 加速比与平均 L_k。

图2：吞吐量微基准测试诊断（启用融合）。加速比与接受的草稿长度相关。

要点。 即使采用保守的缓存复制实现，树形推测在批次大小为 1 时也能显著提升吞吐量；接受长度是每轮加速比方差的主要影响因素。

E2：预算敏感性与实用最佳点（仅 HumanEval 扫描）。由于掩码/张量开销以及接受收益递减，树预算并非“越大越好”。我们进行扫描：(i) 固定深度上限，调整节点预算 M；(ii) 固定 M，调整深度上限；为提高效率，使用 max_new_词元=256。**目标**。在现实开销（掩码处理、张量化和提交）下，确定实用的树预算设置。**设置**。在 HumanEval 子集上，设置 max_new_词元=256，扫描节点预算 M（固定 $D_{\max} = 10$）和深度上限 $D_{\max}$（固定 $M = 64$）。**结果**。吞吐量随 M 和 $D_{\max}$ 变化并非单调（表2、图4）。本次扫描中的最佳平均加速比为 $1.48\times$，对应 $M = 16$，$D_{\max} = 10$。超过此范围继续增大 M 或 $D_{\max}$ 可能会降低加速比，这与以下情况相符：(i) 张量化/掩码处理/提交开销更高；(ii) 在更深的草稿位置上，接受概率更低。

要点。 树形推测存在一个取决于配置的最佳区间；要在部署中保持稳定性能，必须进行轻量级预算扫描（或采用自适应策略）。

E3：时间都花在哪里了？阶段分解（仪器化性能分析）。**目标**。 分析各项开销，以确定核心教师前向传播之外的优化目标。**设置**。 我们运行了仪器化性能分析（使用相同的提示集，最多_个新_词元=1024）。由于插桩会扰动运行时，因此我们使用这些测量结果进行诊断，而非作为主要吞吐量指标。

结果。 树张量化和掩码构造的平均耗时为毫秒级，而验证和提交的耗时与之相近。预填充呈现出明显的长尾（图5）。

要点。 进一步加速应优先提升提交效率，并改善长上下文预填充表现；掩码构造并非我们系统中的主要瓶颈。

E4：负面结果——固定窗口草拟器截断会造成损害（以及原因）。**目标**。 测试减少草拟器上下文是否能在不损害接受率的情况下提高吞吐量。**设置**。 我们将草稿模型上下文截断为固定窗口 $W \in \{128, 256, 512\}$，同时保留教师上下文不变，并与不进行截断的情况进行比较。

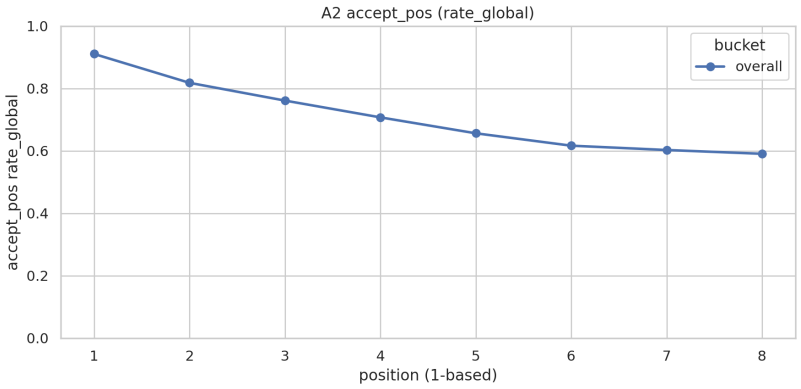


图 3：在评估集上汇总的逐位置接受（accept_pos）。较后的草稿位置更难被接受，这解释了 L_k 的长尾行为。

表 2：预算扫描摘要（仅 HumanEval，max_new_tokens=256，融合开启）。我们报告平均词元/秒和平均加速比。

扫描	设置	EA 词元/秒（平均）	加速比（平均）
扫描 M （固定 $D_{\max} = 10$）	$M = 16$	27.35	1.48
	$M = 32$	19.54	1.06
	$M = 64$	21.44	1.17
	$M = 128$	22.74	1.24
	$M = 256$	22.95	1.25
扫描 $D_{\max}$ （固定 $M = 64$）	$D_{\max} = 4$	19.58	1.06
	$D_{\max} = 8$	22.33	1.21
	$D_{\max} = 10$	21.60	1.17
	$D_{\max} = 12$	20.52	1.11
	$D_{\max} = 16$	18.78	1.02

结果。 固定窗口截断会显著降低接受率，甚至可能使加速比转为负值（表3、图6）。注意力分析提供了支持性证据：Top-1注意力位置经常落在远历史分桶中（图7），这表明硬截断会丢弃草稿模型经常依赖的位置。

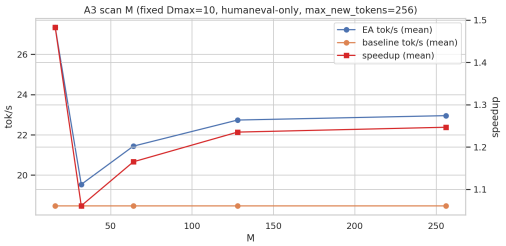
要点。 朴素起草器截断在此场景下适得其反；上下文缩减必须具备语义感知（例如自适应记忆、检索，或由注意力／重要性信号指导的截断策略）。

可纳入附录的补充分析。 除上述核心结果外，我们还对提示长度、合成聊天历史长度以及合成前缀／上下文长度如何影响接受率、树形结构和分阶段计时进行了结构化分析。这些结果非常适合放入附录，以强化“为何如此”的论述，同时不增加正文篇幅。

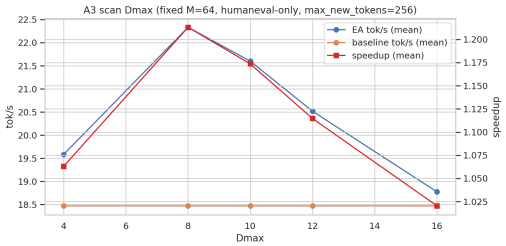
6 结论

我们提出了 EAGLE-PANGU，这是在盘古+Ascend 技术栈上对树式推测解码所做的加速器安全移植。我们的系统结合了：（i）显式的分支／提交式 KV 缓存管理器；（ii）避免未定义索引的设备定义的树张量化；以及（iii）一条兼容融合内核的树掩码教师验证路径，并配备即时执行回退。

在批量大小为 1 的基准测试中，EAGLE-PANGU 实现了显著的吞吐量增益（平均加速比最高达 1.27×，p99 加速比最高达 2.46× ）。

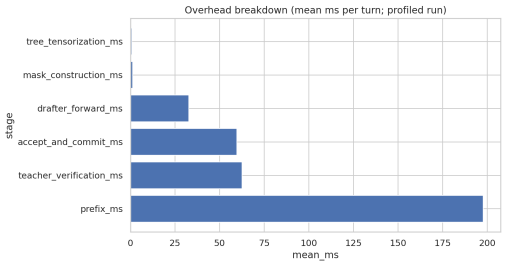


(a) 扫描 M（固定 $D_{\max} = 10$）。

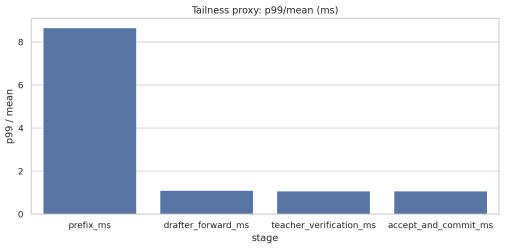


(b) 扫描 $D_{\max}$（固定 $M = 64$）。

图4：预算扫描显示，吞吐量表现出非单调变化，并存在依赖配置的最佳点。



(a) 平均阶段时间（毫秒）。



(b) 尾部比率（p99/平均值）。

图5：开销分解（已插桩；仅用于分析）。树张量化和掩码构造耗时均为毫秒级，而验证和提交的耗时与之相当。预填充呈现出明显的长尾。

我们的实验设计结合了轻量级分析研究（掩码泄漏检查、接受率/开销分解，以及对推测预算和深度的敏感性分析）与验证基准测试，测量单请求和类服务批处理条件下的端到端吞吐量和延迟。这些结果共同刻画了草稿器容量、推测预算、批处理/填充效率和验证开销之间的实际权衡，并表明树形推测能够减少教师模型调用、提高解码效率，而无需进行侵入式内核重写。

局限性与未来工作。首先，加速比受起草器成本和验证开销（掩码构造、缓存提交）的限制；随着推测树增大，这些开销会更加显著。更紧凑的掩码表示和更深入的内核融合是值得探索的方向。其次，性能取决于草稿模型的质量：更强的草稿模型可提高接受率，但如果起草过程变得计算受限，也可能削弱收益。推测感知蒸馏和自适应分支策略有望改善这种权衡。最后，我们目前的评估侧重于解码效率；将研究扩展到更长上下文的工作负载、结合工具使用的多轮聊天以及多设备服务，将有助于进一步明确部署范围。

总的来说，本研究为在昇腾级加速器上部署树式推测解码提供了一条实用路径，并明确了正确性不变量，同时构建了可根据不同模型规模和服务约束进行调优的模块化流水线。

表3：仅草稿器固定窗口截断（最大 _ 个新词元_=1024，已启用融合）。截断会降低接受率，并可能抵消吞吐量增益。

窗口 W	EA 词元/秒（平均值）	加速比（平均值）	accept_L 平均值	accept_L 第 90 百分位
none	19.93	1.15	3.17	6
128	11.97	0.69	1.48	3
256	13.22	0.76	1.68	4
512	15.14	0.87	1.99	4

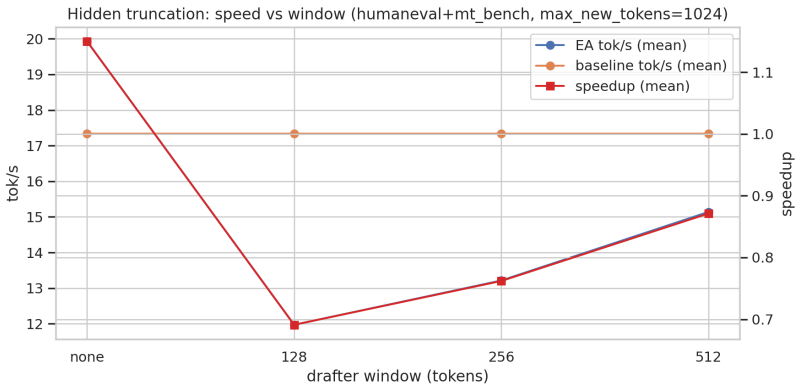


Figure 6：仅草稿器固定窗口截断下的吞吐量加速比。更小的窗口会降低并降低端到端速度。

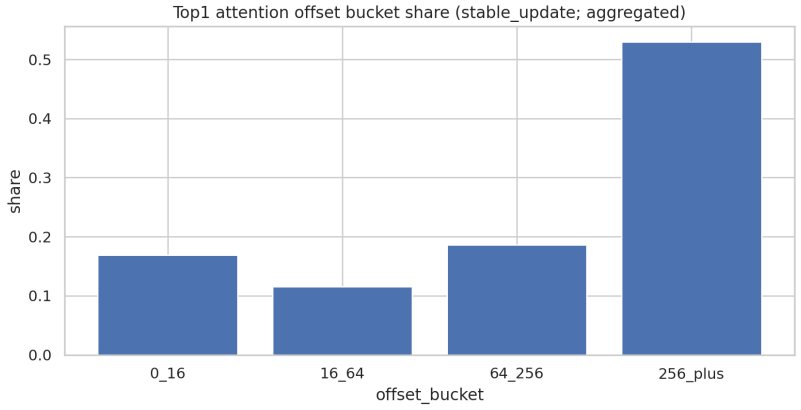


图7：草稿注意力证据（插桩；仅用于分析）：Top-1注意力位置经常位于远距离历史（256_plus桶），这与截断损害草稿质量和接受率相一致。

参考文献

[1] Yaniv Leviathan、Matan Kalman 和 Yossi Matias。通过推测解码实现Transformer模型的快速推理。载于 Andreas Krause、Emma Brunskill、Kyunghyun Cho、Barbara Engelhardt、Sivan Sabato 和 Jonathan Scarlett 编，第*40*届国际机器学习会议论文集，《机器学习研究论文集》第202卷，第19274–19286页。PMLR，2023年7月23–29日。网址 https://proceedings.mlr.press/v202/leviathan23a.html。

[2] Yuhui Li、Fangyun Wei、Chao Zhang 和 Hongyang Zhang。EAGLE-3：通过训练时测试扩展实现大语言模型的推理加速。载于神经信息处理系统进展（*NeurIPS*），2025。网址 https://openreview.net/forum?id=4exx1hUffq。NeurIPS 2025海报。于OpenReview发表：2025年9月18日。

[3] Tianle Cai、Yuhong Li、Zhengyang Geng、Hongwu Peng、Jason D. Lee、Deming Chen 与 Tri Dao。Medusa：具有多个解码头的简单大语言模型推理加速框架。载于 Ruslan Salakhutdinov、Zico Kolter、Katherine Heller、Adrian Weller、Nuria Oliver、Jonathan Scarlett 与 Felix Berkenkamp 编，第*41*届国际机器学习会议论文集，《机器学习研究论文集》第235卷，第5209–5235页。PMLR，2024年7月21–27日。URL https://proceedings. mlr.press/v235/cai24b.html。

[4] H. Chen, Y. Wang, K. Han 等. 盘古嵌入式：一种高效的双系统大语言模型推理器，具备元认知。*arXiv* 预印本 *arXiv:2505.22375*，2025。URL https://arxiv.org/abs/2505.22375。