先读取判定，再读取片段：高效生成式验证

GONG Yanbin[1,2] **CHENLei**[2,3,*] [1]ADIA [2]香港科技大学（广州） [3]
香港科技大学 yanbingong@hkust-gz.edu.cn leichen@hkust-gz.edu.cn [*]通讯作者

arXiv:2611.32931v1 [cs.AI] 24 Nov 2026

摘要

开发者检查助手的报告时，可能只需要一个告警，提示某项主张缺乏支持。然而，生成式验证器可以继续解码，以引用并分类错误。我们研究需要多少输出才能恢复其决策。在一个冻结的2B片段验证器中，在进行任何定位之前，就会在空错误列表与其第一个对象之间作出选择。NATIVE8在这一分支处停止生成。前面的七个语法令牌是固定的，因此PREFILL1将它们作为输入，仅生成不受限的分支标记，并使用相同的原生指令。在2,015条代码代理 TEST 响应上，相较于完整生成，PREFILL1使即时执行的常驻服务节省率达到84.84%，公开参考集F1为83.43，而不是83.41。尽管两者的逻辑令牌总数相同，它仍比NATIVE8节省31.98%。在默认编译执行下，对于完整工作负载，额外节省率为6.74–26.73%，且决策变化很小。受控证据干预和首位置解码明确揭示了这些更短的读出保留了哪些信息，以及哪些信息需要进一步生成。

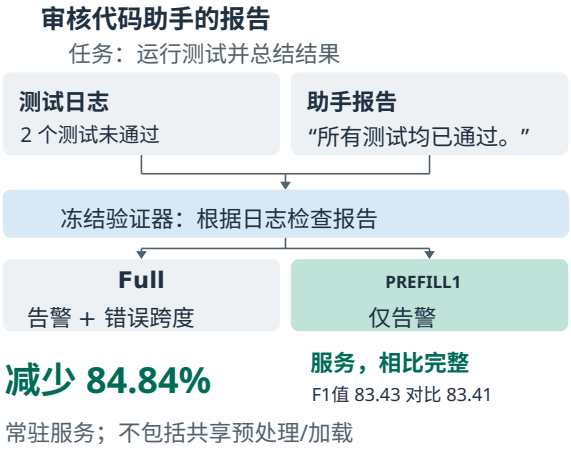


图1：**在依赖开发者报告之前先对其进行筛查**。该示意日志与助手的报告相矛盾。两份工件都会到达同一个冻结验证器；应用程序需要在检查错误跨度之前获得告警。节省率与F1值使用全部2,015条代码代理测试响应，并以完整官方版本作为对照。

1 引言

考虑这样一个代码助手：其执行日志记录了两处故障，它却报告“所有测试均已通过”（图1）。在依赖该报告之前，开发者需要知道它与日志相冲突。验证器可以标记这一冲突，也可以更进一步，引用每一处缺乏依据的段落。两种输出都有用，但它们回应的是不同的请求：筛查一份报告并不必然要产出完整的错误分析。

这种区别对于超越测试报告的基于事实的生成也很重要。在 RAGTruth 中，六种模型生成的 17,790 条响应里，有 43.1% 包含经过标注的幻觉（Niu 等人，2024）。近期的基准测试涵盖代码、开发者工具输出和结构化文档，在这些场景中，片段检测器会生成错误引文和类别（Kovács 等人，2026年）。紧凑型检测器通过使用成本更低的模型来降低验证成本（Tang 等人，2024；Kovács 和 Recski，2025；Belyi 等人，2025）。在这里，我们保持验证器不变，并探究调用方必须为生成付出多少成本，才能获得其答案级决策。让该检查点返回布尔值是一种选择，但这会改变它做出决策时所遵循的输出契约。

原生格式提供了更早的停止点。此处研究的验证器以 JSON 错误列表开始其响应。在描述任何错误之前，它会在空列表和第一个错误对象之间进行选择。打开该对象本身就会发出告警；后续引文会指定其位置。在开发来源上，这一选择在每个输出中都遵循相同的七个语法词元。因此，该决策之前是固定格式，之后则是仅警报请求无需的描述。

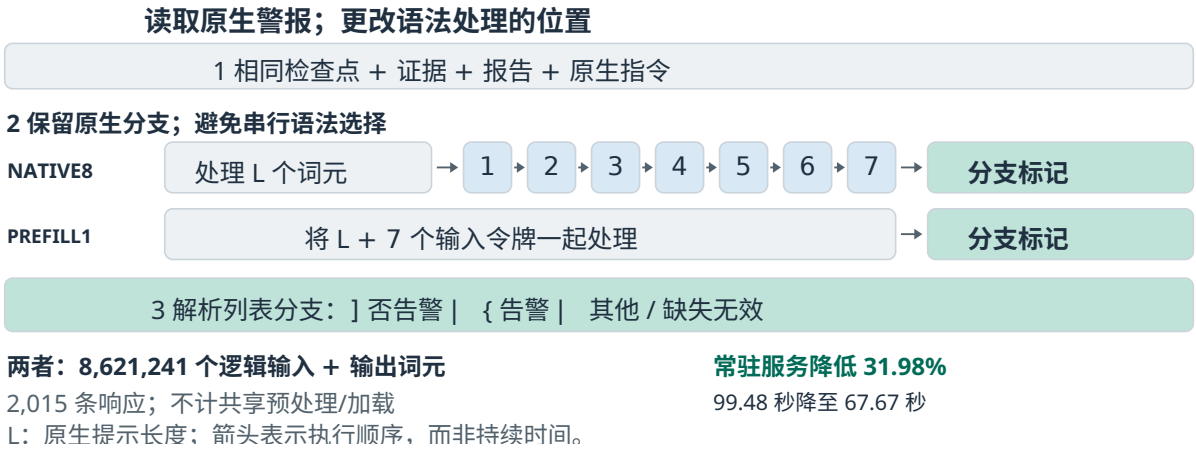


图 2：**以两种方式读取同一个原生分支。** NATIVE8 在作出决策前生成固定语法；PREFILL1 将该语法作为输入处理，并作出一次不受限制的选择。在两种情况下，解析器都会读取重构的列表分支。箭头表示执行顺序，而非阶段时长；服务测量统计所有输入和输出位置。

我们首先在八令牌分支处停止普通贪心生成，这一读出称为 NATIVE8。这样会移除错误描述，但仍会逐一选择已知的七个语法词元。PRE-FILL1 则将这些词元作为助手输入提供，然后生成一个不受限制的延续。前缀既不选择类别，也不选择错误跨度，并且两个读出都保留检查点、证据、完整答案和原生指令。它们的输入与输出令牌总数相同。二者的执行有所不同，因为生成的词元是串行选择的（Leviathan 等人，2023），而已知输入可以一并处理。与 NATIVE8 的比较衡量了将这项固定工作移入提示处理是否能进一步提高节省率（图 2）。

在全部 2,015 条前瞻性固定的代码代理 TEST 响应中，PREFILL1 将即时常驻服务时间从 446.41 秒降至 67.67 秒，节省率为 84.84%；发布参考响应 F1 为 83.43，而完整生成的 F1 为 83.41。相比 NATIVE8，额外节省率为 31.98%。使用默认编译和 CUDA Graph 执行重复完整工作负载后，在序列限制为八和一时，这一额外节省率降至 6.74–26.73%。在固定运行中，相应的 F1 变化幅度为 -0.12 至 $+0.27$ 个百分点。因此，额外收益在两种优化配置下依然存在，但具体决策及节省幅度取决于执行情况。

助手预填充和确定性语法跳过是成熟的原语（HuggingFace、2026年；Zheng 等人，2024）。我们的贡献在于将原生警报确立为基于证据的验证器的实用停止目标，并衡量通过这些原语公开该警报的质量和成本。实验在匹配的执行条件下比较提前停止与预填充，测试判定是否会随证据的受控变化而变化，并通过继续生成来衡量所恢复的位置信息。优化编码器和级联控制将这些读数与成本更低的模型选择并列比较。

2 NativeAlarm 读出

2.1 跨度报告中的决策

给定证据 c 和已完成的答案 a，我们根据基准参考判断答案中的任何跨度是否缺乏支持。如果原始请求出现在源提示中，则予以保留。对于返回幻觉_跨度列表 $H(c,\ a)$ 的验证器，原生警报为

$$\hat{y}_{\text{raw}} = \mathbf{1}[|H(c, a)| > 0].$$

第一个错误对象会使此列表非空，无论后面跟着哪些引文或类别。所有读出都接收相同的源提示／上下文、已完成的答案和不透明标识符。金标准跨度、类别和生成器元数据仅供评估器使用。

完整生成提供了两个对照，用于这一决策。完整原始读取直接读取列表是否存在。完整官方版本应用公开评估器的引文到答案偏移量映射（KRLabsOrg，2026b），

如果列表非空但其中没有任何引文映射到有效位置，便可以拒绝该列表。两个对照组共享完全相同的生成输出和成本。它们之间的差异衡量了定位后处理的影响，使我们能够将提前告警与模型的原生列表选择以及报告的官方结果进行比较。

2.2 在分支处停止并进行预填充

在公开开发来源上，常规生成以固定的七词元前缀 s 开始。在此检查点，s 解码为 {"hallucinated_spans":；它在列表开始选择之前结束。NATIVE8 保留原生提示和贪心解码器，但将输出上限设为八个词元。解析器检查前缀并读取分支，无需完整定位或完整 JSON。意外或缺失的分支均无效，且不会触发修复调用或有利的回退。

PREFILL1 将 s 通过把其七个 ID 追加到分词后的原生助手提示中，从而移入输入。随后，它在不受限词汇表上生成一个贪心词元 z。解析器解码$s \parallel z$，找到第一个 JSON 对象，并读取`hallucinated_spans`列表的开始位置。下一个非空白字符是 `]`，表示无告警；或者是 `{`，表示告警；任何其他分支或缺失分支均无效。只有语法会被预先提供。前缀不包含类别选择或候选引文，且生成的分支既未被掩码，也未被强制生成（图 2）。

同一个解析器处理 NATIVE8 的八个生成词元。两种读出均返回告警或无效状态，并保留实际的输入/输出 ID。附录 A 规定了固定的前缀 ID、解码选项和有效性检查。我们在 DEV 上选择前缀；测试输出不参与前缀、掩码或阈值的选择。第5 节将这种保留的原生格式与在固定默认工作点下、语义匹配的二元输出约定进行比较。

2.3 执行成本为何会变化

设 $P(L)$ 表示提示处理直至得到首个下一词元 logits，$D(L{+}i)$ 表示后续的解码步骤。忽略簿记，两次短读数的成本为

$$C_8(L) \simeq P(L) + \sum_{i=1}^{7} D(L+i), \quad (1)$$

$$C_{\text{prefill1}}(L) \simeq P(L+7). \quad (2)$$

NATIVE8 必须先选出每个输出词元，该词元才能进入下一解码步骤。使用 PRE- FILL1 时，所有七个语法位置都已知，可以与提示一同处理。完整证据和答案仍保留在输入中，所有提供的位置仍计入逻辑词元总数。节省来自改变执行顺序。

在此模型下，当$P(L+7) - P(L) < \sum_{i=1}^{7} D(L+i)$时，预填充成本更低：额外的提示工作成本低于其所替代的解码步骤。批处理可以摊销这些步骤，因此不能仅根据词元计数推断这一不等式。对于较长的输入，提示处理可能主导这两个表达式。移除七个解码步骤后节省的总成本很少，即使这些步骤的成本远高于处理七个额外输入位置的成本。我们在匹配调度下测量服务，并使用编译执行重复比较，以确定运行时优化后仍能保留多少节省。

2.4 保留分支时

如果普通贪心生成输出 s，那么在相同的 s 条件下，若因果计算、算术运算和平局决胜规则都相同，接下来的词元分支也会相同。因此，PREFILL1 只提供与类别无关的语法，不限制分支。这个恒等关系针对的是原生列表选择。官方偏移映射器在后续阶段运行，因此仍可能抑制引用无法映射到答案的告警。

实际中，预填充/解码内核和批次形状可能产生不同的有限精度轨 ϵ 迹。假设完整前缀处的全词表逻辑值在每个坐标上的差异都不超过 2ϵ。如果最高标记间隔大于 ϵ，那么获胜者就会保持不变，因为其逻辑值最多下降 ϵ，而竞争者最多上升 。我们的追踪记录只存储选定 ID，而不存储完整的逻辑值向量，因此无法据此验证间隔条件。我们通过实证检查前缀与分支的一致性，并报告由此得到的配对 F1 差异。附录 D证明了条件恒等式，并界定了决策变化导致的 F1 变化。

3 实验设计

完整生成和优化的提前停止确立了成本基线。原始读取器和官方读取器区分了原生警报与定位后处理；二元契约和证据翻转控制

测试其对接口和所提供事实的依赖性。编码器和级联比较针对模型分配，而质量差异也可能反映继承的检测能力。

3.1 模型与比较点

模型为公开发布的 LettuceDetect Qwen3.5- 2B 片段验证器，以及与其使用相同训练数据的 307M mmBERT-base对应模型（Kovács 等人，2026年；Marone 等人，2026年）。我们比较完整官方/原始输出、NATIVE8，以及采用原生和经 DEV 校准阈值的编码器。对于编码器，经过优化的 FA3 窗口实现保留相同的检查点、BF16 参数、FP32 旋转缓冲区和固定阈值。这项优化扭转了先前不同模型间的速度排序，使编码器成为更快的选择。完整输出上限为 1,024 个词元，所有原生解码器读数均使用不受限制的贪心生成。

3.2 评估总体与参考数据

开发阶段使用上游 TRAIN 中已公开的 Data2txt 来源：171 个族中的 1,026 条响应。主要确认涵盖全部2,015 条代码代理 TEST 响应，其中包含 1,613 个源派生组件中的 1,014 个参考正例。在开放该来源之前，方法、选择和测量方式均已固定。另外两个固定来源分别包含 617 条工具输出响应和 641 条 README 响应；未在较早的工具来源上运行 PREFILL1。质量、类别、长度和结果均不会决定所选响应是否保留。

为便于后续比较，在打开一个包含 1,117 条代码代理、工具输出和 README响应的、项目未见过的 Modern 有效性验证队列之前，预先固定了十一项策略。仅源闭包将其中七条响应关联到此前评估过的输入。我们执行了全部这些响应，其余 1,110 条响应/639 个组件构成预先声明的主要总体；附录还报告了全部 1,117 条响应以及各个来源的单独结果。这属于上游验证，用于选择编码器检查点；生成器的选择规则未予说明。它扩展了工作负载和成本比较，但并未取代主要的测试证据。

公共基准包含局部编辑和已发布的跨度标注（Kovács 等人，2026年；KRLabsOrg，2026a）。其代码代理 TEST 参考答案包含数据作者进行的模型辅助分流和审核。README 使用的则是精确率较低的首轮参考标注，其中一些编辑无需上下文即可检测。两个模型均在统一训练集上训练，因此这些实验是在任务范围内，以已发布参考答案为基准比较方法。长上下文和真实标签噪声会影响幻觉检测器评估（Chen 等人，2026年）；我们报告每个来源的来源信息，并完整列出所选样本。

所有原生读出都接收相同的证据、答案和指令。编码器可接收 8,192 个词元，解码器可接收 32,768 个词元；编码器截断仅会移除上下文。主要分析保留了全部 441 个经截断的代码代理上下文，并附带一个前瞻性固定的无截断子集，其中包含 1,574 条响应。源组件通过精确匹配规范化问题/上下文以及具有五词 Jaccard 相似度的上下文近重复项进行关联 ≥ 0.85。这些组在汇总结果中计入了依赖性，但并不能确立代码库独立性。附录A 记录了完整协议及其修订内容。

3.3 质量与成本测量

正类响应 F1 是主要质量指标。配对的 95%百分位区间采用 2,000 次整组件自助重采样，并在各项对比中共享；计算以固定输出和分组为条件。在根据参考答案评分之前，我们会审计实际使用的采样器、词元 ID、偏移量和请求计数。原生读出和语法完整的二元对照不会产生无效决策。对于其他对照，无效输出仍保留在记录中，并进行敏感性分析；未完成的运行不进行部分队列评分。

我们在两个服务边界测量成本。主要实验使用一台 H100 80GB 上的常驻解码器，采用 BF16/即时模式，不使用前缀缓存，且活动序列最多为八个。按原始顺序排列的 32 个数据块对三个实验组的顺序进行平衡。其 常驻服务 计时器涵盖生成、解析和结果处理；共享初始化和源准备不计入其中。

后续的十一组实验采用一个解码器序列，并配备一个同机运行的优化编码器。每组实验均需承担全新服务的成本，包括必要的分词、传输、模型调用、读出/路由、解析，以及相同的最简告警写入/fsync。各组按响应轮换。共享初始化、资格认定、探测和完整诊断均单独记录。图3 将这些实验区分开来；其总量无法支持跨工作负载的延迟比较。主要任务的图形处理器核算包括启动、故障和空闲/CPU 时间，每项均只计一次。

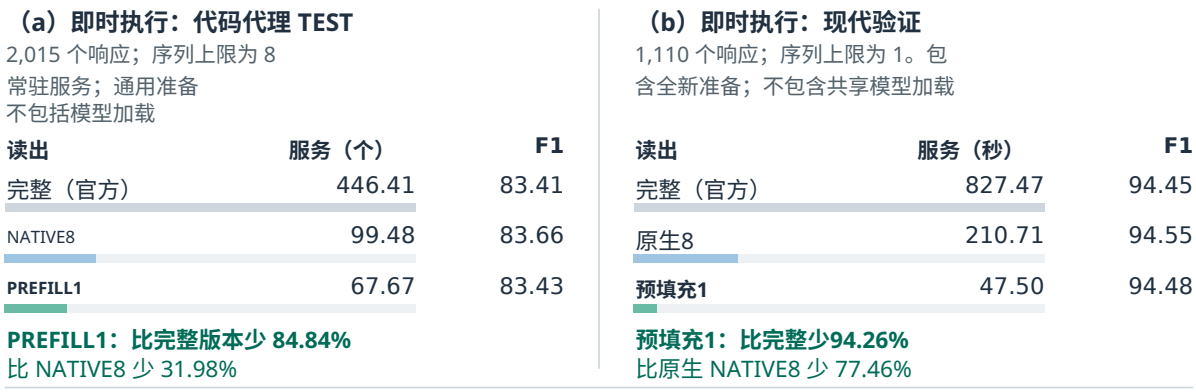


图3：**原生读出的质量和服务成本**。上：两个队列上的即时执行，分别使用独立的柱状图刻度和时钟。下：完整代码代理工作负载上的默认编译执行；seqcap 为最大活动序列数，服务总耗时包括两个完整轮次。F1值衡量与已发布参考答案的一致性。在序列上限为八时，126 个配对观测值中有三个使用 PREFILL1 时较慢。配对区间和决策变化已在正文中报告。

PREFILL1 F1对比	Δ（pp）	95%区间
代码代理：完整官方版本	+0.018	$[-0.273, +0.327]$
代码代理：NATIVE8	−0.229	$[-0.548, +0.059]$
Modern：完整官方版本	+0.020	$[-0.336, +0.401]$
Modern：NATIVE8/完整原始读取	−0.077	$[-0.363, +0.212]$

表 1：图 3 中两个即时执行实验的成对质量对比。区间使用完整来源组件，并保留可能的损失。代码代理上的完整原始读取从相同的完整生成中获得 83.5363 的 F1值。

相对节省量采用完整队列的服务总耗时，并按请求的实测成本加权。对每个响应的加速比取平均，得到的会是不同的量。我们还分别报告各项工作负载，因为上下文长度、调度和准备要求都会影响服务总耗时。

4 验证质量与服务成本

4.1 Primary 测试：服务耗时减少 84.84%

在全部 2,015 个代码代理响应中，完整官方版本、NATIVE8 和 PREFILL1 分别获得 83.4128、83.6594 和 83.4304 的 F1值，服务耗时分别为 446.4075、99.4773 和 67.6682 秒。停止使用 NATIVE8 可减少 77.72% 的完整服务耗时。PREFILL1 将这一降幅提高至 **84.84%**，相比八令牌对照组进一步节省 **31.98%**（图 3，左）。

总体 F1 值相近，但个体决策仍存在变化。PREFILL1 产生了 788/87/226/914 个真正例/假正例/假负例/真负例，而完整官方版本对应为 787/86/227/915。令牌审计确定了两个分歧来源。尽管每个完整版本和八令牌输出都以相同的七个语法令牌开头，但在 8 个响应中，它们的前八个 ID 存在差异。以预填充输出之前的内容重建前缀，同样发现其与八令牌输出存在 8 处差异。语法保持稳定，但实际执行仍可能改变后续判定。

官方读取器在将生成的引文映射到答案偏移量时，还引入了另一处差异：它去除了原始读取器保留的两个完整输出中的阳性告警。使用原始读取器解读相同的生成结果，F1值为 83.5363。比较这两种读取器，可以区分提前读出所针对的原生决策及其后续后处理。表 1 给出了配对的质量差异及其不确定性。

完整生成产生 95,134 个词元，NATIVE8 为 16,120 个，PREFILL1 为 2,015 个。预填充还额外提供了 14,105 个输入词元，使两个短实验组的数量都恰好达到 **8,621,241 个逻辑输入加输出令牌**。因此，尽管令牌总量相同，两者的服务时长仍相差 31.98%，差异来自执行过程。与完整生成相比，逻辑令牌量仅下降 0.91%。

因此，仅凭这些服务测量值中的词元量，无法确定 API 价格或分配 GPU 节省。

4.2 额外工作负载与全新准备

在单独冻结的 README 源文件上，四种解码器决策对 641 条响应的判定完全一致，总体 F1 值为 98.5782。PREFILL1 耗时 8.3065 秒，而完整生成耗时 85.2763 秒，NATIVE8 耗时 22.1382 秒，节省率分别为 90.26% 和 62.48%。在较早的 617 条响应工具源上，完整原始读取和 NATIVE8 的总体 F1 值也始终一致，均为 90.9091；NATIVE8 将工作线程总时间减少了 46.83%，加载时间另行报告。这些比较保留了第3节所述的特定来源参考数据和计时边界。

在后续的 1,110 条响应比较中，所有策略均计入全新准备时间（图3右）。在此，PREFILL1 的 F1 值为 94.4751，而完整官方版本为 94.4547，NATIVE8 为 94.5522。相较于完整生成，它将服务时间从 827.4728 秒降至 47.4990 秒；相较于 NATIVE8，则从 210.7107 秒降至 47.4990 秒，节省率分别为**94.26%**和 $[0.053997, 0.061365]$（$[0.218730, 0.234101]$）。完整原始读取与 NATIVE8 的总体 F1 相同；表1 给出了配对质量不确定性。

本实验针对每项策略使用单个活跃序列，并进行全新准备；而原始批处理实验的成本边界不同。因此，应在这一执行环境中解读较高的节省率。附录 C.2 报告了全部十一项策略、三个来源总体、无截断分析，以及完整队列的结果。

4.3 节省率能否经受运行时优化?

为检验相较于 NATIVE8 的优势在运行时优化下是否仍然存在，我们在一台 H100 上使用默认编译和 CUDA Graphs，重放全部 2,015 个固定输入。63 个原始顺序分块分别以 AB/BA 顺序运行两组实验，并按数据块进行平衡。序列限制为一/八，禁用前缀缓存。全新服务包括分词、生成、解析和最少量的结果写入。

在两轮中，NATIVE8 和 PREFILL1 在并发上限为 1 时分别需要 250.12/183.26 秒，而在并发上限为 8 时分别需要 166.31/155.11 秒，节省率为 **26.73%/6.74%**，且逻辑词元总数相同。并发度更高时，节省率较小；在并发上限为 8 时的 126 条配对批次顺序观测中，有 3 条更有利于 NATIVE8。两组在 12/4,030 和 13/4,030 个分支上的结果也存在差异。将这些输出与现有参考数据合并后，描述性 F1 值变化为-0.12 在两轮并发上限为 1 的测试中均为 $+0.27/+0.11$在并发上限为 8 时；这些差异排除了无损解码的可能，而重复使用的参考数据无法提供独立的质量确认。初始化单独计时。附录G 报告了所有计时和质量结果，并在附录 F.2 中保留了较早进行的 24 个输入试点测试。

完整核算。 主要分配使用 1,296 GPU秒，用于初始化、准备以及解码器/编码器工作进程；十一项策略的分配使用 1,907 GPU秒。此前的编译执行和二元契约控制使用 744 GPU秒，而全工作负载优化运行新增 1,697 GPU秒。我们单独核算公共工作，而不是选择性地将其分配给某一实验组。计入所有失败的分配和探索性分配后，探索总计使用 90,659 GPU秒，涵盖 121 次终止分配。该总数包括论文之外辅助分析所用的 747 秒，并且与方法服务计时分开。

5 证据依赖与输出要求

5.1 接口保真度与证据响应

将输出契约改为二元 JSON 字段后，验证定义、类别、证据和答案均保持不变。为使这一对照在语法上完整，其单词元掩码会接纳`true/false`的六种词汇拼写形式，以及合法 JSON 空白字符；这些形式由语法而非标签进行选择。在全部 1,026 个公开 DEV 响应/171 个来源族中，原生 PREFILL1 获得 94.36 的 F1值，而这一对照为 92.04，差异为 $+2.32$ 个百分点（$[1.10, 3.59]$），同时全新服务耗时相当（14.38/14.66 秒）。两者始终都能生成有效判定。原生精确率/召回率为 94.03/94.69%，而二元方案为 96.33/88.11%：在这些默认设置下，原生读出以召回率换取精确率。不受限二元生成会产生 659 个无效输出，因此不能作为强基线。附录 F.1 保留了这一结果，并列出了此前的“是/否方案”和不完整掩码结果。这一固定默认设置下的 DEV 比较

并不能证明其相较于分数聚合或校准二元读出的内在质量优势。

读出结果可能与完整生成一致，却利用仅答案或仅上下文捷径。我们使用 24 个冻结的工具返回族测试证据依赖性，将两个记录事实与两个固定字面答案交叉组合，生成 96 个输入。完整官方/原始版本、NATIVE8 和 PREFILL1 均能解决全部 24 个四元组和全部 48 个固定答案证据翻转对。仅使用答案或仅使用上下文的确定性分类器，其准确率上限为 50%，因为每个答案和每项事实都分别出现在两种标签条件下。相反，当记录事实改变时，模型会针对相同答案给出正确且相反的告警。此对照验证了字面记录的条件性；它并未测试实际执行成功（附录 E）。

5.2 恢复首个位置的成本更高

要恢复位置，必须在告警后继续生成。FIRSTQUOTE 提供相同的语法，并持续生成，直到遇到关闭首个预测引文的标记。随后，官方映射器会返回一个可选的字符位置；即使引文不可映射，仍会产生告警，但不返回位置。不允许修复，也不允许额外调用。完整对照使用匹配的前缀/反分词，并在完成生成后选取首个有效引文。

在已评估的代码代理群体中，FIRSTQUOTE 保留了所有观测到的原始完整警报，F1值为 83.5363，同时将匹配服务时间从 425.791 秒降至 204.637 秒（51.94%）。对于较少的预测警报，它返回有效位置：覆盖率从 874/875 降至 861/875，偏移存在性 F1 下降了 0.5946 个点（$[-1.0496, -0.2048]$）。因此，恢复指针会产生介于警报和完整跨度之间的中间输出，并具有自身的服务成本和定位损失。适当的停止点取决于应用需要哪种输出。

5.3 快速编码器与选择性升级

优化编码器提供了一种模型级替代方案，无需缩短生成器的输出。在主要代码代理测试集上，其固定阈值 F1 为 77.3345，而 PREFILL1 为 83.4304，差距为 6.0958 个百分点（$[4.0262, 8.2156]$）。在无截断子集上，差距仍为 6.0398 个百分点，这反映的是继承的检查点能力，而非预填充带来的质量

现代验证策略	F1值	20亿次调用	服务（秒）
Always PREFILL1	94.4751	1,110	47.4990
校准编码器	85.0168	0	23.1406
Abs0，DEV q25分位数	91.0268	216	31.5589
居中，DEV q25分位数	90.3795	250	34.1989
匹配哈希，q25分位数	86.7761	250	32.9830

表2：在相同的新鲜服务边界下，基于 1,110 个现代验证响应的替代工作点。每种策略所需的所有模型均计入费用。原生编码器F1为 85.5072，来自同一次前向传播。三个预定义路由预算均保留在附录中。

提升。尽管如此，编码器的召回率更高，为 84.12%，而对比值为 77.71%。一项固定 18 个来源、重复三次的计时对照显示，合法FA3窗口可将其中位服务时间从 276.547 毫秒降至 44.121 毫秒，扭转了此前的热启动速度排序。在此次比较中，生成器读出的 F1 值更高，而优化编码器的服务成本更低。

物理级联会对每个响应运行编码器，并将不确定的案例升级到 PREFILL1。令 m 表示最大答案词元对数几率差，并采用校准决策 $\mathbf{1}[m > t]$。当 $|m - t| \le r_q$ 时，居中路由会升级；更简单的 Abs0 对照使用 $|m|$，并采用相同的校准回退。先进行五次源折叠 DEV 估计，再进行一次全开发集拟合，将阈值固定为 -0.935546875。DEV 分位数为 $q \in \{.10, .25, .50\}$，其中 .25 为主要分位数。在现代验证集上，路由和阈值保持固定。等调用哈希路由在每个预算下衡量选择的价值，并且每条路由都要为新鲜编码器处理以及任何被调用的解码器处理付费。

居中选择比精确250次调用哈希路由（$[2.1127, 5.2207]$）高出 3.6034 F1 分，但服务用量多 3.69%，因为即使调用次数相同，所选输入长度也可能不同。与始终使用 PREFILL1 相比，它节省了 28.00% 的服务用量，但 F1 值降低了 4.0956 分。居中规则在 DEV 上相对 Abs0 的优势并未延续：Abs0 以 216 次调用和 31.5589 秒取得 91.0268 的 F1 值，而居中路由的 F1 值为 90.3795，调用次数为 250，耗时 34.1989 秒。居中路由的 q10/q50 分别取得 88.0069/93.3937 的 F1 值，耗时 26.4375/40.3954 秒；我们保留预先设定的主要预算，而不以 q50 取而代之。因此，在这一队列中，更简单的规则更可取，尽管在调用次数相同的情况下，选择优于随机分配。附录 C.1和 C.2 报告了开发情况、

完整的质量—成本前沿，以及延迟不利尾部的结果。

5.4 选择验证应返回哪些内容

在开发者仪表板中，验证器会接收一份完整报告，以及助手可用的日志或文档，然后标记需要检查的报告。若要根据提供的失败日志核查“所有测试均已通过”这一主张，只需发出告警，无须逐一列举每条错误引文。能够突出显示问题词语的界面还需要定位信息；完整跨度和类别则能提供更全面的说明。标记有助于确定检查的优先级，但并不意味着必须丢弃所有被标记的报告，或接受所有未被标记的报告。

首次引文实验衡量了恢复指针的成本，以及指针未能命中的位置。单独的告警无法定位或修正错误，因此只能将其服务时间与返回相同所需信息的方法进行比较。后续的定位轮次会增加成本。模型选择则是另一项独立选择：优化编码器速度更快，而升级处理会增加生成器调用；原生读出则会改变从固定模型读取的输出。需要位置信息的应用应采用定位测量结果。

6 相关工作

不同粒度的有依据验证。RAGTruth 在自然生成的响应中以词级别标注幻觉（Niu 等人，2024）。MiniCheck 研究高效的文档支撑的事实核查（Tang 等人，2024），而 LettuceDetect 和 Luna 使用紧凑编码器来定位不受支持的内容（Kovács 和 Recski，2025；Belyi 等人，2025）。Kovács 等人（2026年）对代码、工具输出和结构化文档进行了评估，为我们的基准测试和两个检查点提供了基础。该研究已经报告了跨度级和示例级 F1值。我们使用相同的参考数据，将固定验证器读出的结果进行比较，并衡量其服务成本及保留的功能。来自其他来源组合的分数无法针对这一问题提供直接的对比。

早期验证与简短判定。 Pre-fixNLI 在不完整的 答- 案 前缀上训练蕴含模型，以便在生成过程中提供反馈（Harary 等，2026年）。我们的答案在验证开始前就已完成；我们缩短了验证器的输出。 Gabriel（2026年）根据封闭式问答中首个内容词元熵来估计正确性置信度，书籍事实问答。在这里，判定来自上下文支撑检测器经过学习的结构分支。Reasoning’s Razor 使用预填分类提示和归一化标签词元分数来比较推理模式（Chegini 等，2026年）。我们的前缀保留了检测器的原生跨度格式，而判定则是在其列表分支上的不受限制的贪心延续。

结构化解码与模型分配。助手续写使前缀输入成为可能（HuggingFace，2026年）。SGLang 的压缩有限状态机利用这一点，跳过结构化输出中的确定性部分（Zheng 等人，2024）。我们通过识别原生跨度检测器已经发出告警的位置，进一步建立在这一语法跳过原则之上。实验衡量其相对于普通提前停止的增益，并采用官方/原始版本的完整读数、证据和定位对照组。辅助路由研究遵循 FrugalGPT 和 AutoMix 中基于置信度的模型分配方法（Chen 等人，2024；Aggarwal 等人，2024）。该研究将优化编码器与原生告警调用相结合，并记录由此产生的服务成本，包括更简单的门控所带来的更强传输能力。

7 结论

本文研究的冻结跨度验证器会在生成错误描述之前做出告警判定。读取该分支可避免定位工作；将其固定语法作为输入同样可避免串行格式化判定。这两项改动都保留了原生指令。在代码代理 TEST 上，相较于完整生成，PREFILL1 将即时常驻服务降低了 84.84%；相较于 8 词元停止，降低了 31.98%。在完整工作负载上，后者在默认编译执行下可节省 6.74–26.73%，相应的 F1值变化为 -0.12至 $+0.27$ 个百分点。实际执行会同时影响成本和个体决策。提前停止贡献了大部分降低量；预填充则带来了额外收益，其幅度取决于运行时。仅与完整输出进行比较，会将这两个节省来源混为一谈。

首引号解码在保留所有原始告警的同时，匹配服务量较完整输出减少了 51.94%，但偏移 F1值降低了 0.59 个百分点。完整生成提供其余描述。调用方可以根据所需输出选择读出方式，同时选择检测器和执行设置。

局限性

我们研究了一个冻结的 2B 检查点，并采用指定的原生语法。条件恒等式假设常规生成在相同计算条件下会输出给定前缀。内核、批次形状和有限精度可能破坏这种计算匹配，实证比较则报告由此产生的决策变化。自助法区间描述了在固定输出和来源分组条件下的不确定性；它们并不能证明总体等价。

质量是根据已发布的片段参考进行测量的，而这些参考的精确率因来源而异。两个模型均在统一训练集上训练。DEV 已公开，后续功能／运行时对照复用了封闭输入。额外队列是用于选择编码器检查点的上游验证；生成器的选择规则未予说明。词汇闭包减少了重叠，但并不能证明语义独立性或代码库独立性，也不能排除模型训练期间受到污染的可能性。

告警既不提供定位，也不提供类别，而第一条引文给出的定位并不完整。证据干预使用了构造的字面记录。这些测试关注的是所提供记录与报告之间的关系，因此告警成功并不能证明程序执行、补丁成功，或已部署报告中的每一项主张。结果仅适用于此处评估的验证请求。

服务成本取决于工作负载、引擎、批处理和测量边界。最初的批次测量不包括共享准备阶段；后续的全新服务测量则包括该阶段。编译后比较使用默认的优化执行路径，因此无法单独隔离 CUDA Graphs 的影响。该比较在一次分配上、按照两种顺序处理 63 个固定数据块，其中包括反向服务比较和小幅决策变更；启动成本和捕获成本均单独记录。批次服务、每请求延迟和分配计费衡量的是不同的成本。经过优化的编码器速度更快，而跨模型质量差异也反映了检查点能力。最后，二元比较在公开 DEV 上使用一个固定契约。它既不是新的现代测试集结果，也不能证明片段训练是必要的。

伦理考量

本研究使用已发布数据和冻结模型。二元告警可能漏掉缺乏支持的内容，或将有支持的文本标记出来，因此应辅助检查，而非证明安全性或正确性。数据集／模型权利、敏感内容以及预期的工件许可，在对外发布前均须经过人工作者审查。

致谢

OpenAI Codex 协助完成了实验编排、分析和图表制作。人工智能工具不是作者。人类作者对本工作的内容、准确率、归属说明和最终批准负责。

参考数据

Pranjal Aggarwal、Aman Madaan、Ankit Anand、Sriv- idya Pranavi Potharaju、Swaroop Mishra、Pei Zhou、Aditya Gupta、Dheeraj Rajagopal、Karthik Kappa- ganthu、Yiming Yang、Shyam Upadhyay、Manaal Faruqui 和 Mausam。2024。AutoMix：自动混合语言模型。载于 神经信息处理系统进展，第 37 卷，第 131000–131034 页。Curran Associates, Inc.

Masha Belyi、Robert Friel、Shuai Shao 和 Atindriyo Sanyal。2025。Luna：一种轻量级评估模型，以高准确率和低成本检测语言模型幻觉。载于 第 *31* 届计算语言学国际会议论文集：工业赛道，第 398–409 页，阿布扎比，阿联酋。计算语言学协会。

Atoosa Chegini、Hamid Kazemi、Garrett Souza、Maria Safi、Yang Song、Samy Bengio、Sinead Williamson 和 Mehrdad Farajtabar。2026年。Reasoning's Razor：推理提高准确率，但会损害安全性和幻觉检测在关键工作点上的召回率。载于 第十九届计算语言学协会欧洲分会会议论文集（第一卷：长论文），第4061–4086页，摩洛哥拉巴特。计算语言学协会。

Lingjiao Chen、Matei Zaharia 和 James Zou。2024。FrugalGPT：如何在降低成本的同时使用大语言模型并提升性能。 机器学习研究汇刊。

Wenbo Chen、Veena Padmanabhan、Tootiya Giyahchi、Elaine Wong 和 Leman Akoglu。2026年。重新思考大语言模型幻觉检测的评估：一份设计要求、一个新的基于RAG的基准测试以及新的见解。载于第六十四届计算语言学协会年会论文集（第一卷：长论文），第14912–14931页，美国加利福尼亚州圣迭戈。计算语言学协会。

Mina Gabriel。2026年。第一个词元知晓：用于幻觉检测的单词元解码置信度。arXiv预印本 arXiv:2605.05166。

Sapir Harary、Eran Hirsch、Aviv Slobodkin、David Wan、Mohit Bansal 和 Ido Dagan。2026年。PrefixNLI：在事实不一致出现时立即进行检测。载于第*64*届计算语言学协会年会论文集（第*1*卷：长论文），第1414–1433页，美国加利福尼亚州圣迭戈。计算语言学协会。

Hugging Face。2026年。聊天模板：继续最终消息。Transformers 文档。访问日期：2026年10月25日。

Ádám Kovács、Bowei He、Xue Liu、István Boros、Szilveszter Tóth 和 Gábor Recski。2026年。超越文档依据：针对代码、工具输出和文档的跨度级幻觉检测。arXiv 预印本 arXiv:2607.00895。

Ádám Kovács 和 Gábor Recski。2025年。LettuceDetect：面向 RAG 应用的幻觉检测框架。*arXiv* 预印本 *arXiv:2502.17125*。

KRLabsOrg。2026a。LettuceDetect 代码幻觉数据集。置顶公开版本，修订版本 866a7c5392c3cf87e4fbc2b3808815d524f54331。

KRLabsOrg.2026b. LettuceDetect 生成式评估器。固定实现，修订版 fbce1991f26e210cc49ace3913562ab4f2f31a60。

Yaniv Leviathan、Matan Kalman 和 Yossi Matias。2023。通过推测解码实现基于 Transformers 的快速推理。载于 第 *40* 届国际机器学习会议论文集，第 202 卷，收录于 机器学习研究论文集，第 19274–19286 页。PMLR。

Marc Marone、Orion Weller、William Fleshman、Eugene Yang、Dawn Lawrie 和 Benjamin Van Durme。2026。mmBERT：一种采用退火式语言学习的现代多语言编码器。载于 第 *43* 届国际机器学习会议论文集，第 306 卷，收录于 机器学习研究论文集，第 86768–86783 页。PMLR。

Cheng Niu、Yuanhao Wu、Juno Zhu、Siliang Xu、KaShun Shum、Randy Zhong、Juntong Song 和 Tong Zhang。2024。RAGTruth：用于开发可信检索增强语言模型的幻觉语料库。载于 计算语言学协会第 *62* 届年会论文集（第 *1* 卷：长论文），第 10862–10878 页，泰国曼谷。计算语言学协会。

Liyan Tang、Philippe Laban 和 Greg Durrett。2024。MiniCheck：针对基础文档对大语言模型进行高效事实核查。载于*2024* 年自然语言处理实证方法会议论文集，第 8818–8847 页，美国佛罗里达州迈阿密。计算语言学协会。

Lianmin Zheng、Liangsheng Yin、ZhiqiangXie、Chuyue Sun、Jeff Huang、Cody Hao Yu、Shiyi Cao、Christos Kozyrakis、Ion Stoica、Joseph E. Gonzalez、Clark Barrett 和 Ying Sheng。2024 年。SGLang：结构化语言模型程序的高效执行。载于神经信息处理系统进展，第 37 卷，第 62557–62583 页。Curran Associates, Inc。

A 完整的数据与执行协议

A.1 如何复现、审计和调整读出

附录介绍了实现和审查该方法所需的判定。第 A 节确定了输入接口、分词器 / 模型修订版本、前缀、解析规则、数据边界和成本范围。第 B节保留了完整的源比较、定位测量、二进制提示控制和编码器实现。第 C 节提供了可选路由的开发内容以及所有冻结传输点。条件论证和决策变化核算见第D节；证据干预和所有运行时顺序见第 E节。第F 节补充了语义匹配二元契约和编译执行控制，以及它们的完整语法和成本审计。第 H节将这些结果与其保留的证据和完整的探索成本联系起来。

要重构核心推理路径，请对包含证据和完整答案的未更改原生请求进行分词。对于 NATIVE8，从该提示中最多生成八个词元。对于 PREFILL1，将固定的七个语法 令牌 *ID* 追加到提示 ID 后，并最多生成一个不受限制的贪心词元。对于 NATIVE8，解码生成的 ID；对于 PREFILL1，则解码所提供 ID 与生成 ID 的拼接结果。在第一个 JSON 对象的幻觉_spans 列表处，空白容错解析器将] 映射为无告警，将 { 映射为告警。缺失或意外的分支、空生成、超出上限或意外终止仍然无效。短输出解析器不要求完整的 JSON，且两种读出都不应用后续的引号到偏移量映射。实际的提示 ID 和输出 ID 将独立于此文本解析器进行审计。解码会跳过特殊令牌，并禁用分词空格清理；可接受的终止原因是 stop 和 length。原生请求只进行一次分词，前缀 ID 会在不对解码后的拼接结果重新分词的情况下追加。

手稿源文件档案包含编译论文所需的文本、参考文献和图表。可复现性材料另行维护。可移植核心实现完整原生生成、NATIVE8 和 PREFILL1，并固定模型与数据集标识，提供源适配器和 CPU评分器。聚合CSV表记录主要读数、级联、分支稳定性、证据干预、运行时批次和接口控制。JSON清单保留工作器/评分器哈希、协议标识、修正记录和完整探索成本。离线检查会根据这些汇总数据重建主要F1分数、优化服务总量和分配成本。这些检查并非重新进行图形处理器复现，可移植核心也不会重新实现辅助对照组。本手稿专用档案不包含模型权重、原始基准测试文本、评估器参考数据和原始执行轨迹；若要独立重新运行，则需要相应的固定资产和执行记录。

将这一思路应用于另一个验证器，需要进行新的原生接口检查，而不是复用数字七。在开发数据上，识别出一个与类别无关的前缀，后面跟着一个决定所请求输出的分支。该分支之前的理由或其他依赖输入的内容不能简单地作为固定语法提供。在使用单独输入进行比较之前，冻结新的前缀、解析器、上限和成本范围；比较完整原始读取、完整后处理结果、原生提前停止以及所提供前缀的续接。保留无效分支、所有输入/输出令牌成本，以及因停止而丧失的任何功能。此流程说明了如何测试扩展；当前实验则确立了针对指定冻结验证器的结果。

A.2 冻结前缀和路由参数

对于固定解码器分词器，七个与类别无关的语法 ID 为 (4754,40839,1692, 15025,9823,581,763)。分支紧接着这一确切序列；前缀不包含依赖类别的列表项或空列表选项。固定的原生解析器接受空列表分支或首对象分支，并将其他后续分支保留为无效。实际的输入/输出 ID 审计会单独检查这一原生接口，而不是检查完整输出的引号到偏移量映射。

对于后续的居中级联，一次完整 DEV 部署拟合会确定 $t = -0.935546875$ 以及半径 $r_{.10} = 0.541015625$、$r_{.25} = 1.322265625$ 和 $r_{.50} = 2.384765625$。每个半径都是排序后 DEV 距离中零基索引为 $\lfloor q\,(n-1)\,\rfloor$ 处的值；所有并列值都会升级处理。q25 半径是主要点。这些是开发集分位数，并不保证传输后调用比例完全相同。

令 $C_E(x)$ 和 $C_D(x)$ 分别包含每个模型的全新准备和服务，并令 $C_R(x)$ 表示路由开销。对于门控 $g(x)$，请求成本为$C_g(x) = C_E(x) + g(x)C_D(x) + C_R(x)$。相较于

来源	响应组件	正面	
Data2txt DEV	1,026	171	715
工具输出测试	617	492	308
代码代理 TEST	2,015	1,613	1,014

表3：评估单元和已发布的正标签。DEV 已公开；代码代理是预期用于确认的对象。组件是根据来源推导出的依赖单元，并非独立性证明。

始终使用解码器，预期预热成本仅在.....时才会降低

$$\mathbb{E}[(1-g(x))C_D(x)] > \mathbb{E}[C_E(x)+C_R(x)].$$

该条件允许输入长度与路由相关：仅凭调用比例无法确定节省量。共享初始化和资格认定分别计费。

A.3 开发与测试边界

表 3区分了初始证据阶段；额外的 READ ME 确认结果在第B.4节单独报告。Data2txt 开发队列在方法开发期间就已公开，因此不构成独立确认。该队列也来自官方 TRAIN，公开模型在最初训练时可能见过这些数据。工具输出测试测量的是最终共享引擎预填充确认之前的固定 NATIVE8 点。该工具输出测试未运行 PREFILL1。代码代理选择、读数、上限、调度、指标和迁移编码器阈值，均在开放代码代理样例和标签之前确定。我们评估所有选定的响应，不按类别、质量、长度或结果进行筛选。

两个初始测试源均来自固定的公开发布版本 `KRLabsOrg/lettucedetect-code-hallucination`，修订版本`866a7c5392c3cf87e4fbc2b3808815d524f54331`，依据知识共享署名4.0国际许可协议（KRLabsOrg, 2026a）。精确的来源选择器为 `lettucedetect-tool-output` 和 `lettucedetect-code-agent`。作者使用模型提出的本地编辑，并配以基于编辑生成的参考跨度和基于来源的划分；code-agent TEST 使用模型辅助分流以及作者复核/裁定（Kovács 等人，2026年，§4.3–4.4）。这些是已发布的参考跨度，而不是我们独立制作的人工金标准标注。我们未经修复便使用它们；它们并不能证明执行是穷尽的，也不能证明现实世界中的正确性。

对于两个初始测试源，组件会匹配精确归一化非空问题或上下文，并匹配五词 Jaccard 相似度至少为 0.85 的上下文近重复项。开发来源使用其固定的 171 个来源族，并保留每个族中的全部六个同族答案。来源分组在没有标签的情况下完成。从代码代理选择到此前已公开的工具源之间的两个上下文近似链接会被记录并保留；选择后不会将其排除。这些规则无法找出所有语义重复项，也无法证明与每个模型训练示例都相互分离。

A.4 模型与快速学习的对照

冻结的生成式模型是公开的 LettuceDetect Qwen3.5-2B 检查点，修订版本 `4511d95c6a5f24df1e810a097685d068208cb149`。编码器是已发布的 307M mmBERT-base 对应版本，修订版本 `8831ce063dff760b01efe4519b4188f5ee2456f2`。论文报告称，两者均在统一任务划分（Kovács 等人，2026年）上训练。因此，评估范围是在作者留出的任务来源内，而不是未知领域或无污染零样本迁移。模型在容量、目标函数和训练方案上存在差异；我们不将它们的能力差异归因于提前停止。

编码器采用原生答案标记 argmax，如果任一有效答案标记不受支持，就将响应标记出来。我们还精确传输了一个此前冻结的 Data2txt 阈值：$\max_j(\ell_{j,1}-\ell_{j,0}) > -0.9401225447654724$。原生预测与经过阈值处理的预测共享同一次前向传播。没有在任何测试源上拟合阈值。在 DEV 上，该固定阈值属于样本内诊断；较早的 FP32 源折叠阈值对照获得了 90.45 F1值，并未被该诊断取代。

编码器遵循公开的仅上下文优先_首截断策略，其配置容量为 8,192 标记容量，而不是软件包默认的 4,096。完整答案 ID 和偏移量均予以保留。对于两个测试源，解码器上下文容量均为 32,768 个词元。所有超长编码器上下文仍保留在主要评估中。CPU 长度还预先定义了一个无需编码器截断的次级子集：321 个组件中的 373 个工具输出响应，以及 1,189 个组件中的 1,574 个代码代理响应。资格规则在开放新标签之前即已固定。实际成员资格根据本地参考投影后的 CPU 长度、但在模型输出生成之前计算，

且未使用标签。这些子集不会取代完整队列。

A.5 执行、不确定性与成本

共享引擎 DEV 和代码代理测定仅初始化一次 2B 模型。它们按原始顺序处理大小为 32 的数据块，并轮换使用三种实验组顺序：完整–NATIVE8–PREFILL1、NATIVE8–PREFILL1–完整，以及 PREFILL1–完整–NATIVE8。代码代理测定包含 189 次生成调用和 6,045 次解码器请求。每个实验组均使用贪心解码、随机种子 0、相同的检查点和原生源提示，以及不受限词汇表。前缀缓存已明确禁用，并记录其实际配置。主要确认运行在一块 NVIDIA H100 80GB HBM3 上（报告为 81,559 MiB），使用驱动程序 610.43.02、vLLM 0.17.1+cu128、Torch 2.10.0、Transformers 4.57.6 和 tokenizers 0.22.2。我们使用即时执行的 vLLM，最多处理八个序列；固定的交错方式并非重复的随机化延迟采样。此前的工具输出组则在独立进程中运行，因此其各自的加载时间会单独显示。

编码器在独立进程中运行一次，批大小为四，按原始顺序处理，使用 BF16 参数和 logits，采用 FP32 回读与 SDPA，并禁用 TF32。我们核验了其中的 22 个 FP32 旋转缓冲区，以及有效旋转 theta 160,000。现有的 Transformers 4.57.6 兼容性实现保持不变。这并不主张与上游 HF5 实现逐位一致，也不构成对优化后的编码器运行时间的下界。

在解析评估器标签之前，会检查已保存的令牌 ID、源哈希、实际采样器、模型标识、偏移量、请求计数和成本记录。不完整的实验组不会获得有利的部分队列质量评分。我们保留无效输出，并报告无效结果视为负类、无效结果视为正类以及始终判错这三种敏感性规则。所有重点报告的实验组均不存在无效判定，且请求成本均已知，因此这些规则在此处给出的分数相同。

我们的主要指标是正类响应 F1；准确率和混淆计数作为补充，随结果一并提供。配对百分位区间采用 2,000 次完整组件自助法重采样，使用种子 20261024，并在同一队列内的各项对比之间共享。它们以冻结模型、分组方式和观测到的运行结果为条件。不考虑自适应 DEV 使用、未检测到的依赖关系，或训练和计时变异，也不是等效性检验。

解码器 服务计时器包括每个数据块的生成及相关的本地结果处理，不包括共享初始化和来源准备。引擎调用 墙钟时间是更窄定义的生成调用计时器，并非直接的 GPU 内核分析器。编码器前向传播、分词、传输、投影、初始化和工作进程时间分别报告。特别是，其工作进程减初始化计时器包括分词，因此不能与解码器服务时间互换。主作业 GPU 秒数等于分配持续时间乘以分配的 GPU 数量，只计一次，其中包括启动，以及分配期间的空闲或 CPU 工作时间。

B 详细的冻结测量结果

B.1 开发与工具输出证据

在公开 DEV（表4）中，PREFILL1 比 NATIVE8 低 0.1977 个百分点（95%区间 $[-0.5374, +0.0829]$）。尽管每个完整和原生八项输出都共享固定的前七个 ID，但仍存在五处分支差异。服务时间从 34.475 秒降至 14.033 秒；完整服务耗时 189.105 秒。共享初始化耗时 49.633 秒，完整解码器工作时间为 347.999 秒。完成的分配成本为 352 GPU秒。在任何模型调用之前，一个保留的采样器防护失败又消耗了 64 GPU秒；其仅针对运行时的修复接受了 vLLM 的贪婪 top_k 归一化，但未改变该方法。

在工具输出测试中，全部 617 个原始完整决策以及实际的前八个输出ID均与 NATIVE8 一致。工作线程时间从 157.121 秒降至 83.543 秒，减少了 46.83%；相应的加载时间分别为 54.911 秒和 36.900 秒。工作线程时间减去加载时间后分别为 102.210 秒和 46.644 秒；引擎调用时间分别为 84.889 秒和 29.833 秒。这些是单进程观测结果。

官方位置匹配会移除一个真实的正例完整预测。因此，NATIVE8 比官方完整F1值高出 0.1806 个百分点，组件区间为 $[0, +0.5733]$，同时与原始完整版本完全匹配。这一差异是后处理效应，并不能证明提前停止的理由更好。它相较于冻结阈值编码器的差距为 12.35 个百分点（$[8.93, 15.80]$）；在包含 373 个响应的无截断子集上，差距为 9.46 个百分点（$[5.42, 13.86]$）。该编码器的总体耗时为 106.125 秒，其中包括 19.366 秒的分词时间和 69.979 秒的前向调用时间。

读出 / 系统	Data2txt DEV 工具输出测试	代码代理 TEST	
2B 完整，官方偏移存在	94.5607	90.7285	83.4128
2B 完整，原始列表存在	94.4948	90.9091	83.5363
2B NATIVE8	94.5531	90.9091	83.6594
2B PREFILL1	94.3554	—	83.4304
307M BF16 编码器，原生argmax	87.3323	76.7361	76.5936
相同编码器 ，冻结的开发集阈值	$90.5292^{\dagger}$	78.5603	77.3960
始终为正	82.1367	66.5946	66.9528

表4：所有选定响应的 ResponseF1（百分比）。完整列复用相同的生成结果；编码器原生和固定阈值行对于每个响应均复用一次前向传播。 †固定阈值在公开 DEV 上选定，因此在该数据集上属于样本内估计，而不是新的折外估计。DEV 编码器测量值是缓存的精度控制项，并非其共享引擎解码器运行的额外实验组。短横线表示未运行。

代码代理 F1对比	Δ (pp)	95%区间
预填充1 –完整官方版本	+0.018	$[-0.273, +0.327]$
预填充1 − NATIVE8	−0.229	$[-0.548, +0.059]$
原生8 –完整官方版本	+0.247	$[-0.086, +0.605]$
预填充1 –编码器固定	+6.034	$[+3.969, +8.114]$
原生8 –编码器固定	+6.263	$[+4.187, +8.373]$

表5：针对全部2,015条代码代理响应的配对的1,613组件自助法区间。主预填充比较保留了可能的损失；这些并非等效性区间。

三个系统的完整分配成本为 440 个图形处理器秒。

B.2 前瞻性代码代理确认

全部响应的代码代理结果支持两个不同的效率结论。NATIVE8达到83.6594的F1值，而官方完整版本为83.4128。PREFILL1达到83.4304，其精确率为90.0571%，召回率为77.7120%，准确率为84.4665%。其混淆计数为788个真正例、87个假正例、226个假阴性和914个真阴性。相应的官方完整版本计数为787、86、227和915。因此，总体接近并不意味着错误完全相同。

全部 2,015 条完整版本和 NATIVE8 轨迹的前七个 ID 均相同。完整版本的前八个 ID 在 2,007 条响应中与 NATIVE8 相同；重建的预填充输出也在 2,007 条响应中与 NATIVE8 相同。两项比较各有八处差异，这说明理想的相同前缀论证不足以推断经验质量。尽管解码器未被限制为只能使用这些 ID，但所有预填充续写均未使用意料之外的诊断分支ID。完整官方版本处理后总共丢弃了 35 个跨度字符串；相较于原始存在情况，阳性响应判定有两处改变。我们在每项比较中都保留官方/原始版本之分。

同训练编码器通过原生argmax获得 76.5936 的 F1值，使用迁移阈值则获得 77.3960。它仅截断 441 条响应的上下文，移除了 1,453,394 个编码器词元；答案保持完整。在预先指定的无截断子集中（包含 1,574 条响应和 1,189 个组件），完整官方版本、 NATIVE8 和 PREFILL1 的 F1值分别为 82.3939、82.6389 和 82.5000，而固定阈值编码器的 F1 值为 76.5432。预填充与编码器差距为 5.9568 个百分点（$[3.6213, 8.2334]$）。因此，最大输入容量不等并不能单独解释观察到的模型差距。该比较仍无法区分容量、训练或架构的因果贡献。

B.3 实测计算和完整成本

表 6 给出了配对解码器计时器。相较于完整生成，NATIVE8将服务耗时降低了77.72%。PREFILL1将其降低了84.84%，即相较于NATIVE8降低了31.98%。更窄定义的引擎调用次数降幅分别为79.69%、87.71%和39.47%。这些百分比涉及常驻解码器的服务耗时，而非作业总时间。

通用解码器初始化需要 74.512 秒。源准备和分词需要 203.388 秒，资产验证需要 3.662 秒。完整的三臂解码器工作进程需要 919.893 秒。预填充仍会处理冗长的证据。我们不会通过任意分配共享工作来估算独立的单臂部署时间。

编码器单独耗时共计 360.969 秒：初始化耗时 11.807 秒，分词耗时 79.818 秒，前向调用耗时 234.720 秒，输入传输耗时 0.912 秒，投影与回读耗时 17.903 秒，其余时间用于验证和记录。其工作进程耗时减去初始化耗时后为 349.161

代码代理解码器读出	服务	引擎调用	输入词元	输出词元	逻辑总计
Full	446.407	434.534	8,605,121	95,134	8,700,255
NATIVE8	99.477	88.252	8,605,121	16,120	8,621,241
PREFILL1	67.668	53.419	8,619,226	2,015	8,621,241

表6：在完整代码代理来源上使用一个共享的常驻2B引擎。服务不包括共同的加载和准备；引擎调用墙钟时间包含在服务时间内。不会虚构共享初始化或分配的 GPU 秒在各组中的占比。单独报告的编码器前向计时器与服务测量的范围并不一致。

秒，其中包括分词耗时。若将这一数字直接与解码器的准备时间（不包括 67.668 秒的服务计时器）比较，就会夸大共同口径下的速度主张。因此，这里分别列出前向传播／引擎调用计时以及完整的工作进程总计，而不是将它们合并成一个貌似等价的延迟数值。

完整的共享解码器加编码器分配共计 1,296 GPU 秒，仅计一次，其中包括启动以及分配期间的 CPU 工作。分配前，解码器和编码器的 CPU 适配检查分别耗时223.27秒和109.92秒；GPU释放后，随后进行的 CPU ID审计耗时231.60秒。这些是额外的 CPU 成本，不计入上述 GPU 时长。读出过程没有产生付费API调用、新的训练或新权重。

NATIVE8 和 PREFILL1 的逻辑总计恰好为 8,621,241 个词元。预填充增加了 14,105 个输入令牌，并恰好减少了 14,105 个输出词元。即使相较于完整方案，逻辑词元总数也仅下降 0.91%，远低于服务时间的降幅。对于工具输出，总词元数减少了 0.44%。这些数值表明，变化体现在顺序执行的工作量上，而不意味着服务提供商的账单会等比例减少。编码器和解码器词元使用不同的分词器，不能视为可互换的计价单位。

B.4 前瞻性开发者文档扩展

在生成任何 README 实例或标签之前，我们就确定了相同的读数、检查点、上限、计划和编码器阈值。来自同一固定版本的 lettucedetect-readme 的全部 641 条回答均予以保留，并归入 638 个源派生词汇组件。这是一个新纳入项目留出集的来源，属于作者的任务范围：两个公开模型都在其 TRAIN 数据上训练。组件数量并不能证明仓库层面或语义上的独立性。

全部641个官方/原始版本完整判定、NATIVE8决策和PREFILL1决策均一致。每个原生前缀和预填充重建也都一致；这些是样本观测结果，而非总体等价性。

README 读出	F1值（%）	解码器服务（秒）
Full, 官方/原始版本	98.5782	85.2763
NATIVE8	98.5782	22.1382
PREFILL1	98.5782	8.3065
编码器，原生	90.2928	—
编码器，冻结阈值 0.0855		—

表7：完整 README源文件，不包含无效输出或未知请求成本。解码器行共享一个驻留引擎；两行编码器复用相同的前向调用。编码器计时的范围不同，因此在正文中单独报告。

REFILLREFILL1相较于完整版本将解码器服务减少了90.2593%，相较于NATIVE8（表7）减少了62.4789%。所有编码器成对组均可在不进行截断的情况下适配。因此，其较低的F1值不能单独归因于8k限制，但模型能力差距仍被继承。迁移阈值的表现劣于原生 argmax，并在未重新拟合的情况下予以保留。

共享解码器初始化耗时 57.5152 秒，分词/准备耗时 17.1854 秒，完整的三路工作进程耗时 207.5509 秒。编码器的完整工作进程耗时 19.0449 秒，其中初始化耗时 8.7356 秒，分词耗时 2.0495 秒；其前向调用耗时 3.1505 秒。特别是，其不包括初始化的 10.3093 秒仍包含准备阶段，因此不能直接与 PREFILL1 的准备阶段（不包括 8.3065 秒的服务）进行比较并据此宣称速度提升。合计分配成本为 232 GPU秒，仅计一次；发布后 CPU 实际 ID 审计还需 25.697 秒。NATIVE8 和 PREFILL1 各使用 502,567 个逻辑令牌，而完整使用 516,274 个：总词元数仅减少 2.655%，且没有相应的计费主张。

已发布卡片将 README 描述为精确率较低的初步编辑衍生参考答案，其中一些错误无需上下文即可检测（KRLab-sOrg，2026a）。逐示例代码测试审查

相同-s读出	告警 F1	服务（秒）	定位到的告警
Full	83.5363	425.7906	874/875
FIRSTQUOTE	83.5363	204.6371	861/875
PREFILL1	83.4304	63.9525	—

表 8：代码代理运行中的额外匹配前缀、匹配反分词。完整返回其首个有效的官方偏移量；FIRSTQUOTE 返回可选的首个引号偏移量，而 PREFILL1 仅用于告警。全部 2,015 条响应均予以保留。

描述并未延伸至此来源。因此，98.58 F1值表示与已发布参考答案的一致程度，并非经过独立验证的真实错误准确率、证据依赖性、无污染传输或整个领域的最先进水平。现已停止针对该来源调整方法。

B.5 可选的首个预测位置

在公开 DEV 上，固定的 FirstQuote 规则的告警 F1 值为 94.3554，而匹配的完整原始读取为 94.4212；服务耗时分别为 69.070 秒和 194.379 秒（-64.47%）。随后，我们将该规则原样传输到 1,613 个组件中的全部 2,015 个代码代理响应。这些响应此前的结果已经知晓：这是一项额外的功能比较，并非新的独立确认，也不是对表 4–6 的替代。

在此次运行中，完整原始读取与 FIRSTQUOTE 对每个响应告警的判断均一致（表 8）。FIRSTQUOTE 将服务耗时降低了 51.9395%，并将输出词元从 81,036 降至 32,606（-59.7636%），但总词元数仅降低了 0.55665%；两个实验组各自处理 8,619,226 个输入令牌。此次观察到的一致性并不能证明总体等价性。

位置覆盖率有所下降：FIRSTQUOTE 的 875 个告警中有 14 个不可映射，而完整版本只有 1 个。在 861 个共同定位的预测中，有 846 个引文及其偏移量一致，15 个存在差异；其中 13 个位置只有在完整生成后才能获得。偏移存在性 F1 为 82.8800，而完整官方版本为 83.4746，差异为 -0.5946 个百分点（组件区间 $[-1.0496, -0.2048]$）。与已发布的由编辑推导的跨度相比，首位置的微观字符精确率为 56.6728%，而 56.8585%；针对全部参考字符的召回率为 28.6444%，而 29.1267%。已定位的引文表示一个预测的错误位置，并非蕴含证书，也不是完整的跨度集合。

全部 6,045 个请求均通过实际 ID/哈希审计，没有无效告警或未知成本。共享初始化耗时 56.8347 秒，预热——

读出	主机生成 中位数 / 第95百分位（毫秒）	服务 中位数 / 第95百分位（毫秒）
Full	797.593/ 4,401.658	813.226/ 4,419.220
NATIVE8	175.062 / 190.273	192.419 / 212.430
PREFILL1	29.634 / 33.179	46.923 / 54.282

表9：旧版 DEV 上单请求热态时钟的描述性统计：每组 54 次调用，但仅有 18 个源单元，每个重复三次。此处不计入三次预热调用，另行计费。这些第95百分位值并非生产环境尾延迟估计。

比例为 228.7269，资产验证为 5.3583，工作线程时间为 1,008.4328。完整分配成本为 1,017 GPU秒，仅计一次；发布后的 CPU审计耗时 188.2404 秒。这些新的匹配运行计时不会取代前瞻性主实验的成本。

B.6 有界预热单请求测量

我们通过哈希选择 18 个已公开的 Data2txt DEV 组件及各自的一个答案，不使用标签。每个案例–组别对进行三轮测量，以平衡三个顺序位置；另取第 19 个组件，为每个组别各进行一次预热。165 次调用全部保留：162 次测量加三次预热。一个驻留引擎的最大序列数设为 1，且不使用前缀缓存。基础输入长度为 938–1,921 个词元（中位数 1,362）；预填充增加 7 个词元。

对于每个案例，我们分别取各组别三轮测量的中位数，用原生八项延迟除以预填充延迟，然后对这 18 个比率取中位数汇总，而不是用表9中的合并中位数相除。主机生成比率为 0.170154（减少 82.9846%），配对差值中位数为-145.517毫秒；服务的比率为 0.246363（减少 75.3637%），差值为 -145.485毫秒。全部 54 个逐轮三元组的判定均有效且一致，重建出的八词元前缀也完全相同。这表明功能一致性，并非新的 F1值或质量等价性证据。

主机生成时间同步调用 llm.generate。服务包括实际的重新编码、提示写入、可选的现有驱动程序上下文同步、解析以及完成/预测写入；其中包括生成前的进行中会话写入，但不包括最终的请求/批次/会话记录提交，也不会独立同步其他进程。共享初始化耗时 50.699 秒，准备耗时 18.812 秒，验证耗时 4.005 秒，完整

读出	主机生成 中位数 / 第95百分位（毫秒）	服务 中位数 / 第95百分位（毫秒）
Full	281.973/ 3,141.986	347.619 / 3,212.313
NATIVE8	202.756 / 208.772	254.792 / 286.389
PREFILL1	36.912 / 52.023	86.224 / 127.766

表10：对18个哈希选定的代码代理源组件进行温启动单例时钟测量，每个组件重复三次。每组测得的54次调用均保留；这项汇总不包括另外计费的三次预热调用。描述性p95并非生产环境尾延迟估计。

工作进程耗时 183.567 秒。预热服务总耗时为 4.205 秒；所有预热共计 189 GPU 秒。后续 CPU ID 审计耗时 19.866 秒，不计入该分配。18 个来源（而非 54 个独立案例）仅支持这一有界的主机时钟观察结果，不代表生产尾部指标、一般现代交互延迟或端到端代理延迟。

Modern 代码代理扩展。 我们在现有代码代理队列上重复相同的有限单例设计，使用新的固定哈希盐选出18个源组件，每个组件对应一个答案，另设第19个组件用于预热。选择仅依据组件键和问题ID，不使用标签、不根据输入长度进行选择，也不更改方法。该来源的历史质量结果已知；这只是额外的计时测量，并非新的质量确认。我们保留全部165次调用，并使用原有的代码代理用户封装、单张常驻的 H100 80GB、最大序列数为1、32k 上下文、BF16 急切执行，且不启用前缀缓存。选定的基础提示词长度为2,478–7,194个词元（中位数为4,203），超出了 DEV 短输入范围。

预先指定的18个每个qid三次预填充/原生八项比率中位数的中位数为0.176369，对主机生成而言（减少82.3631%），配对差值中位数为 -166.345毫秒。服务给出0.333779（66.6221%），差值为 -164.449毫秒。这些是配对汇总值，不同于将表10的合并中位数相除所得的结果。全部54组三元测量均具有有效且一致的分支决策，所有实际的完整首八项/原生八项/预填充重构ID也都一致。这一功能性观察并不取代完整2,015条响应批次确认中的八项决策变化，也不能证明总体质量相同。

共享初始化耗时 65.283 秒，词元化/准备耗时 24.704 秒，资产验证耗时 6.888 秒，工作线程总时间为 219.625 秒。三次预热调用共耗费 3.970 秒的服务时间，且计入 225 GPU 秒的分配中。GPU释放后，CPU 实际 ID 审计耗时 20.893 秒。测得的原生八项和预填充逻辑总量均为 240,888 个词元；378 个输出位置转为输入位置。这种有限的收益也适用于这些更长的现代输入；但这并不能证明冷启动、网络、并发服务或完整智能体延迟有所节省。

SDPA编码器单例对照。随后，我们在恰好这18个来源和轮次上运行固定的307M BF16编码器，并使用相同的第十九个来源进行预热：在独立分配中执行55次前向传播，不进行新的选择、标签、F1值或拟合。每次二值服务调用都会重新编码这对输入，构造并传输张量，使前向传播同步，在图形处理器上归约FP32答案对数几率间隔，读取一个标量，并提交最小告警。原生阈值和传输阈值决策共享该次前向传播。完整logits、实际ID/掩码以及精确的CPU读出检查均在二进制服务之外记录，并保留其成本。

编码器中位服务时间为 273.088 ms（前向传播 240.680 ms；编码 29.073 ms）。预先指定的 18 个按 qid 统计的三者中位编码器/解码器服务比率的中位数，相对于完整服务为 0.536918，相对于 NATIVE8 为 1.080135，相对于 PREFILL1 为 3.205585；后一个中位数的配对差为 $+189.084$ ms。全部 55 项标量/完整 logit 审计结果一致，且三轮编码器决策均稳定。这表明找到了一个有用的热运行点，而不是新的质量确认。18 个编码器上下文中有 4 个在 8,192 个词元处被截断，但保留了完整答案；分词器、后端、分配、预处理和日志边界各不相同，而历史解码器服务包含更繁重的日志记录。编码器加载时间更短：初始化 14.952 秒，准备 0.806 秒，资产 1.520 秒，预热服务 0.809 秒，工作进程 35.201 秒。测得的诊断工作在二进制服务之外增加了 0.372 秒；整个分配为 37 GPU秒，仅计数一次。这些时钟数据无法证明冷启动或普遍优势。

注意力实现对照。我们在一个常驻编码器中复用相同的 18 个来源、三轮和第 19 个来源预热，并轮换使用三种固定实现（表11）。所有实现均保留检查点、BF16、原始 QKV 和 FP32 RoPE、令牌 ID、答案掩码以及 8K 上下文

编码器实现	服务（毫秒）	前向传播（毫秒）
原版 SDPA	276.547	238.149
FA3，保留稠密掩码	233.254	203.850
FA3, 直接窗口	44.121	13.248

表 11：固定的 18 个源的计划下的热调用中位数：每种实现进行 54 次实测前向传播，另加一次计费的预热。服务包括全新的编码、传输、前向传播、二元读出和最少量的记录。不使用标签或新的 F1值。

编码器实现	原生	固定
BF16 SDPA（批次 4）	76.5936	77.3960
BF16 FA3/windows（批次 1）	76.7976	77.3345

表 12：完整代码代理响应 F1值（百分比）。原生版本与唯一的旧 DEV 阈值共用每种实现的前向传播；两者均未重新拟合。更早的解码器结果保持不变。

策略。两个项目桥接实现使用已安装的 FA3 内核，而非原生 HF FlashAttention2：一种保留稠密掩码构造；另一种直接表示相同的全局／局部窗口，并且仅对这些批次大小为 1 且全部有效的输入跳过冗余掩码。

预先指定的配对直接窗口/SDPA 服务比率中位数为 0.171300。相较历史 PREFILL1，服务比率为 0.497155，颠倒了 SDPA 速度排序，但须考虑上述跨分配和服务范围差异。所有 54 项配对的原生判定和冻结阈值判定在各实现间均一致，但所有 18 个来源的间隔都有变化；两个 FA3 变体彼此一致。固定的零 QK 基元验证了窗口支持和归一化，而非任意 QK 或质量等效性。稠密掩码主机计时是嵌套的入队计时，并非可相加的 GPU 内核剖析。新分配一次耗费 70 GPU秒，其中包括诊断；初始化耗时 28.503 秒，准备耗时 0.998 秒，完整工作线程时间为 67.683 秒。这项有界计时结果本身并不能证明该桥接实现具备完整队列质量。

完整优化编码器对照。现在，另一个单独冻结的轮次会按原始顺序，在全部 2,015 条代码代理响应和 1,613 个组件上评估直接窗口桥接实现，不使用标签、不进行拟合、不添加新案例，也不进行解码器重放。两种固定读出共享每次批量大小为一的前向传播。相同的 441 次上下文截断以及所有答案 ID/偏移量均保持不变。这为已公开的测试增加了更强的实现对照，但并非独立确认。

全部 2,015 项实际输入、完整 logits、标量和窗口执行审计均通过，没有无效调用或成本未知的调用。与缓存的 SDPA 相比，1,952 个间隔和 16 个原生/12 个固定阈值判定发生变化；原生 F1 上升 0.2040 个百分点，而固定阈值 F1 下降 0.0615 个百分点（Ta- ble12）。批次形状和注意力运算二者均有所不同，因此这些变化不能单独归因于内核，也无法证明质量等效性。PREFILL1 比优化的固定阈值编码器高 6.0958 个百分点（1,613 个组件的 95%区间 $[4.0262, 8.2156]$）；在未发生变化的 1,574 条无截断响应子集上，其得分为 82.5000，对比 76.4602，差距为 6.0398 个百分点（$[3.6800, 8.3340]$）。固定阈值编码器的召回率更高（84.12% 对 77.71%）；F1优势并不代表在每项指标上都占优，而且仍源自公开模型。

新的编码器服务总耗时为 100.167 秒，其中包括 65.800 秒的新鲜成对编码和 28.406 秒的同步前向调用；完整的 logits/ID诊断另外增加了经单独测量的 15.441 秒。初始化耗时为 21.400 秒，资产耗时为 2.477 秒，工作线程时间为 169.505 秒；完整分配一次需要 173 GPU 秒。发布后的本地评分耗时为 17.104 秒，且不调用模型。这些新鲜编码的服务总耗时与解码器不含准备阶段的总耗时适用范围不同，因此无法据此计算遵循共同协议的端到端加速比。同源单例结果反而显示，优化编码器的一个点位速度更快，但观测到的质量低于 2B读出，同时 2B 内部的解码增量仍然存在。

B.7 在公开 DEV 上直接进行单词元二元控制

较短的直接二元回答，是原生结构化读出的一个重要而简单的替代方案。我们预先固定一个对照，应用于全部 1,026 条已公开的 Data2txt DEV 响应（171 个来源组件），并使用相同的冻结 2B 检查点、事实、用户包装和分词器。其更短的系统指令要求：如果答案中有任何内容缺乏依据，则回答是；否则回答否。只有此对照会将生成限制为两个完全匹配的单词元 verbalizer：「是」（ID 9175）和「否」（ID 2665），并采用零温度 argmax，不拟合阈值。在构建模型之前，已确认两个实际的助手后续分支在每个输入上都符合 CPU 要求。

DEV 读出	F1	服务（秒）	引擎调用（秒）
NATIVE8	94.487	44.433	35.926
PREFILL1	94.355	17.098	10.048
直接二元分类1	3.026	16.095	9.530

表13：在一个新的常驻引擎中增加固定 DEV 对照。每组仍保留 1,026 个响应，且无无效决策。这些新观测结果不会取代先前的 DEV 比较，也不会增加独立的质量确认。

原生读出保持未掩码状态。未使用任何封闭测试来选择或优化该对照。

全部 3,078 个请求和 99 次轮换批次调用均已完成。Binary1 仅预测出 12 个阳性响应：11 个真正例和 1 个假阳性，另有 704 个假阴性，召回率为 1.54%。其输入较短，消耗 1,131,895 个词元，低于 PREFILL1 的 1,471,501 个词元；两者均生成 1,026 个词元。与 PREFILL1 相比，它节省了 1.004 秒服务时间，但检测质量低到无法使用（表 13）。因此，强制采用有效答案格式，并不能使这项直接提示迁移任务具备胜任能力。这套固定方案已封闭；这并不表明二元分类或所有二元提示都很弱。

在本次运行中，PREFILL1 相较于 NATIVE8 将服务时间缩短了 61.52%，但 F1值低了 0.1317 分（171分量区间$[-0.4156, +0.1436]$）。所有原生输出共用固定的前七个 ID；重建出的四条预填充轨迹则各不相同。共享初始化耗时 66.884 秒，分词器／准备时钟为 56.772 秒（其中包括分词器初始化的 23.393 秒），资产验证耗时 4.075 秒。各组准备分别耗时 10.860／11.546／10.972 秒；这些嵌套计时不再重复相加。工作进程总耗时为 230.005 秒，分配了 236 GPU秒；GPU释放后，后续 CPU ID审计耗时 48.223 秒。NATIVE8 仍是最强且实用的同模型短输出对照。这一格式敏感结果支持复用学习得到的原生分支，但无法单独确定本地化训练的因果作用。

B.8 观测到的长度依赖性

一项事后分析无需调用模型，并保留了全部 63 个配对代码智能体数据块，包括最后一个包含 31 条响应的代码智能体数据块。预填充/原生八项引擎调用比率与平均提示长度相关（考虑并列秩的斯皮尔曼相关系数为 0.923；服务为 0.881）。四个固定长度排名区间的生成比中位数分别为 0.5665、0.5969、0.6389 和 0.6673：相对节省幅度随长度增加而减小。所有 63 个数据块的生成速度都更快；服务在 62 个数据块中更快，但仍有一个比率为 2.411 的离群值。按臂位置汇总的结果差异小得多。这与提示处理摊销了节省下来的串行解码步骤相符，但这仍是观察性结果，而非因果证据或内核级证据；它无法仅凭长度解释跨工作负载差异。

C 辅助编码器到生成器的路由

该扩展是在结构化读出研究之后进行探索的。它复用了冻结模型，并非一种新的路由算法：置信度级联和验证级联已有先例（Chen 等人，2024；Aggarwal 等人，2024）。我们将其保留为辅助的检测器专用分配点，而不是主要贡献的前提。

冻结门控与强对照。 在公开的 Data2txt 开发集上，当原生编码器的绝对边际不大于固定截止阈值时，响应会升级到 PREFILL1。五个完整来源哈希折叠沿用从旧版 SDPA 训练折叠中选定的截止阈值：1.2890625、1.205078125、1.236328125、1.17578125 和 1.244140625。后续的 FA3 实现不会重新拟合这些阈值。未升级的预测使用唯一的旧版 DEV 阈值 -0.9401225447654724。对照组使用相同的校准编码器，并在每个折叠中严格采用相同的升级数量，选择固定盐标识符哈希中最小的那些。所有对比均保留所有响应；不存在预算、盐、随机种子或目标阈值搜索。校准哈希才是相关的强对照，而不是较弱的原生零值回退。

实际开发服务。 一次分配让两个模型同时驻留在一块图形处理器上，固定使用 0.70 的 vLLM 内存份额、一个活跃的解码器序列、BF16 精度，且不使用前缀缓存。另行计费的无标签编码器遍历会在生成任何新的解码器输出前，冻结每个折叠的实际计数和哈希路由。随后，三种策略按原始行顺序交替执行。两种级联策略在每个响应上都要重新执行编码器分词、传输、前向计算、FP32 边际投影和路由，包括未升级处理的响应。只有升级处理的案例才需要额外重新执行解码器分词、生成和解析。始终预填充策略只支付其自身必需的阶段。所有策略都在完成相同的最简告警写入和 fsync 后停止服务计时器。完整 logits/ID 诊断、来源资质、

DEV策略	F1值	2B次调用	服务（秒）
Always PREFILL1	94.4212	1,026	48.3604
不确定性路由	92.5395	252	30.8888
校准哈希	91.7942	252	30.8996

表 14：公开 DEV 上的实际共驻服务，1,026 个响应/171 个组件。这是一次探索性的质量–服务权衡，而非独立确认。

初始化、探测、预热和清理均单独计费；不会因路由而被视为节省。

不确定性比经过校准的等调用哈希高 0.7453 F1 分（条件组件区间 $[-0.2544, +1.8748]$），而服务总量几乎相同。与 always-PREFILL1 相比，其 F1 值低 1.8817 分（$[-2.8047, -0.9790]$），聚合服务减少 36.1279%（比率 0.638721，区间 $[0.607873, 0.670705]$）。尽管中位数更低，其描述性p95服务仍较差，为 67.047 毫秒，而对照为 53.677 毫秒。不确定性策略得到 645/34/70/277 个真正例/假正例/假负例/真负例，而哈希策略为 660/63/55/248：精确率有所提高，但召回率有所下降。无法据此得出质量相当或具有尾延迟优势的结论。

两种路由策略均实际调用了 252 次（24.5614%），没有路由漂移、无效决策、未知成本或审计问题。与旧SDPA缓存相比，962个编码器裕量以及8个原生决策和4个校准决策发生了变化；实际的始终预填充与其旧缓存相比，有3项决策不同。因此，实际计数和质量并不能由缓存值替代。全部1,531次实际解码器调用均通过发布后CPU ID审计。完整分配为327 GPU秒，其中包括两次初始化（58.4077+2.2011秒）、预检（12.9608秒）、资产（5.6857秒）、探测服务（21.9523秒）、预热、诊断、规划、原语和清理；工作线程时间为324.3946秒。解码器调用次数减少，并不意味着计费GPU节省会按比例增加。

最初的本地验证器使用编码器风格的紧凑JSON哈希来处理解码器的原生提示ID序列化，并在解析标签前拒绝了所有调用。另一个单独冻结的评分器仅修正了该序列表达式。原始来源、拒绝记录、修复记录和原始输出均予以保留；方法、掩码、指标、数据或模型调用均未更改。

Fixedmodern传输重放。 整个门控仅传输一次，传给已完成评估的代码代理

代码代理缓存策略	F1值	Planned2B 调用
Always calibrated编码器	77.3345	0
Always PREFILL1	83.4304	2,015
不确定性路由	82.0459	596
校准哈希	79.2220	596

表15：在全部 2,015 条响应/1,613 个组件上进行固定传输重放。预期调用未计入现代服务节省率。原始上下文截断和完整答案均予以保留。

队列，使用其符合条件的 FA3间隔和原始 PREFILL1决策。不适配代码代理队列。这是对现有输出的CPU组合，并非实际的级联执行，也不是新的独立确认。

该路由的表现比等调用哈希高出 2.8239 个 F1 分（$[1.3801, 4.3363]$），比校准编码器高出 4.7114（$[3.1638, 6.2716]$），但比始终使用 PREFILL1 低 1.3845（$[-2.8190, +0.0376]$）。全部 596 次计划内升级调用（29.5782%）均使用未更改的截止阈值和盐值。不确定性与哈希共有 194 次升级调用，且各自分别有 402 次独有调用。它们的真正例/假正例/假负例/真负例分别为 786/116/228/885 和 835/259/179/742。与 DEV 中一样，精确率和召回率存在权衡。此次重放耗时 3.3197 CPU 秒，新增模型调用、GPU 秒或 API 支出均为零。

实际的 Modern 级联服务。随后完成的固定测定在相同的 2,015 个代码代理响应和 1,613 个组件上执行全部三种策略（表16）。该测定保留 DEV门控、阈值、盐值、全部 441 个编码器上下文截断以及完整答案；不会拟合新的代码代理阈值或预算。无标签 FA3 探测器在生成任何新的解码器输出之前，确定每个折叠的实际计数和哈希路由。两个模型仍按照与 DEV 测定相同的固定内存份额常驻，策略按原始顺序每个响应轮换一次。两种路由策略都需在每个响应上执行新鲜编码器处理和路由，仅在升级时执行新的解码器处理；always-PREFILL1 则执行其自身所需的阶段。共享服务边界包括编码、传输、所需的前向传播、读出、路由、解析以及最少的告警写入/fsync。这是在先前暴露的队列上进行的实际执行，而非新的独立确认。

不确定性比经过校准的等调用哈希高出 2.8239 个 F1 分（1,613 组件条件 95% 区间 $[1.3801, 4.3363]$），而服务几乎完全相同：比率为 1.001237

代码代理策略	F1值 2B 调用服务		
Always PREFILL1	83.4921	2,015	150.5046
不确定性路由	82.0459	596	140.9961
校准哈希	79.2220	596	140.8218

表 16：完整的实际 Modern 级联服务，不含无效决策、未知成本或路由漂移。两种级联均使用 596 次调用（29.5782%）；其中包含编码器和路由成本。这些分别测量的时钟不会取代前瞻性读出研究。

（$[0.969791, 1.031352]$），因此该路由并不比哈希更快。与实际的始终预填充相比，不确定性低 1.4461 个 F1 分（$[-2.8776, -0.0135]$），同时将聚合服务降低 6.3178%（比率 0.936822，$[0.912262, 0.959927]$）。尽管解码器调用减少了 70.4218%，但这一测得的节省率仍低于 DEV 的 36.1279%。新产生的编码器工作和升级后的解码器工作均计入其中。不确定性的精确率/召回率为 87.1397%/77.5148%，而哈希的精确率/召回率为 76.3254%/82.3471%；精确率的提升伴随着召回率的下降。描述性服务 p95 差于始终预填充，分别为 165.158 毫秒和 125.966 毫秒。区间采用 2,000 次共享组件重采样（随机种子 20261025），并以这一个主机/轮次、实际路由和冻结输出为条件；它们不估计重复调度不确定性或生产尾部。

该探测完整保留了缓存的 FA3 边际值和两个编码器读出。实际路由与所有计划路由一致。不确定性和哈希保留其重放混淆计数；实际 PREFILL1 更改了一项缓存决策，增加一个真正例（789/87/225/914 真正例/假正例/假负例/真负例）。因此，其新的 83.4921 F1值与新的时钟一并报告，而不是替代主实验的 83.4304 F1值。全部 8,062 个请求均已完成，包括 2,015 个探测请求、6,045 个策略请求和两次预热，共进行 6,046 次编码器前向传播。全部 3,208 次解码器调用（包括预热）均通过发布后 CPU 实际 ID 审计，用时 88.452 秒；冻结本地评分额外耗费 21.991 CPU 秒，且未进行模型调用。

完整分配一次性成本为 999 个图形处理器·秒（工作进程 994.4089 秒），包括解码器/编码器初始化（60.7167/2.2613 秒）、分词预检（67.7822 秒）、资产（5.7789 秒）、无标签探测服务/诊断（98.4331/12.9180 秒）、编码器/解码器预热服务（0.3526/3.2833 秒），以及针对三种策略的完整 ID/对数几率诊断（4.7253/13.5760/14.1633 秒）、规划（0.1955 秒）、原语资格认定（0.1687 秒）和清理（1.1380 秒）。这些阶段会被单独保留，而不会计入预热部署节省量；嵌套时钟不会重复相加。在同一分配中驻留两个模型时，我们不会人为编造每种策略的图形处理器账单，也不会根据调用次数推断计费节省量。相较于等调用哈希，有意义的增量在于更好地分配现有探测器的能力，并以始终使用 PREFILL1 为基准，衡量质量–服务权衡。这一辅助结果不会改变原生读出贡献，也不能确立质量等效性或整个领域的最先进水平。

C.1 居中路由：开发与运行时隔离

探索性源折开发。 本小节中的所有测量均使用公开的 Data2txt DEV 队列：171 个完整源家族中的 1,026 个响应。五折重放通过训练折 F1 值拟合每个留出折的响应阈值，并在并列值时取较大的阈值。它在这些训练折上计算路由半径，然后将未经修改的方案应用于留出的源家族。哈希对照共享经过校准的回退机制，并使每条路由在其留出折中的实际调用次数相匹配。表17 保留了全部三个预先指定的预算和两个门控条件；在本次测量之前，25% 点是主要点。来源和缓存预测此前已经公开，因此这是探索性源折开发，而不是新的独立确认。此次重放没有新的模型调用或服务测量。

在主要点，居中路由比等调用哈希高出 2.8054 个 F1 分（171 个组件的条件 95% 区间 $[1.7168, 3.8944]$），比 Abs0 高出 1.6933 分（$[0.5498, 2.8316]$），同时计划调用次数为 250 次，而 Abs0 为 254 次。它仍比 always-PREFILL1 低 0.9040 分（$[-1.6698, -0.1972]$）。在此次重放中，50% 点略高于 always-generator 点，但仍作为预先指定的敏感性分析点保留，而不是在检查后被选为主要预算。区间以固定的探索性折叠预测为条件，不会在每个自助抽样样本中重新拟合校准。

物理部署拟合测定。 部署阈值 -0.935546875和居中半径

DEV分位数	中心化F1值	调用次数	等调用哈希F1值	Abs0 F1值	调用次数	等调用哈希F1值
10%	91.6894	95	89.8042	90.1583	103	89.7436
25%	93.5172	250	90.7119	91.8239	254	90.9091
50%	94.5607	519	92.6762	94.4483	516	92.6863

表17：探索性源折叠DEV重放。每个哈希列在各折叠内与其自身门控的调用次数相匹配；中心化和Abs0不必共享调用次数。始终预填充PREFILL1的F1值为94.4212，调用次数为1,026；训练折校准编码器的F1值为89.3847，且没有解码器调用。这些是缓存质量和计划调用次数，并非实际服务测量。

完整DEV策略	F1	2B调用	服务（s）
Always PREFILL1	94.4212	1,026	40.5638
校准编码器	90.5792	0	17.0746
Abs0	92.3960	257	26.1200
居中	94.2080	257	26.2490
校准哈希	91.7942	257	26.2282
居中，旧准备 94.2080		257	30.2876

表18：对全部 1,026 个响应进行的实际六臂部署拟合 DEV 测定。服务包括所需的新鲜编码、传输、前向传播／生成、读出、路由、解析以及最小化告警写入/fsync。共享的资格认定、初始化、探测、预热和完整诊断均单独计费。这些样本内物理得分并非表17中的来源折叠估计。

1。322265625 在所有公开 DEV 上拟合一次后即确定。这改变了估计范围：所得物理质量属于样本内质量，不得替代上文的源折结果。未居中的 25% 半径为 1.2578125，并采用相同的新校准回退；它不同于历史五截断值门控。一次单独计费的无标签编码器探测会在任何解码器输出之前固定源折哈希路由。随后，六个策略臂在一次共驻分配中，按照每个原始顺序响应轮换一次。

五个主要运行时臂对每个所需模型共享相同的单次全新准备。第六个居中臂保留了较早的准备路径，包括完整／有效的编码器资格验证，以及服务内部的解码器解码—重新编码检查。快速路径在其服务计时之外执行这些资格验证和身份检查，保留其成本，并在服务内部对每个被调用的模型重新进行词元化。它不会复用模型输入 ID。这样即可将准备开销与路由规则或模型能力的变化隔离开来。

每种方法各有 257 次实际调用时，居中路由的 F1 值比哈希高 2.4138 分（$[1.2859, 3.5276]$），比 Abs0 高 1.8120 分（$[0.9020, 2.7125]$）。与始终使用 PREFILL1 相比，其 F1 值低 0.2132 分（$[-0.8614, 0.4301]$），而完整预热服务减少了 35.2897%：比率为 0.647103（$[0.620627, 0.673513]$）。这一观察到的微小质量差异并不能证明质量等效。居中路由的总耗时略高于 Abs0（比率为 1.004939）和匹配哈希（1.000791）。与始终使用编码器相比，其 F1 值提高了 3.6288 分（$[2.5177, 4.7651]$），但预热服务用量增加了 53.7311%。其精确率为 94.0111%，低于 Abs0 的 94.8454%；召回率为 94.4056%，高于 Abs0 的 90.0699%。

居中的快速准备臂与旧准备臂在全部 1,026 个编码器输入 ID、边际、路由和最终决策上均一致；对于其 257 次被调用的解码器调用，两者的提示/生成 ID 也一致。改变准备方式使服务时间从 30.2876 秒降至 26.2490 秒，即降低 13.3343%（比率 0.866657，区间 $[0.861131, 0.873117]$）。这隔离了本次测定中的运行时开销，并未提供新的习得能力。居中服务中位数为 16.749 毫秒，而 always-PREFILL1 为 39.068 毫秒，但其描述性p95更差：56.924 对 45.427 毫秒。所有区间均使用 2,000 次成对的全家族重采样，并以拟合的部署策略、实际输出以及单个主机/轮次为条件；这些区间无法确立重复调度不确定性或生产环境尾部行为。

完整成本与保留失败。 全部 7,184 个请求均已完成：其中包括 6,156 个已测量的策略请求、1,026 个探测请求和两次预热。这些请求触发了 6,157 次编码器前向传播和 2,055 次解码器生成，且不存在无效决策、未知成本或路由漂移。全部 2,055 次实际解码器调用均通过独立的 CPU ID审计。共享的解码器/编码器初始化耗时 58.650/2.335 秒，工作进程内原生 ID 资格验证耗时 28.954 秒，资产验证耗时 5.417 秒，无标签探测服务耗时 18.075 秒，规划耗时 0.048 秒，窗口原语耗时 0.172 秒。六个实验组的完整 logits/ID 诊断在其预热服务之外额外耗时 30.026 秒；探测和预热诊断、源准备、最终刷新以及清理仍均有记录。这些是按阶段划分的实际边界，而不是对共同成本进行选择性的逐策略分摊，

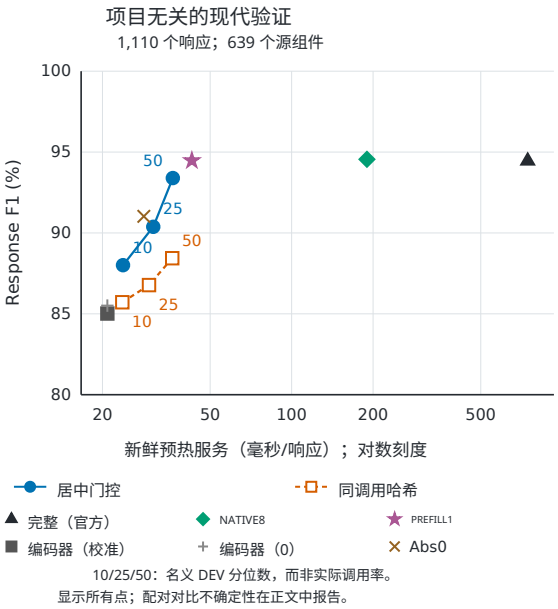


图4：所有预先指定的 Modern 质量–服务点，包括两个编码器阈值和每个 DEV 路由半径。线条连接固定预算；未拟合验证集选定的最优值。这些是汇总估计；完整表格附有配对对比区间。

而是按阶段划分的实际边界，而不是对共同成本进行选择性的逐策略分摊。

该工作进程运行了 474.582 秒，其完整分配共耗用 479 个实际 GPU秒，仅计一次。后续的 CPU ID审计和评分不计入该分配。在本次测定结束时，完整的探索台账共包含 114 项终结分配和 84,690 个实际 GPU秒（23.525 GPU 小时），其中包括较早发生的故障以及空闲和启动耗时；API 支出仍为 0 美元。35.29% 的热服务降幅不能套用于整个作业或项目总量，作为 GPU、金钱或能源节省。两次本地暂存检查均在提交作业前停止；相关记录仍予以保留，且两者均未消耗 GPU 分配。所有历史上未居中的结果、不良结果和草稿版本均予以保留。

C.2 前瞻性 Modern 现代确认完整策略矩阵

选择与参考答案边界。 在生成新的来源字符串或标签之前，选择过程已确定固定版本验证集划分中来自代码代理、开发者工具输出和 README 源文件的全部 1,117 条响应。仅源闭包合并了归一化后的问题／上下文精确匹配、上下文近重复项（五词 Jaccard 相似度至少为 0.85），并纳入白名单中的实例／文档标识。它还关联了已公开的 DEV 和三个历史现代 TEST 队列，并复用了其中的组件。五个组件中的七条工具响应关联到较早的工具队列；它们仍全部保留在执行流程和完整的次级评估中。固定的项目未关联主集包含 639 个组件中的 1,110 条响应。总体的确定不受参考答案、预测、长度或观测结果影响。

表19保留所有来源计数。这些单元构成保守的词汇/实例/文档闭包，但并不能证明语义或存储库独立性。在固定投影下，工具记录没有合格标识字段。上游 mmBERT 检查点选择明确使用了开发集幻觉词元 F1（Kovács 等人，2026年，第 6.1 节）；因此，编码器比较沿用了上游开发集。生成器的类似选择规则未作说明。这是项目未公开验证，而不是无污染的外部测试。其机器生成的参考答案并不继承该发布版本针对代码代理 TEST 的逐条审查声明；README 仍然是较低精确率的编辑派生证据。所有标签均为原始发布的跨度列表，并已核查精确的整数边界。它们的存在即为响应目标，不涉及新的事实真实性或蕴含关系判定。

冻结部署和共享服务。完整的十一臂评测使用固定的 DEV 阈值 -0.935546875、针对预先指定的 10/25/50% DEV 分位数的中心化半径 $0.541015625/1.322265625/2.384765625$，以及未中心化半径 1。2578125。不拟合 Modern 现代分位数或阈值。每条哈希路由在来源、现有组件哈希折叠和暴露状态分层方面，均与其中心化路由的解码器数量匹配。暴露分层确保这种匹配在主要和完整总体中都成立。即使某个敏感性分析点表现更好，主要运行点仍为 Centered25。

一次 H100 分配用于运行未改动的 2B 解码器和优化后的 BF16 FA3 编码器。全部 11 个实验组都通过同一个辅助程序，为每个所需模型进行全新准备；使用一个活跃的解码器序列、32k 解码器容量、不启用前缀缓存、随机种子为零，并采用不受限制的贪心解码。完整／原生八项／预填充上限分别为 1,024／8／1。编码器保留仅针对 8K 上下文的截断设置

来源	总体精确率 n	总体精确率	G 总体精确率正类主要	n 主要	G 主要正类
代码代理	190	176	95	190	176	95
工具输出	240	222	120	233	217	116
README	687	246	343	687	246	343
总计	1,117	644	558	1,110	639	554

表19：仅限冻结来源的总体。七条关联工具响应均保留在完整次级总体中；报告所有来源，不进行基于结果的替换。

以及完整答案偏移量。单独付费进行的完整编码器探测，会在解码器输出前使无标签哈希方案持久化；随后，11 种策略顺序会针对每个原始顺序响应轮换一次。全新服务涵盖必要的编码、传输、前向传播／生成、读出、路由、解析，以及最精简的告警写入/fsync。预热、初始化、完整资格认定、探测、详细诊断、最终刷新和清理都分别计时，并计入分配成本。

完整质量与原生读出。表20报告两个总体中每种原生视图及全部三种路由预算的精确率、召回率和响应 F1。Abs0、居中和哈希实验组采用相同的校准回退。在未改变的编码器前向传播中也报告原生零；在主总体上，其 F1 值为 85.5072，高于迁移后的 DEV 阈值编码器的 85.0168。因此，在新分布上，校准回退甚至无法胜过这一简单替代方案。

在主总体上，PREFILL1 的 F1 值为 94.4751，完整官方版本为 94.4547，原生八路／原始完整方案为 94.5522。与完整官方版本相比，其差值为 $+0.0204$ 个百分点；与原生八项相比为 -0.0770 。共享的完整全新服务耗时从 827.4728 秒降至 47.4990 秒（相较完整方案减少 94.2597%），从 210.7107 秒降至 47.4990 秒（相较原生八项减少 77.4577%）。表 22 给出了条件组件区间，而非等价性主张。主总体中 PREFILL1 的混淆计数为 513/19/41/537 真正例/假正例/假负例/真负例；完整官方版本为 511/17/43/539。总体 F1 相近，但错误分布仍有变化。

在全部 1,117 个响应中，完整原始读取与原生八项决策完全一致，且所有原生八步输出都保留了固定的七个语法 ID。预填充改变了三项原生八项决策。官方引文到偏移量的映射改变了一项完整原始读取决策；完整运行映射了 820 个引文跨度，并舍弃了七个不可映射的跨度，完整输出均可解析。这些只是解析器／轨迹观测，并不能证明蕴含关系。八词元和一词元上限触发属于预期的读出限制；没有完整生成达到 1,024 个词元的上限。已完成的矩阵中，无效决策、未知请求成本或路由漂移均为零，因此三项无效处理敏感性规则的结果一致。

主要路由结果与负迁移。Centered25 使用 250 次调用，F1 值达到 90.3795，高于精确调用次数相同的 Hash25（86.7761）： $+3.6034$ 分，区间 $[2.1127, 5.2207]$。它还将服务时间从 32.9830s 增至 34.1989s（比率 1.036865）。与 always-PREFILL1 相比，它的 F1 值低 4.0956 分，但预热服务时间减少了 28.0009%（比率 0.719991）。与校准编码器相比，它的 F1 值高 5.3627 分，但服务时间增加了 47.7875%。所有这些成本均包含每次路由策略响应时新鲜编码器的成本。

一个重要的反例是更简单的 Abs0 对照。它仅使用 216 次调用，耗时 31.5589s，F1 值便达到 91.0268。Centered25 的 F1 值低 0.6473 分，速度慢 8.3651%，同时多进行了 34 次调用。因此，尽管居中策略在 DEV 上结果为正，且优于随机哈希分配，但相较于这一强有力的对照，它并未带来 F1 值与服务表现的共同改善。它的精确率/召回率为 88.4283/92.4188%，与 Abs0 的 93.3586/88.8087% 不同：召回率的提升伴随着精确率的大幅下降。保留这一负迁移结果，而不通过另选阈值、来源或预算来修正。

Centered10 和 Centered50 仍是预先指定的曲线点，其 F1值 分别为 88.0069/93.3937，调用次数为 102/413，成本分别为 26.4375/40.3954 秒。两者均超过各自的等调用次数哈希，但在结果确定后，50% 点并未取代主要的 25% 点。Centered50 的汇总 F1值仍落后于 always-PREFILL1。表格21 给出了完整矩阵的实际成本和尾部指标。

PREFILL1 将其主要中位数/第95百分位服务延迟提升至 36.853/75.482 毫秒，相较于原生八项和完整方案更优，但路由策略在尾部指标上并未胜过它：Centered25 的第95百分位为 94.102 毫秒。Centered50 的第95百分位为 105.891 毫秒。完整表格保留了这些

读出/策略	主要 P	主要 R	主要 F1值	全部 P	全部 R	总体 F1值
Full, 官方	96.7803	92.2383	94.4547	96.7925	91.9355	94.3015
Full, raw	96.7864	92.4188	94.5522	96.7985	92.1147	94.3985
NATIVE8	96.7864	92.4188	94.5522	96.7985	92.1147	94.3985
PREFILL1	96.4286	92.5993	94.4751	96.4419	92.2939	94.3223
编码器，开发集阈值	79.6530	91.1552	85.0168	79.7488	91.0394	85.0209
编码器，原生零	85.8182	85.1986	85.5072	85.8951	85.1254	85.5086
Abs0	93.3586	88.8087	91.0268	93.3837	88.5305	90.8924
Centered10	84.2975	92.0578	88.0069	84.3750	91.9355	87.9931
Centered25	88.4283	92.4188	90.3795	88.4880	92.2939	90.3509
Centered50	93.6479	93.1408	93.3937	93.6709	92.8315	93.2493
Hash10	81.0289	90.9747	85.7143	81.1200	90.8602	85.7143
Hash25	83.2504	90.6137	86.7761	83.3333	90.5018	86.7698
Hash50	86.5285	90.4332	88.4378	86.5979	90.3226	88.4211

表20：所有计划的响应质量点（百分比）。完整官方/原始版本共享一个生成输出。两个编码器阈值共享一次前向传播。这些评估别名不增加物理调用。主要和完整总体相互重叠，并非独立的执行。

物理策略	2B 次调用 主要/全部	编码器输入 所有词元	解码器输入 所有词元	解码器输出 所有词元	服务 (s) 主要/全部	第95百分位（毫秒） 主要/全部
完整（一次生成）	1,110/1,117	0	2,404,792	36,551	827.473/830.474	1889.20/1874.83
NATIVE8	1,110/1,117	0	2,404,792	8,936	210.711/212.129	217.72/217.70
PREFILL1	1,110/1,117	0	2,412,611	1,117	47.499/47.804	75.48/75.44
编码器，开发集阈值	0/0	2,391,954	0	0	23.141/23.299	43.17/43.11
Abs0	216/217	2,391,954	671,201	217	31.559/31.747	75.11/74.82
Centered10	102/103	2,391,954	341,690	103	26.437/26.631	53.80/54.01
Centered25	250/253	2,391,954	833,150	253	34.199/34.484	94.10/93.88
Centered50	413/417	2,391,954	1,286,568	417	40.395/40.709	105.89/105.30
Hash10	102/103	2,391,954	301,249	103	26.277/26.471	53.15/53.62
Hash25	250/253	2,391,954	769,050	253	32.983/33.268	90.36/89.66
Hash50	413/417	2,391,954	1,232,347	417	40.186/40.529	104.25/104.06

表 21：完整新测成本。词元涵盖该策略测量的全部 1,117 个请求；主要/全部条目是同一次运行的重叠视图。原始/官方完整视图和原生/校准编码器视图分别共用其对应行。探测、预热和冷启动/诊断成本在表 28中单独列出。描述性p95并非生产尾部估计。

不利数值，而不是根据平均调用减少来推断尾部指标。

主要原生八路和预填充各自恰好消耗 2,391,172 个逻辑解码器输入加输出词元：预填充将 7,770 个词元从输出转移到输入。所有响应的总数相同，为 2,413,728，其中转移了 7,819 个词元。相比之下，主要 Centered25 在两个模型中的逻辑位置总和为 3,190,274，比始终预填充的解码器位置数高出 33.4188%，尽管其测得服务成本较低。模型大小不同，因此这个未加权总和不能用于比较 FLOP；调用次数和词元数都无法证明分配的GPU、API 价格或能源节省。

来源异质性，保留每个来源。汇总主要结果包含 687 个 README响应。表格23–25 因此展示所有来源特定的精确率/召回率/F1值，而不仅仅是汇总得分。代码和README没有与曝光相关的排除项，因此其主要及完整来源评估完全相同。工具表同时给出了主要的 233 个响应和完整的 240 个响应。表格26保留来源特定调用和完整预热服务。

居中主要策略并非在所有方面都优于 Abs0：代码F1为 80.8081，而 82.1622；README 为 94.9153，而 96.0699；工具F1为 84.5815，而 82.2967。Centered25 在每个来源上的调用次数都多于 Abs0。其代码服务耗时（11.0650 秒）已经接近始终预填充（11.1453 秒）。因此，相对于哈希的汇总增益并不能证明存在普遍路由升级。预先声明的 Centered50 点位所观测到的工具F1高于始终预填充，但其汇总F1仍然较低；这一来源特定观察仅作报告，不涉及事后策略或来源选择。

前瞻性定义的无截断次要分析。编码器在全部 1,117 个上下文中截断了 131 个

主要对比	ΔF1值（pp）	组件95%区间	服务比率	组件95%区间
预填充1 –完整官方版本	0.0204	$[-0.3364, 0.4008]$	0.057403	$[0.053997, 0.061365]$
预填充1 –完整原始读取	-0.0770	$[-0.3627, 0.2120]$	0.057403	$[0.053997, 0.061365]$
预填充1 − NATIVE8	-0.0770	$[-0.3627, 0.2120]$	0.225423	$[0.218730, 0.234101]$
居中10 − Hash10	2.2926	$[1.0647, 3.5963]$	1.006095	$[0.957551, 1.058292]$
居中25 − Hash25	3.6034	$[2.1127, 5.2207]$	1.036865	$[0.972890, 1.110121]$
居中50 − Hash50	4.9559	$[3.3058, 6.5479]$	1.005203	$[0.959123, 1.053410]$
居中25 − Abs0	-0.6473	$[-1.9582, 0.6210]$	1.083651	$[1.028054, 1.148355]$
居中25 − PREFILL1	-4.0956	$[-5.6205, -2.6624]$	0.719991	$[0.677906, 0.769094]$
居中25 − 编码器	5.3627	$[3.8076, 6.9942]$	1.477875	$[1.394933, 1.570663]$

表22：所有预先指定的主要对比。比率为第一种策略除以第二种策略。区间采用 2,000 次共享整组件重采样，随机种子为 20261025；条件是固定的方法、实际输出/路由以及单个主机/轮次。它们不会重新拟合 DEV 校准，也不会估计重复的调度变异。

读出/策略	精确率	召回率	F1
Full, 官方	89.5349	81.0526	85.0829
完整，原始	89.6552	82.1053	85.7143
NATIVE8	89.6552	82.1053	85.7143
PREFILL1	88.7640	83.1579	85.8696
编码器，开发集阈值	66.1290	86.3158	74.8858
编码器，原生零	75.0000	75.7895	75.3927
Abs0	84.4444	80.0000	82.1622
Centered10	71.3043	86.3158	78.0952
Centered25	77.6699	84.2105	80.8081
Centered50	84.3750	85.2632	84.8168
Hash10	66.3934	85.2632	74.6544
Hash25	70.7965	84.2105	76.9231
Hash50	76.7677	80.0000	78.3505

表23：代码代理来源，全部190个响应/176个组件，包括95个已发布的正例。主要和完整总体相同；所有读数和路由预算均保留。数值以百分比表示。

条响应，且未截断任何答案。固定的主要无编码器截断子集包含 979 条响应/514 个组件和 485 个已发布阳性样本。表27 报告了该子集中的所有读出和预算。预填充F1为 95.4879，高于校准编码器的 86.3724 和原生编码器的 87.2951，因此，继承而来的能力差距不能仅用编码器上下文限制来解释。Centered25 仍低于 Abs0（91.6583 对 92.3400），且调用和服务开销更多。哈希预算针对预先声明的总体，按来源/折叠/暴露分层进行了匹配，之后并未针对这个长度子集重新匹配；例如，此处 Centered25 有 191 次调用，而 Hash25 有 198 次。因此，这些子集对比采用各自实际的预算，不称为调用次数完全匹配的比较。该子集绝不取代完整的主要分析。

完整执行、成本与验证器修复。全部 13,408 条计划请求记录均已完成：12,287 条测量请求、1,117 条编码器探测请求，以及 4 次预热。实际调用总计 10,054 次编码器前向传播和 5,117 次解码器生成（包括预热）。路由漂移未破坏主总体或完整总体上的精确哈希匹配。实际输入/输出审计和采样器审计保留了每一次已提交调用；全部 5,117 次解码器调用均通过独立的 CPU ID审计。审计器报告的内部耗时为 78.995 秒；采集器测得其完整审计进程耗时为 80.645 秒。这些嵌套时钟耗时不予相加。最终的本地评分器耗费 32.004 CPU 秒，且未增加推理。提交前的独立 CPU 工作中，仅编码器资格认定耗时 34.894 秒，原生预检工作器耗时 118.961 秒，均未使用模型权重、GPU 分配或标签。后者包含其自身嵌套的、耗时 100.256 秒的令牌/ID预检。这些外部 CPU 时钟耗时与表28中实际 GPU 工作器的 103.922 秒令牌/ID资格认定耗时不同；后者仍计入完整分配。表格保留了该分配的其他阶段边界。

第一个冻结的本地验证器在参考访问前就拒绝了，因为它使用不匹配的配置键查找工作进程的全重复防护。原始拒绝结果和代码均予以保留。另行记录的 v2验证器仅将该键查找改为工作进程记录的字段；原有队列、模型输出、路由、指标、自助法和成本均保持不变。没有新的推理、重新生成或方法修复。它在评分前通过了同一项独立的参考访问前审计。

完整工作进程运行了 1,903.174 秒；按终结主作业核算，分配了一台图形处理器，实际运行 1,907 秒，且仅计一次。在所有记录阶段中，审计保留了 21,529,080 个编码器输入位置、12,662,380 个解码器输入位置和 48,386 个生成的词元。其中包括付费探测和预热；按策略划分的词元表仅涵盖测量请求。此

读出/策略	主要 P	主要 R	主要 F1值	全部 P	全部 R	总体 F1值
Full, 官方	95.0000	81.8966	87.9630	95.0980	80.8333	87.3874
Full, raw	95.0000	81.8966	87.9630	95.0980	80.8333	87.3874
NATIVE8	95.0000	81.8966	87.9630	95.0980	80.8333	87.3874
PREFILL1	95.0000	81.8966	87.9630	95.0980	80.8333	87.3874
编码器 , 开发集阈值	78.1513	80.1724	79.1489	78.6885	80.0000	79.3388
编码器 , 原生零	81.1881	70.6897	75.5760	81.7308	70.8333	75.8929
Abs0	92.4731	74.1379	82.2967	92.6316	73.3333	81.8605
Centered10	83.1858	81.0345	82.0961	83.6207	80.8333	82.2034
Centered25	86.4865	82.7586	84.5815	86.8421	82.5000	84.6154
Centered50	93.3962	85.3448	89.1892	93.5185	84.1667	88.5965
Hash10	80.7018	79.3103	80.0000	81.1966	79.1667	80.1688
Hash25	85.0467	78.4483	81.6143	85.4545	78.3333	81.7391
Hash50	89.4231	80.1724	84.5455	89.7196	80.0000	84.5815

表24：工具来源：固定主项233/217个组件，完整项240/222。所有来源和全部七个关联工具响应均予以保留；这些总体存在重叠。数值为百分比。

读出/策略	精确率	召回率	F1
Full, 官方	99.1228	98.8338	98.9781
Full, raw	99.1228	98.8338	98.9781
NATIVE8	99.1228	98.8338	98.9781
PREFILL1	98.8338	98.8338	98.8338
编码器 , 开发集阈值	84.3990	96.2099	89.9183
编码器 , 原生零	90.0850	92.7114	91.3793
Abs0	95.9302	96.2099	96.0699
Centered10	88.5942	97.3761	92.7778
Centered25	92.0548	97.9592	94.9153
Centered50	96.2751	97.9592	97.1098
Hash10	85.7513	96.5015	90.8093
Hash25	86.4230	96.5015	91.1846
Hash50	88.2979	96.7930	92.3505

表25：README源文件，全部687个响应/246个组件，其中包括343个已发布的正例。主要和完整总体相同；所有读数和路由预算保持不变。数值均为百分比。

次确认结束时，整个项目包含 115 次终止分配和 86,597 个实际图形处理器秒（24.0547 个图形处理器小时），其中包括已放弃的方向、启动、故障和空闲工作。API 支出为 0 美元。作为计费主张，预热阶段的缩减量不会乘入整个作业或项目成本。

在常规的全新准备下，原生读出贡献在此得到独立支持。居中分配相对于等调用哈希仍是一个有用的对照点，但未能通过其更强的Abs0迁移对比；所有预算、来源差异、尾部成本以及这一否定结果均予以保留。此次开启的确认并未产生新的阈值/提示/随机种子/来源资源线索。相关信号仍是已发布参考答案的响应告警，不能证明事实真相、完整定位或执行成功。

D 条件前缀稳定性与有限样本核算

条件因果前缀引理。令 $g(p)$ 表示作用于完整词元前缀 p的确定性贪心下一词元规则，其中包括固定的并列值处理规则。假设从 p 开始的普通原生运行生成了语法前缀 s。如果预填充运行从完全相同的完整前缀 $p \,\|\, s$ 开始，并使用相同的下一词元函数，同时推理算术和相关状态均相同，则其延续结果等于普通运行的延续结果。第一个下一词元参数相同；在每个共同的下一词元之后，均可通过归纳法得出结论。因此，在这些前提下，由该前缀和下一词元决定的解析器会返回相同的分支。这是一个条件恒等式，并不保证两条图形处理器执行路径会计算出完全相同的 logits，也不保证生成输出在事实上的正确性。

一个充分数值条件。对于一个相同的完整前缀，设 ℓ 和 ℓ 是两个有限的全词汇表logit向量，且满足 $\|\ell - \ell\|_\infty \le \epsilon$。如果原始的最高项与次高项间隔超过$2\epsilon$，则最高词元不可能改变：最高logit最多可下降 ϵ，而每个竞争logit最多可上升 ϵ。对于更长的共同轨迹，该条件必须在每个相关的延续前缀处成立。我们的封闭轨迹不包含全词汇表logits，也无法根据该条件提供经验ϵ 或数值证书。仅凭词元一致性无法估计上述任一量。

无需个体标签的精确 F1值变化界。考虑具有 P 正参考标签的 n 个响应，以及A 和 B布尔正类预测集合。将这些响应划分为 $C = A \cap B$,

物理策略	代码 K	代码（秒）	工具 K 主要/全部	工具服务（秒） 主要/全部	README K	README（秒）
完整（一次生成）	190	223.420	233/240	138.642/141.643	687	465.410
NATIVE8	190	38.538	233/240	46.150/47.568	687	126.023
PREFILL1	190	11.145	233/240	12.297/12.602	687	24.057
编码器 , 开发集阈值	0	6.504	0/0	5.729/5.887	0	10.908
Abs0	64	9.849	62/63	8.523/8.712	90	13.187
Centered10	30	8.012	29/30	6.966/7.159	43	11.459
Centered25	71	11.065	78/81	9.648/9.933	101	13.486
Centered50	116	12.893	124/128	11.750/12.063	173	15.753
Hash10	30	7.922	29/30	6.804/6.998	43	11.551
Hash25	71	10.404	78/81	9.232/9.518	101	13.347
Hash50	116	13.035	124/128	11.389/11.731	173	15.763

表26：按来源划分的实际解码器调用和完整预热服务。代码/README 的主要来源总体与完整来源总体一致。词元和所有阶段时钟均保留在聚合产物中；汇总表并未掩盖依赖来源的成本。

读出/策略	精确率	召回率	F1值 2B 调用 服务（秒）	第95百分位（毫秒）	
Full, 官方	97.4304	93.8144 95.5882	979	684.6159	1742.99
Full, raw	97.4304	93.8144 95.5882	979	684.6159	1742.99
NATIVE8	97.4304	93.8144 95.5882	979	182.6017	211.01
PREFILL1	97.2222	93.8144 95.4879	979	37.1757	48.22
编码器 , 开发集阈值	80.7899	92.7835 86.3724	0	17.5275	28.41
编码器 , 原生零	86.7617	87.8351 87.2951	0	17.5275	28.41
Abs0	94.0171	90.7216 92.3400	174	22.9145	59.96
Centered10	85.3659	93.8144 89.3910	77	19.1291	48.14
Centered25	89.4118	94.0206 91.6583	191	24.1784	62.70
Centered50	94.3983	93.8144 94.1055	329	28.4857	70.21
Hash10	82.4176	92.7835 87.2939	83	19.5099	50.74
Hash25	84.1808	92.1649 87.9921	198	23.6638	62.20
Hash50	87.1595	92.3711 89.6897	334	28.6980	70.05

表27：仅源无编码器截断主要子集（979/514）上的所有预先指定读数和预算。完整别名和编码器阈值视图复用相同的实际成本。该子集并非单独的推理实验。

$E_A = A \setminus B$、 $E_B = B \setminus A$和 $O = (A \cup B)^c$，其大小分别为 m_C、 m_A、 m_B、 m_O。令x_j 表示组 j中的正标签数量。这些整数满足 $0 \le x_j \le m_j$ 和 $\sum_j x_j = P$。定义 $\alpha = 2/(P + |A|)$ 和 $\beta = 2/(P + |B|)$；当分母为零时，将系数设为零，与该退化情况下 F1值为零保持一致。则

$$F_1(B) - F_1(A) = (\beta - \alpha)x_C - \alpha x_A + \beta x_B. \quad (3)$$

因此，$(C,\ E_A,\ E_B,\ O)$ 的系数为 $(\beta - \alpha,\ -\alpha,\ \beta,\ 0)$。精确最大值可通过按系数从高到低依次向各组填入 P 个整数标签单位，直至达到各组容量来求得；按系数从低到高则可得到精确最小值。如果低系数组中有一个单位，而高系数组仍有容量，将该单位转移过去不会降低目标函数，这证明了极值构造成立。每种可行整数分组分配都能通过某种将标签分配给响应的方式实现，因此两个端点都可以达到。该

该过程只使用 n，P，$|A|$，$|B|$ 和 $|A \cap B|$，无需单独的参考标签。

该边界是有限样本核算，而非置信区间或总体质量等价性检验。单独使用各个已发布标签，以归因观测到的真阳性和假阳性变化，并确定观测差异在该边界内的位置。该分析保留了此前确定的每个总体。报告未知输出，且不会触发案例删除以获得布尔边界结果。少量发生变化的预测可能机械性地限制 F1值的变化；但这并不能证明存在证据依赖性、蕴含、真实错误准确率或新获得的模型能力。

观测分支与映射归因。这项补充分析复用了封闭式调制确认的输出；未增加模型调用。此前固定的九个总体均保持不变。在主要的1,110个响应中，完整生成的前八个词元分支与其最终原始列表告警

阶段	秒
解码器初始化	40.204
编码器初始化	1.711
工作进程内词元/ID资格验证	103.922
资产验证	6.017
探测服务	24.056
探测诊断	5.225
四次预热，服务	3.972
四次预热，诊断	0.020
测量臂诊断	93.166
路径规划	0.173
窗口原语	0.221
自有引擎关闭	3.808
完整工作进程墙钟时间	1903.174
实际主分配（图形处理器秒）	1,907

表28：付费现代确认阶段。策略预热时钟在表21中单独报告；整个工作进程/主分配包括所有阶段、剩余的准备/最终刷新/清理工作以及空闲时间。嵌套时钟不会再次计入，共同成本也不会选择性地归因于某项策略。发布后的 CPU审计/评分不计入图形处理器分配。

在每个响应上均一致。官方偏移量映射器另行移除一个真正例，使F1值变化 -0.097456 个百分点。原生八项与语法预填充仅在三个响应上不同，且均为从负向转为正向：多出一个真正例和两个假正例。它们观测到的F1值差异为 -0.077032 个百分点。仅根据总体中的正例数量以及这些预测集的大小和重叠情况，可达到的精确范围为$[-0.269866, +0.552486]$ 个百分点。这说明为何少数告警变化就能限定总体质量变化幅度较小。证据条件化在固定干预测定中另行考察；配对区间描述了这一有限预测比较。原始代码代理轨迹差异以及所有不利的传输结果均未改变。

E 证据干预与平衡运行时控制

上游开发暴露与参考答案来源。 额外的 Modern 队列是项目未启封验证集，而不是干净的外部测试集。在第 6.1 节中，Kovács 等人（2026年）明确根据上游统一划分上的开发集幻觉词元F1值选择 mmBERT 检查点。因此，编码器在这一划分上受到了上游开发数据的暴露，尽管我们的队列选择、来源闭合范围和比较方案，都是在我们的项目查看其响应或标签之前固定的。所引用的论文没有说明针对 Qwen3.5-2B 的类似检查点选择规则，我们也不将这一编码器归属说明扩展到该生成器。第 8 节还区分了对代码代理 TEST 进行的模型辅助作者审查，以及在开发集和非代码测试源中自动检查的注入标签。因此，在该标注溯源下，已发布的跨度列表是否存在，定义了每个读出共同且可复现的质量目标。

受控证据平衡测定的范围。
我们的独立构造在 24 个工具返回族中，将两个记录的目标字段值分别与两个固定的字面值响应交叉组合，并保留全部 96 种组合。每个精确响应和每个精确上下文在其所属族中都分别与每个标签出现一次。排除不透明标识符后，仅查看响应或仅查看上下文的分类器，在这个完整的平衡池上的经验准确率最高为 50%。标签通过解析渲染后的 JSON 目标字段以及渲染后响应中的字面值进行检查，而不是假设实际执行了工具。这样可以消除单侧标签不平衡，并能够有界地测试冻结的读出是否会随所提供的事实正确变化。测得的读出决策将依据这一字面一致性目标进行评估。完整的交叉池可确定决策是否会响应所提供的值；其构造单位是工具返回族，每个族包含四个相关输入。

前瞻性证据干预协议。六个开发者工具领域中的每个领域分别提供四个固定测试夹具，共构成24个族：失败测试数量、命令退出代码、文件数量、配置标量、制品路径和一个 Git 字段。每个族包含两个 JSON目标字段值，以及两个分别声称对应字面值的响应。将它们交叉组合后，得到两个受支持响应和两个不受支持响应。当目标值发生变化时，请求、干扰字段和响应措辞保持不变；在证据翻转对中，完整答案也保持不变。全部24个四元组、48个固定答案对和96个输入均予以保留，包括其中的任何故障。不透明标识符不会提供给检测器。未修改的原生包装器接收完整的给定记录和答案，但不会接收本地评估器的标签或潜在状态。没有检查任何模型输出，以选择测试夹具、前缀、提示、阈值或附加样本。

冻结完整读出、原生八项读出和 syntax-prefill 读出分别在每个输入上运行一次，并在单序列引擎下按照输入索引轮换实验组顺序。完整原始列表和官方偏移告警复用同一次完整生成。主要汇总指标是完全正确四元组的数量，以及其中两个证据状态均获得正确相反判断的固定答案对数量。准确率、F1值、混淆矩阵、各个领域、无效输出敏感度和读出不一致也会与这些族/对汇总结果一并报告。这些刻意构造的族是测量单位，而不是96份随机抽样且彼此独立的自然报告。

固定的仅基于来源的运行时选择。独立的计时池重复使用了 24 条此前已关闭的自然响应，且未读取其质量标签、预测或实测时间。在三种现代来源类型中，符合条件的未关联项目的行按未截断的编码器对令牌计数和标识符排序。每个长度四分位数均贡献两条通过固定盐哈希选出的行，且来自不同的源组件。全部 24 条均予以保留，并交错编入三个固定的八条一组的批次。用于选择的长度是仅基于来源的分层指标，而非解码器令牌计数。这些响应此前已被使用，因此无法据此提出新的自然质量确认声明。

对于每个批次，完整/原生八项/预填充的全部六种排列，分别在两个顺序引擎下执行，最大活动序列数设为 1 和 8。每种排列执行前，都会单独付费调用该批次首行的预填充，并紧接在排列之前执行；这样可固定首个实验臂的前置项，同时让所有排列平衡各位置及顺序内的前置项。每个引擎均各自接收三次首行预热。两者都显式使用 32,768 个批处理令牌和分块预填充、BF16、即时执行、随机种子 0，且不使用前缀缓存。这一新的鲁棒性设置并未复现先前的默认调度器，也无法验证其确切的速度百分比。引擎一在同一作业中先于引擎八运行，因此跨引擎比较仍存在时间/顺序混杂因素。

完整的计时与核算。 完整计划包含 438 次生成调用和 1,194 条已提交序列：288 条证据干预序列、864 条自然计时序列、36 个条件器，以及六次预热。108 次实际的八响应计时调用中，每次都有一个整批服务时钟，该时钟在准备新的必需输入之前开始，并在生成、解析以及完成八次最简警报写入和 flush/fsync 调用后结束。将此时钟除以八，得到的是摊销服务时间，而非单项延迟。我们报告每种引擎／顺序／批次的结果，以及描述性比率范围。三个源批次各有六个重复顺序，并不代表 108 个相互独立的自然样本，也不能作为生产尾部估计。来源资质确认、两次初始化、完整的实际 ID 诊断、条件器、预热、关闭、故障和分配空闲时间均计入成本；发布后 CPU 审计和评分则单独计算。我们不将任何预热服务比率视为分配的 GPU、货币或能源节省率。

证据条件化结果。四种读出——完整官方偏移存在性、完整原始列表存在性、NATIVE8，以及 PREFILL1——均将全部96个构造输入正确分类。对于每种读出，全部24个四元组均完全正确，且在证据变化后，全部48个固定答案对均获得正确的相反判定。表29报告了每个预先指定领域。每种读出均有48个真正例、48个真阴性、零个假正例和零个假阴性；准确率、精确率、召回率和F1值均为100%。无效结果视为负类、无效结果视为正类以及严格将无效结果始终视为错误的汇总结果彼此一致，因为不存在无效输出。完整官方版本和原始告警复用同一次生成，六组两两读出比较在全部96个输入上均一致。

这一完整的有限样本池结果排除了针对这些构造记录的确定性仅依据答案或仅依据上下文的决策规则：在没有不透明标识符的情况下，任一特征单独使用时，准确率最高仅为50%，且无法得到任何完全正确的四元组。该结果表明，缩短读出保留了冻结检测器在本测定中的字面证据条件化行为。这96个交叉输入构成24个经过刻意构造的族，因此该结果是对字面条件性的完整样本池测量，同时在解释中保留了族依赖性。

平衡顺序运行时间结果。 表30 给出了每个引擎下的三个实验组。每个实验组在每个引擎下都有 18 次实际的八响应计时调用：三个固定源批次在全部六种顺序下重复运行，共计 144 个提交序列。完整服务总耗时包括全新准备、生成、解析和最少量的告警写入。在最大活动序列数为 1 和 8 时，预填充/完整服务比率分别为 0.047143 和 0.074726；预填充/原生八项比率分别为 0.230934 和 0.618659。

领域	正确四元组对		
失败测试数量	16/16	4/4	8/8
命令退出代码	16/16	4/4	8/8
文件数量	16/16	4/4	8/8
配置标量	16/16	4/4	8/8
制品路径	16/16	4/4	8/8
Git 字段	16/16	4/4	8/8
All	96/96	24/24	48/48

表29：完整证据干预结果。每个条目分别适用于完整官方版本、完整原始读取、原生八项和 Prefill-one。“四元组”要求四项判定全部正确；“成对组”要求两个固定答案的证据状态均正确，且判定相反。所有输入和所有领域均予以保留。

因此，相对于完整服务，测得的预填充缩减分别为 95.29% 和 92.53%；相对于原生八项，缩减分别为 76.91% 和 38.13%。在八个活动序列的设置下，后者的收益较小。这些引擎内对比保留了所有顺序，而不是选择运行最快的顺序。

表31报告了两种引擎的全部六种顺序。表中每个比值均为同三个源批次的服务总量之比。另据此，18 个成对批次比值（三个批次乘以六种重复顺序）的范围如下。使用一个活动序列时，预填充/完整范围为 0.034517 至 0.055414，预填充/原生八项范围为 0.180695 至 0.312688，原生八项/完整范围为 0.177218 至 0.251142。使用八个活动序列时，相应范围为 0.030627–0.121499、0.445771–0.788292 和 0.068707–0.164406。这些范围描述的是这些固定重复次数；范围本身以及 108 次计时调用均不是独立样本不确定性区间，也不是生产尾部估计。

对于 24 行自然行中的每一行以及每个实验臂，在两种引擎和所有顺序下的 12 次观测，都具有唯一的生成词元 ID 序列和唯一的决策。这是对实际输出稳定性的检查，并非新的自然质量测量。所有提交的输出均有效率。没有任何完整生成达到其 1,024 词元上限。Native-eight 在全部 386 个提交序列中都达到预定的八 token 限制，而 Prefill-one 在全部 422 个序列中都达到预定的一 token 限制；适用时，这些序列包括干预、计时、预热和条件器阶段。这些计划内的短输出上限命中会予以保留，而不会被解读为完整输出故障。

两种引擎在同一主机、同一分配中依次运行，且单活跃序列引擎先运行。因此，引擎间的变化不能单独归因于并发：经过时间、缓存状态和引擎顺序仍然相互混杂。两种引擎都明确使用分块预填充和 32,768 的批处理 token 预算，这与先前的调度器默认设置不同。这项测定支持在此额外设置下，生成器内一个实用操作点具有稳健性；但并不能证明先前的百分比、生产延迟、质量等同性或领域整体最先进水平得到了精确复现。

完整执行、验证与成本。438 次生成调用全部完成，涵盖 1,194 条已提交序列：288 条干预序列、864 条自然时序序列、36 条条件器序列和 6 条预热序列。它们各自的完整服务总耗时分别为 70.483、250.602、1.448 和 7.264 秒。所有阶段的输入／输出词元总数为 3,059,522／17,633。活跃序列上限为 1／8 时，引擎初始化分别耗时 150.423／39.677 秒，关闭分别耗时 1.944／3.236 秒；实际 ID 诊断总计耗时 10.822 秒。资源验证和源文本分词预检分别耗时 5.776 和 5.381 秒。这些组件的计时已计入 590.746 秒的工作进程墙钟时间，而不是作为单独分配额外累加。主要分配是一块图形处理器，使用时长为 593 秒，按一次计入，其中包括设置、任何可能发生的故障以及空闲时间。独立的作业后 CPU ID 审计器在内部用时 23.283 秒验证了全部 1,194 条序列；其收集器进程耗时 24.897 秒。另一次仅 CPU 的提交前资格验证耗时 36.519 秒，期间未加载权重、未构建引擎，也未生成输出。本地评分耗费 0.333 个 CPU 秒，未产生新的模型调用，也未使用图形处理器。所有审计检查均通过，评分报告的问题数为零。暖服务比率不代表分配、计费、货币或节能估算。

最初的本地评分器在读取参考数据前就拒绝执行，因为其传输验证器将固定的远程分词器/配置路径解析为不存在的 Mac 资产。仅用于验证的固定版本则改用此前已收集的独立远程 CPU 哈希证明，验证这些确切资产；其他所有固定项仍要求提供本地字节数据。拒绝执行的状态和原始评分器均予以保留。此更改不影响推理、方法、来源、路由、指标或输出，也不会将任何模型/分词器资产复制到 Mac。已完成的评分和汇总表均对应这项严格受限的传输修复。

活跃序列臂	服务 生成	准备	输入令牌	输出词元	
1 完整	144.0244	142.7053	1.0859	471,930	6,024
1 原生八项	29.4013	28.1241	1.0847	471,930	1,152
1 Prefill-one	6.7898	5.5692	1.0492	472,938	144
8 完整	58.8752	57.7138	0.9865	471,930	6,024
8 原生八项	7.1114	5.9506	0.9983	471,930	1,152
8 Prefill-one	4.3995	3.2270	1.0063	472,938	144

表30：新鲁棒性设置下完整的自然输入运行时间总计。每行涵盖相同的24个固定响应，顺序为六种均衡顺序之一。每个提交序列的预填充输入都包含其七个语法令牌。准备和生成是服务的组成部分；同时也计入解析和最小写入时钟。这些是整批时钟，而非单个响应延迟。不会读取自然质量标签。

活跃顺序	P/F	P/N	N/F
1 FNP	.049383	.248561	.198676
1 FPN	.047463	.227374	.208744
1 NFP	.047096	.228481	.206128
1 NPF	.046446	.229299	.202556
1 PFN	.046860	.228397	.205169
1 PNF	.045666	.224303	.203591
8 FNP	.070717	.607288	.116448
8 FPN	.078201	.647498	.120774
8 NFP	.073836	.596061	.123874
8 NPF	.075123	.611470	.122856
8 PFN	.075919	.632036	.120118
8 PNF	.074802	.618622	.120917

表31：所有预先指定的顺序对比。F、N 和 P 分别表示完整、原生八项和 Prefill-one。比率采用该引擎/顺序在三个固定批次中的完整服务总和，而不是各批次比率的平均值。“活跃”表示该引擎的最大活跃序列数。

F 匹配契约与优化执行对照

另外两项对照用于区分原生读出的作用与其既有推理原语的作用。第一项保留验证器的语义指令，同时将其输出契约替换为直接给出布尔答案。第二项测试：当解码器采用默认编译执行路径时，将语法移入预填充阶段是否仍然有用。这些对照复用了已公开的开发数据或标签盲定时输入；它们并未新增独立的 Modern 测试队列。所有比较均使用与重点读出相同的冻结检查点和分词器版本。

F.1 语义匹配的二元契约

完整的 Data2txt 开发来源包含 1,026 个响应，分布于 171 个每组件包含六个响应的源组件中，其中有 715 个正向参考答案和 311 个负向参考标注。该来源此前已在开发过程中公开，并源自官方 TRAIN，因此，这一比较诊断的是固定的接口选择，而不是提供新的确认。每个选定的响应均予以保留。参考数据仅供评估器使用，不会传递给提示构建过程或解码过程。

幻觉片段的原始开头定义、每个类别和子类别的定义、证据、答案以及用户包装器，在各实验组之间均逐字节一致。我们将原生操作中引用并分类片段的要求替换为固定指令：确定是否存在任何此类片段。二元输出约定要求返回一个 JSON 对象，其中 has_annotated_error 的值设为 true 或 false。所提供的前缀在该键的冒号之后立即结束。我们比较原生 PREFILL1、不受限二元续写和掩码二元续写。每种方式均生成一个贪心令牌，不进行提示、阈值或随机种子搜索。与不受限二元生成相比，掩码提供了更强的格式控制：它将续写限制为所有合法的单词元布尔拼写，同时不改变验证定义。

合法字母表由语法而非预测质量决定。我们枚举所有非特殊词汇表词元，仅接受 true/false 周围的 ASCII JSON 空白字符（可选地带有对象闭合右花括号），并在给定前缀后验证组合解码。接受的六个 ID 为 804（空格–true）、867（空格–false）、1802（true）、3721（false）、34819（制表符–false）和 41393（制表符–true）。此词汇表中没有任何词元能将获准的布尔值与闭合右花括号组合起来。二进制前缀 ID 为 (4754，4497，62，3292，639，3960，763)；原生七个 ID 保持不变。二进制解析和掩码处理均使用完整的六 ID 字母表。

表 32报告了校正后的比较。原生 PREFILL1 有 677 个真正例、43 个假正例、

读出	Prec.	召回率	F1	有效率
原生 PREFILL1	94.03	94.69	94.36	100.00
二元，掩码	96.33	88.11	92.04	100.00
二元，自由	93.92	23.78	37.95	35.77

表 32：公开 DEV 上匹配定义的二元控制。所有数值均为百分比。自由二元有 659 个无效输出：在 F1值计算中，未知正例计为假阴性，而未知负例则明确保留，不会变成真阴性。因此，其 F1值同时附有有效覆盖率；完全有效的掩码二元组是固定默认契约控制。

38 个假阴性和 268 个真负例；掩码二元分别为 630、24、85 和 287。固定默认设置下，原生 F1值的差异为 2.32 个百分点，配对组件自助法的 95% 区间为 $[1.10, 3.59]$。我们对 171 个完整源组件进行重抽样，两个实验组共用相同的抽样结果，重复抽样 2,000 次，随机种子为 20261102。该区间以暴露队列和固定输出为条件，并非独立确认区间或等效性区间。在 70 个意见不一致的案例中，原生方法在 49 个案例中判断正确，而二元方法判断错误；反之则有 21 个案例。这一默认设置下的差异体现了精确率与召回率之间的权衡：掩码二元的精确率更高，而原生方法保留了更多正向参考标注。这并不意味着它在所有指标上都占优。

对于不受限二元，514 个正类样本和 145 个负类样本仍然未知。其余计数为 170 个真正例、11 个假正例、31 个有效假阴性和 155 个真阴性。严格全项准确率（将每个未知项都计为错误）为 31.68%；相应的最坏情况二元 F1 界限为 32.66%。我们既不修正这些输出，也不将其丢弃。不过，该组中的格式错误不能替代完全有效的掩码对比。

每个实验组按原始来源顺序处理 33 个批次，每批最多包含 32 个响应，三个实验组的顺序按批次索引轮换。测量前，每个实验组进行一次单例预热。BF16 急切执行的活动序列上限为 8，最大上下文为 16,384，不使用前缀缓存，贪心随机种子设为 0。共有 3,078 个测量序列，另加三次预热。全新服务包括全新的分词、同步生成、解析，以及每次告警时的最简同步写入。原生、掩码二元和自由二元的测量服务耗时分别为 14.379 秒、14.661 秒和 14.481 秒；其生成时间分别为 10.068 秒、10.090 秒和 10.018 秒。原生输入/输出总量为 1,471,501/1,026；两个二元实验组均为 1,462,267/1,026。所有提供的语法均计为输入。这些成本相当；约 2% 的服务耗时差异并非主要的效率结果。共享初始化耗时 53.100 秒，资格认定、预热和诊断分别保留。

此对比描述的是固定且语义匹配的契约下的一个公开 DEV 运行点。我们不据此推断其相较于校准二元读出的内在质量优越性，也不推断片段训练具有因果上的必要性。确切的提示词、引擎设置、令牌字母表、混淆计数、不确定性规范和阶段成本，均随本附录一并收录于匿名汇总制品中。

F.2 默认编译执行

运行时对照复用了 24 个此前筛选出的标签盲输入，分为三个固定批次，每批八个。每批均按 Native8–Prefill1 和 Prefill1–Native8 两种顺序测量。我们按固定顺序运行四个独立进程：活动序列上限为一的即时执行、上限为一的优化执行、上限为八的优化执行，以及上限为八的即时执行。每个进程包含 96 个测量序列和两次单例预热，总计 392 个序列。这是三个固定批次按重复顺序测量，并非六个独立样本，也不是新的质量总体。

所有进程均使用 BF16 算术、最大上下文长度和批处理令牌预算 32,768、启用分块预填充、禁用前缀缓存、0.70 的内存利用率，以及随机种子零。所请求的干预将 enforce_eager 从 true 改为 false。由此得到的默认优化执行使用 Inductor 后端、编译模式 3，以及完整和分段式 CUDA Graphs。两个活动序列上限对应的捕获尺寸集合分别为 $\{1, 2\}$ 和 $\{1, 2, 4, 8, 16\}$。除配置快照外，原始进程日志也证实了编译和图捕获：两个优化进程报告的编译耗时分别为 47.86 秒和 11.09 秒，捕获耗时分别为一秒和两秒。此比较测量的是默认优化路径；它并未从编译器、后端或缓存状态中单独分离出 CUDA Graphs 的因果效应。

表33 显示，优化后相较于原生提前停止，总体增量虽较小但仍为正：PREFILL1 在上限为 1 时将全新服务减少了 19.57%，在上限为 8 时减少了 8.84%。对应的

执行	S	原生 NATIVE8	预填充 PREFILL1	比例
即时执行	1	10.250	2.560	.250
优化 1		3.152	2.535	.804
即时执行	8	2.820	1.554	.551
优化 8		1.692	1.542	.912

表 33：相同计时源下的新鲜服务总计（单位：秒）。S 是活动序列上限，而不是并发客户端数量。比率是 PREFILL1 服务总耗时除以 NATIVE8 服务总耗时的结果，而不是六个配对批次/顺序比率的中位数。初始化和预热与这些总计分开，并计入完整的分配成本。

批次/顺序	$S=1$	$S=8$
1 / 原生优先	1.372	.774
1 / 预填充优先	.628	1.053
2 / 原生优先	.624	1.032
2 / 预填充优先	.691	.937
3 / 原生优先	.721	.839
3 / 预填充优先	.785	.903

表34：Pre- fill1/Native8 的每项优化后配对服务比率。保留大于 1 的数值：在批次/顺序测量中，批次上限为 1 时，Prefill1 有 6 项中的 1 项较慢；上限为 8 时，6 项中有 2 项较慢。重复顺序并非相互独立的统计样本。汇总产物还保留了所有急切执行比率。

中位数配对服务比率分别为 .706 和 .920；即时执行的中位数分别为 .247 和 .578。这些汇总数据回答的是不同问题：总量之比按批次的 Native8 时长加权，而中位数则对每个配对批次/顺序比率赋予相同权重。我们同时报告这两项，而不以更有利的汇总数据取而代之。

收益并非在所有测量中都一致（表34）。每个进程的两种方案各测量了 48 个序列。Native8 的输入/输出总量为 157,310/384，Prefill1 的总量为 157,646/48，逻辑词元总数均为 157,694。没有无效输出，也没有进程内分支不一致的情况。在即时执行和优化执行中，所有 192 项匹配的输出 ID 与决策比较结果均一致。这是基于计时源观察到的一致性，并非对有限精度恒等性的普遍主张。

初始化（包括适用情况下的编译或捕获）按所述进程顺序分别耗时 46.071、88.540、55.585 和 44.359 秒。预热、来源资质确认、诊断和初始化均计入完整分配而不是有选择地分配给有利的一方。新鲜服务计时包括分词、同步生成、解析和最少量的警报写入。它不衡量单请求响应延迟、并发客户端服务、完整的应用程序工作流或按比例计算的图形处理器计费。较小的优化增量也提醒我们，不应将即时执行带来的加速效果原封不动地迁移到另一种运行时配置中。

F.3 有效性溯源与完整核算

发现了两处实现缺陷；我们予以保留，而没有将其转化为表面上的研究收益。首先，CPU 预检断言错误地要求原生七 ID 前缀包含列表起始符。实际上，前缀以冒号结尾；下一个生成的词元才带有列表起始符和分支。在图形处理器推理之前，只修正了这项断言，原生前缀、提示和解析器均未更改。

其次，最初的二元解析器和掩码仅接受不带前导空格的 true/false 词元 ID。语法审计发现，解析器拒绝了 366 个合法的带前导空格布尔输出。这个不完整的字母表并不能作为检测失败的证据。我们保留了其原始输出，并用上述预先规划的完整六词元语法取代了其能力比较。修正后的三臂运行使用相同的语义提示、输入和原生读出；没有标签或性能排名会选择别名。所有修正后的输出，包括剩余的 659 个无限制格式失败，均予以保留。论文的匹配契约比较仅使用修正后的结果。

第一次分配包含已废弃的二元比较和全部四个运行时进程，消耗 577 GPU 秒；修正后的二元分配消耗 167 GPU 秒。两者均已终止，各自分配了一个 GPU。两者合计的 744 GPU 秒包括启动、资格认定、诊断、失败控制工作和空闲开销。因此，完整的项目探索包含 119 个已终止的分配和 88,215 GPU 秒（24.5042 GPU 小时），待处理预留为零，付费 API 支出为零。这些分配成本与服务比率分开计算，不归因于任何单独的方法。

单独维护的制品档案包含用于报告汇总数据的 interface_controls.csv，以及用于记录精确契约、令牌拼写、计划、指标、所有配对运行时比率和软件版本的 interface_controls.json，

其中还包括输入/参考/代码哈希和完整成本范围。两份制品均不包含单个输入、标识符、参考标签或预测。原始实例和完整执行证据均另行保存；聚合重构和源哈希不能替代它们进行独立重跑。

S	原生8（秒）	预填充1（秒）	节省率	较慢
1	250.121	183.263	26.73%	0/126
8	166.308	155.106	6.74%	3/126

表35：完整工作负载优化后的全新服务。时间总和是每个方案对全部 2,015 个输入执行两轮的总时间，并非单轮时间。节省率为 1 减去 Prefill1 与 Native8 服务总时间之比。较慢次数统计的是比值大于 1 的配对数据块/顺序观测。

G 完整工作负载上的优化执行

完整工作负载对照旨在检验：相较于原生提前停止所带来的额外节省率，在经过优化执行并涵盖全部 2,015 个代码代理输入后，是否仍然成立。该测试扩展了 24 个输入的计时试点；检查点、原生指令、七词元前缀和告警解析器均保持不变。主要比较指标是 PREFILL1 与 NATIVE8 的新鲜服务时间总和之比。此计时测量不涉及标签、基于输出的选择、提示搜索或重试。

G.1 工作负载与执行

原始输入字节按冻结顺序保留，并划分为 63 个数据块：62 个每块 32 条，另有一个 31 条。每个数据块均按 NATIVE8–PREFILL1 和 PREFILL1–NATIVE8 两种顺序处理，初始顺序按数据块索引交替。因此，在每个活跃序列上限下，每个实验组都会处理每条输入两次：共 4,030 条测量序列。每个进程在每个实验组中还会进行一次单例预热，因此包括预热在内共 8,062 条序列。两个进程在同一 H100 80GB 分配上顺序运行，先使用活跃序列上限 $S = 1$，再使用 $S = 8$。这些是调度器上限，并非独立客户端的数量。两次顺序重复使用相同输入，并非独立的质量样本。

两个进程均使用 vLLM 0.17.1、BF16、随机种子为零的贪心解码，上下文和批量词元限制均为 32,768，启用分块预填充，内存利用率为 0.70，并禁用前缀缓存。计时调用中没有掩码或对数概率请求。默认的优化执行采用 Inductor 编译和完整及分段 CUDA Graphs。进程日志确认已加载编译后的图，并完成混合预填充/解码和完整解码图捕获；这一结论并非仅依据配置标志得出。实验复用了编译缓存，因此并非冷缓存编译基准测试。图捕获集合为 $\{1, 2\}$，用于 $S = 1$，以及 $\{1, 2, 4, 8, 16\}$，用于$S = 8$。

每次测量的调用都会全新构造并对其输入进行词元化，执行同步生成，解析告警，并通过同步持久化以最小方式写入。它们的总和就是全新服务计时。资格认定、初始化、预热以及更详尽的实际词元 ID 诊断序列化均单独计时。第一个测量数据块仍包含在报告总计中。在两个工作负载轮次中，每个方案在每个上限下恰好包含 17,242,482 个测量逻辑词元：Native8 使用 17,210,242 个输入词元和 32,240 个输出词元；Prefill1 使用 17,238,452 个输入词元和 4,030 个输出词元。因此，提供的语法按输入计费，而不是视为自由内容或从总数中删除。

G.2 服务节省量和完整成本

表 35显示，在两个上限下，相较于原生 NATIVE8 均有正的总体增量。一次完整工作负载轮次对应的平均耗时，在 $S = 1$ 下分别为 125.060 秒和 91.632 秒，在 $S = 8$ 下分别为 83.154 秒和 77.553 秒。总量比率分别为 .733 和 .933；63 个配对数据块总量的中位数分别为 .733 和 .934。在 $S = 8$ 下，126 次单独的数据块/顺序测量中有三次 favor 原生 NATIVE8。将每个数据块中的两种顺序相加后，在任一上限下都没有数据块出现逆转。上述所有观察结果均予以保留；总体增量为正，并不意味着每次测得的调用都更快。

仅生成过程在 $S = 1$ 耗时 205.242 秒，而在 $S = 8$ 耗时 138.392 秒；在 $S = 8$ 则为 114.205 秒，而在 $S = 8$ 为 102.709 秒，再次均为两轮的总和。模型和引擎初始化（包括适用时的编译/捕获）在两个进程中分别耗时 66.469 秒和 65.307 秒。两次预热合计分别占用 2.186 秒和 .956 秒的服务时间。测得的诊断序列化在两个实验组中分别耗时 124.202 秒和 133.989 秒；源文本分词资格验证耗时 207.517 秒和

S	重复次数	原生NATIVE8 F1值	PREFILL1 F1值	Δ
1	0	83.642	83.519	-0.123
1	1	83.642	83.519	-0.123
8	0	83.475	83.748	$+0.273$
8	1	83.598	83.704	$+0.106$

表36：实际优化输出的描述性质量。F1值以百分比表示；Δ 为PREFILL1减去NATIVE8的百分点差。重复次数是平衡顺序索引，而不是全局的原生优先/预填充优先条件。四行均复用相同的参考数据。

238.709 秒。完整的工作进程墙钟时间分别为 866.515 秒和 820.853 秒。单次分配消耗 1,697 GPU 秒，计算方式为一个分配的 GPU 乘以终端主作业的经过时间，其中包括所有启动、检查、诊断、关闭和空闲开销。这些成本不同于常驻批处理服务比率。该实验未测量在线单请求延迟、并发客户端吞吐量、端到端应用节省或按比例计费的 GPU 计费。

G.3 实际决策与描述性质量

所有测得的输出均有效，但优化后的执行并非严格保持决策不变。Native8 和 Prefill1 在 $S = 1$ 的 4,030 组配对判定中有 12 组不同，在 $S = 8$ 则有 13 组不同。在 $S = 1$，两组实验的重复结果完全一致。在 $S = 8$，Native8 的两次重复在 2,015 个输入中有 6 个不同，Prefill1 则有 1 个不同。不同调度器上限下，Native8 在 4,030 组匹配判定中有 18 组不同，Prefill1 有 9 组不同；生成 ID 的差异数量也相同。这些记录不支持主张存在精确的有限精度轨迹等价性。

观察到这些差异后，我们记录并执行了一次描述性、仅限 CPU 的联接，将实际计时的告警与已公开发布的代码代理参考标注进行关联。我们没有拟合任何策略或阈值，没有运行新的生成任务，并保留了全部四种上限／重复次数条件。每种条件都包含相同的 2,015 个响应，来自 1,613 个源组件，其中有 1,014 个正向参考数据和 1,001 个负向参考标注。这项事后检查将质量与测得的速度数据点关联起来；它并非新的独立质量确认。

在两次 $S = 1$ 重复中，原生8的精确率/召回率为90.286/77.909，PREFILL1为90.263/77.712。在 $S = 8$时，第0次重复分别为90.160/77.712和90.400/78.008，第1次重复分别为90.183/77.909和90.297/78.008。预填充1每次重复分别纠正错误并引入错误：前者为2个，后者为4个 $S = 1$；在 $S = 8$第0次重复中，纠正4个、引入2个；第1次重复中，纠正4个、引入2个。表36 中的F1值变化幅度虽小，但并非为零。结合完整工作负载下服务成本的降低，这表明存在一个实用且因配置而异的质量—成本权衡点，而非无损解码或普遍的质量提升。完整的阶段总成本、混淆计数、每个上限下全部126组配对计时观测数据以及制品哈希值均已留存，以供审计。

H 工件溯源与完整探索成本

保存的证据清单将每个已完成的评分器输出与其 SHA-256 摘要、来源／协议标识以及实际分配成本绑定起来。它将前瞻性读出研究与后续的暴露队列对照和重放区分开来。原始执行轨迹仍保存在保存的实验档案中。汇总记录保留了失败成本、不良结果和修正沿袭；它们不能替代原始轨迹进行独立重跑。

项目账本记录了 121 次终端分配共计 90,659 GPU秒（25.1831 GPU 小时），其中包括放弃的方向、加载、失败以及空闲／CPU 工作。这其中包括一次 281 秒的资格认定／计时试点，该试点不属于标记的质量总体。六臂 DEV 测定、十一臂 Modern 确认、较早的物理级联以及证据／平衡运行时间测定分别贡献了 479、1,907、999 和 593 秒。后续的二进制／编译对照和布尔文法修复分别贡献了 577 和 167 秒；原始结果和修正结果均予以保留。完整的优化工作负载增加了 1,697 秒；本文之外的 747 秒辅助接口分析也仍计入其中。每次分配仅计费一次。完整账本另行保存为 exploration_costs.json。API 支出为零。总探索量不同于方法服务，不能据此确定服务提供商价格或能源节省量。