LEGOSpec：用于推测草稿设计与训练的可组合高效框架

Hanzhen Wang[1],Jiaming Xu[1,3],Guohao Dai[1],
[2], [3], *[1]上海交通大学 [2]Infinigence AI [3]SII
*通讯作者：daiguohao@sjtu.edu.cn

arXiv:2611.33004v1 [cs.LG] 24 Nov 2026

摘要——投机解码通过使用轻量级草稿模型加速大语言模型（LLM）的生成。现有方法在不同训练配置下评估完整方案，因此各组件的效果、交互作用及组合优势尚不明确。为此，我们提出 LEGOSpec，这是一个用于推测草稿设计与训练的可组合框架。LEGOSpec 将草稿模型拆解为上下文、骨干网络、提案和目标函数组件，并由统一的训练基础设施提供支持。借助 LEGOSpec，我们系统地改变各组件，并在匹配的训练条件下评估选定的组合，从而分离出它们的效果和交互作用。这些实验揭示了三点发现：（1）不同工作负载会因上下文长度和内存预算而偏好不同的骨干网络，其中块内依赖关系建模对长上下文尤为重要。（2）提案设计必须兼顾接受度与延迟。（3）训练目标函数应在局部匹配词元分布，并优化整个块的接受长度。这些发现为工作负载感知配置提供指导，并催生了*DFlux*——一种均衡的通用草稿模型，结合了配备卷积的 SWA–GQA 混合骨干网络、残差校正器和分层监督。在 Qwen 和 Gemma 模型上，与最强基线相比，*DFlux*在常规基准测试上的接受长度最高提升 13.7%，在长上下文工作负载上的接受长度最高提升 34.7%。为支持高效训练，LEGOSpec 集成了元数据感知调度和异步执行，使训练吞吐量最高提升 79%，并将离线监督准备速度提高至 SpecForge 的 3.11× 倍。

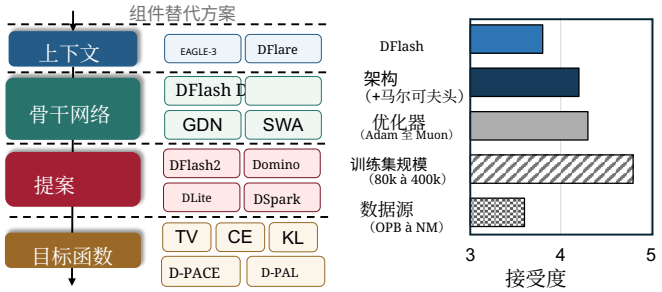
索引词—投机解码，训练框架

§ SJTU-DAI-Sys/LEGOSpec 🤗 SJTU-DAI-Sys/LEGOSpec

I. 引言

投机解码通过并行验证草稿令牌来加速自回归大语言模型 [1], [2]。现有方法改进了不同的草稿组件 [3]–[9]，但在不同训练配置下对完整方案进行比较，会掩盖组件的影响、相互作用及组合优势。图 1 展示了训练变化可以对接受度的影响不亚于架构变化。因此，需要进行受控比较，以区分各组件的影响与训练数据和优化方面的差异。

我们提出 **LEGOSpec**，这是一个用于推测草稿设计与训练的可组合框架。LEGOSpec 沿四个维度组织草稿方案：*Context* 选择并融合目标模型特征；*Backbone*处理历史上下文，并对草稿块内的依赖关系进行建模；*Proposal*生成候选分布，并可选择根据前置词元进行条件化；*Objective* 定义监督方式。借助 LEGOSpec，我们围绕共享基准系统地替换各个组件，并评估选定的组合



(a) 分散式优化 (b) 训练可能占据主导地位

图 1. (a) 上下文、骨干网络、提案和目标函数的不同选择。(b) 在 DFlash 中，更换优化器或将训练集规模从 80K 增至 400K，带来的提升可与添加马尔可夫头相当，甚至更高；而更换数据源则可能降低接受度。

在匹配的训练条件下，以隔离组件的影响及其交互作用（第三节）。这一共享的实验基础将组件发现与面向工作负载的设计以及通用的 *DFlux* 草稿方案联系起来。受控实验揭示了三项发现。**第一，不同工作负载偏好不同的骨干网络。** 全注意力 [10]适用于短上下文任务，而滑动窗口注意力（SWA） [11] 和卷积 [12] 则能提升长上下文接受率；紧凑历史状态可在内存受限的情况下增加上下文容量。**第二，提案设计应在保留全词表输出的同时，平衡接受度与延迟。第三，监督应在局部匹配词元分布，并针对整个草稿块优化接受长度。**

这些发现为**工作负载感知的草稿配置**提供了指导：密集全注意力的常规 接受度最高，而 SWA 在长上下文场景中表现最佳。在内存预算相同的情况下，SWA 和 MLA 等高效骨干网络在 Qwen3.5-9B 上支持的上下文容量可达到 MHA 的 5–7× 倍（第 IV 节）。

基于这些发现，我们还开发了*DFlux*，这是一种均衡的通用草稿模型，结合了配备卷积的 SWA–GQA 混合骨干网络、残差校正器和分层监督。在 Qwen [13] 和 Gemma [14]模型上，*DFlux* 相较于最强基线，在常规基准测试上的接受长度最高提升 13.7%，在长上下文工作负载上的接受长度最高提升 34.7%。

LEGOSpec 利用序列和锚点元数据进行前瞻调度与成本感知放置，从而减少目标计算浪费和草稿工作器失衡。结合异步执行（第 V 节），这些优化提升了

相较于 SpecForge [15]，训练吞吐量最高提升 79%，离线监督准备速度最高提升 3.11×。我们的贡献包括：(1) 一个可组合框架，以及在匹配条件下对各组件效果与交互作用的对照研究。(2) 由研究发现指导的工作负载感知配置，以及一个均衡的通用草稿模型。(3) 采用元数据感知调度和异步执行的高效训练基础设施。

II. 背景与相关工作

A. 推测解码设计

每轮输出一个接受的草稿前缀和一个额外令牌，并通过拒绝采样保持目标分布 [1], [2]。由此产生的解码加速约为

$$\text{Speedup} \approx \frac{\tau\, t_{\text{AR}}}{t_{\text{draft}} + t_{\text{verify}} + t_{\text{other}}}, \tag{1}$$

其中，τ 表示平均接受长度，包括每轮保证输出的令牌。此处，t_{AR} 表示每个令牌的平均自回归生成时间，分母则是每轮草拟、验证及其他开销之和。

EAGLE-3 [3] 融合目标特征以进行自回归草拟。DFlash [16] 采用块扩散，而 DFlare [6] 学习特定于层的特征融合。DSpark [17] 和 DLite [7] 分别通过顺序校正和循环头来建模块间依赖。DFlash2 [12] 将候选选择与块内卷积相结合。

B. 训练目标

DFlash [16] 使用位置衰减交叉熵损失，DART [18]使用 KL 损失进行蒸馏，而 DSpark [17]将词元监督与 TV 损失结合起来进行分布匹配。负对数接受率损失（NLA）是一种直接优化词元接受度的 LK 损失 [8],，而 D-PACE [19] 则根据接受长度代理量，为交叉熵损失推导自适应权重。D-PAL 损失 [20]利用分布重叠，为每个位置计算依赖于前缀的权重。

C. 草稿模型训练

草稿模型使用从目标模型获取的隐藏状态。令牌标签或目标分布为草稿预测提供监督。对于块式草稿生成，采样锚点词元定义预测块 [16]。在线训练在训练期间获取目标特征，而离线训练则预先计算并缓存这些特征。SpecForge [15] 将目标模型推理与草稿训练解耦，并使用 SGLang 进行特征提取。此工作流程包括目标模型预填充、监督信号传递和草稿模型更新。

III. 受控组件研究

我们围绕一个在 80K 样本上训练的共享 Qwen3-4B 参考配置，改变上下文、骨干网络、提案和目标函数：DFlash 上下文、五个密集 GQA 层、RNN 提议，以及混合损失（0.1 交叉熵 + 0.9 总变差）（表 II）。所有配置均在 $T = 1$ 下进行评估，*Regular* 按照第 VI-A 节对六个短上下文基准测试取平均值，*Long*则对六个 LongBench-v2 领域取平均值，输入长度为 32 K–64K 个令牌。附录 A 给出了目标函数定义和训练曲线。

表 I 不同训练集规模下的上下文融合。

上下文	80K		400K		1M	
	Regular	Long	Regular	Long	Regular	Long
DFlash	**4.472**	**2.450**	**4.902**	2.493	5.072	**2.512**
DFlare	4.391	1.623	4.829	**2.496**	**5.082**	2.428

A. 上下文

在使用 80K 训练样本时，DFlare 在*Regular* 和*Long*上均逊于 DFlash，在*Long* 上的接受长度低 33.8%（表 I）。由于这一结果与公开报告的结果 [6]，不同，我们进一步考察了数据扩展。在使用 400K 和 1M 训练样本时，DFlare 仍未表现出相对于 DFlash 的一致优势。因此，我们在后续比较中保留 DFlash 作为默认配置。

DFlare [6] 为每个草稿层 ℓ 提供由标量加权的目标层混合信息：

$$f_t^{(\ell)} = \text{RMSNorm}\left(\sum_{j=1}^{M} \text{softmax}(W_{\ell,:}^{\text{fuse}})_j h_t^{(j)}\right). \tag{2}$$

其中，$h_t^{(j)}$ 是位置 t处来自选定目标层 j 的隐藏状态，M 是选定层的数量。每个标量融合权重在通道之间共享，因此加权和会跨层合并对应通道，而不会进行经学习得到的跨通道混合。相比之下，DFlash 会投影串联后的目标特征，在注入之前混合特征通道。这一区别可能解释了两者的比较结果，并表明仅有特定层访问并不能替代通道混合。

B. 骨干网络

我们在可比参数预算下比较了五层骨干网络（表 VI）：多头注意力（MHA）、分组查询注意力（GQA）、滑动窗口注意力（SWA）、门控 DeltaNet（GDN）和多头潜在注意力（MLA）。历史访问和块内混合在草稿预测中发挥着不同作用。密集型 MHA 和 GQA 更有利于*Regular*接受度，而 SWA 和 GDN 在*Long*上表现更好（表 II）。与 GQA 相比，SWA 将*Long* 从 2.450 提升至 3.023，同时将*Regular* 从 4.472 降至 4.422。在 GQA 中加入卷积后，*Long* 提升了 20 %，达到 2.950，而*Regular*仅发生小幅变化。这些差异促使我们考察不断增长的历史信息如何影响固定大小的草稿块。

沿着 DFlash2 的注意力分析 [12], ，我们考察上下文增长如何影响草稿块内的注意力。在全注意力机制下，历史键集合会随上下文长度增加，而草稿块保持不变，导致参与竞争注意力权重总量的历史位置增多。这可能使整体注意力转向历史信息，并削弱块内信息交换。图 2(c) 支持这一解释：上下文长度从 2K 增至 32K 时，大多数全注意力查询头分配给草稿块内键的权重会减少，而 SWA-512 则几乎保持不变。SWA 通过限制可见历史来减少这种竞争。GDN 则采用另一种

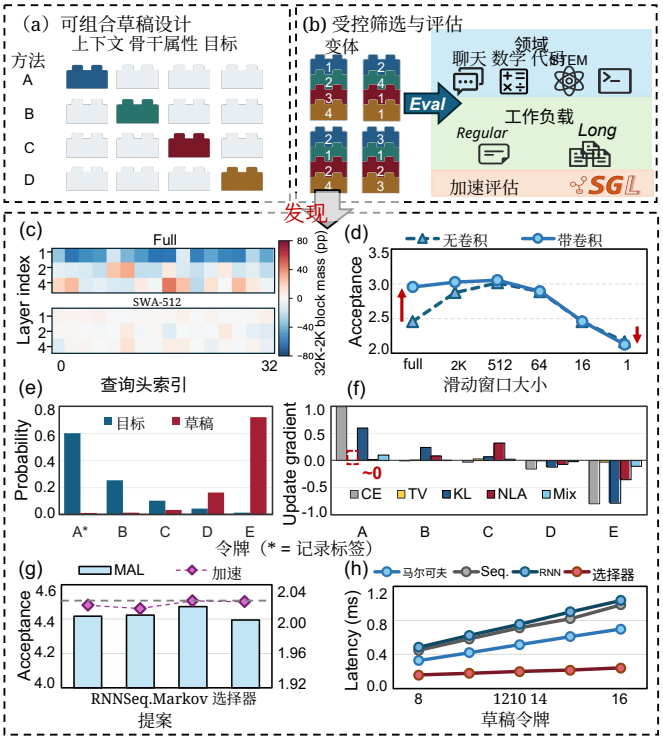


图2. 受控组件研究。(a)–(b) 草稿组件与匹配评估。(c) 上下文长度从 2K 到 32K 时的块内注意力权重总量。(d) 不同注意力窗口下的卷积效应。(e)–(f) 词元概率与负向 logit 梯度。(g) 提案接受率与加速。(h) 提案延迟与草稿长度的关系。

机制：将历史信息压缩到固定大小的循环状态中，而不是在不断增长的历史信息上扩展注意力。

卷积在每一层直接混合相邻的块词元，从而在不依赖注意力分配的情况下实现块内信息交换。如果这条路径能够弥补块间依赖的减弱，那么当 SWA 已能保留这些依赖时，其额外收益应会减小。窗口扫描结果符合这一预测：随着 SWA 窗口变窄，卷积带来的增益逐渐下降，并在窗口大小为 1 时变为负值（图2(d)）。这种交互效应支持了对二者长上下文优势的统一解释。

骨干网络在历史状态存储方面也有所不同（表 II）。全注意力缓存会随上下文长度增长；MLA 可减少每个词元的存储量，而 SWA 和 GDN 则保留有界的历史状态，使缓存大小减少 500×。在固定的系统内存预算下，较小的草稿状态可为上下文留出更多内存，这也促使我们在第 IV-B 节进行最大上下文分析。

要点1：不同工作负载适合不同的骨干网络。全注意力适合短上下文；SWA 保留块内注意力，而卷积则增强局部依赖。SWA、GDN 和 MLA 的紧凑状态还使其能够在内存限制下支持更长的上下文。

C. 提案

依赖校正提高了接受度，但也在草稿生成的关键路径上增加了工作量。循环神经网络按顺序

表 II $T = 1$处的逐组件接受率。缓存内存按上下文长度为 64K 个词元计算。

组	上下文	骨干网络	提案	目标函数	Regular	Long	缓存 (MiB)
Ref.	**DFlash**	**5×GQA**	**RNN**	**Mix**	**4.472**	**2.450**	**1280**
A	DFlare	—	—	—	4.391	1.623	—
B	—	5×GQA +卷积	—	—	***4.489***	***2.950***	1280
	—	5×MHA	—	—	4.459	2.391	5120
	—	5×SWA	—	—	4.422	**3.023**	10.00
	—	5×GDN	—	—	4.226	2.589	10.18
	—	5×MLA	—	—	4.436	2.384	360
C	—	—	并行马尔可夫	—	4.100	2.337	—
	—	—		—	4.416	2.388	—
	—	—	Sequential	—	4.408	2.389	—
	—	—	选择器*	CE	***4.294***	2.180	—
D	—	—	—	CE	4.193	2.290	—
	—	—	—	TV	4.131	2.320	—
	—	—	—	KL	4.422	2.436	—
	—	—	—	NLA	4.421	2.439	—
	—	—	—	D-PACE	4.242	2.074	—
	—	—	—	D-PAL	**4.524**	2.103	—

*选择器仅保留排名前 k 的逻辑值；我们遵循 DFlash2 使用交叉熵损失。

更新隐藏状态，以概括采样得到的令牌前缀；而马尔可夫方法仅根据紧邻的前一个令牌进行校正。Sequential 通过循环方式预测下一个分布。循环神经网络的接受度最高。（表 II）。然而，每次采样后处理循环前缀都需要额外计算。

相较之下，选择器会并行计算所有位置上相邻候选项的转移分数，而遍历和采样仍按顺序进行。延迟和端到端测量揭示了其中的权衡（图 2(g)–(h)）：选择器的提案延迟更低，尽管接受度较低，但其加速接近循环神经网络。因此，提案的贡献既取决于有多少令牌通过验证，也取决于它在验证开始前增加了多少耗时。

这些结果表明，有必要在保留并行转移评分的同时提升预测质量。残差校正器基于这一结构构建，以支持全词汇表分布校正和监督。结合分层监督后，它在我们的组件消融实验中取得了高于循环神经网络的接受度（第 VI-D 节）。

要点 2：在依赖校正中平衡接受度与延迟。提案设计应同时优化接受度和执行延迟。并行转移评分在保留前驱条件信息的同时，减少了串行提案计算。

D. 目标函数

词元级监督。 分布匹配使用完整词表上的目标概率来监督草稿，而非仅使用单个令牌标签。分布损失、NLA、KL 散度和混合损失能够提升*Regular* 接受度，效果优于仅标签交叉熵损失（表 II）。然而，尽管 TV 损失直接优化接受率，其表现仍不如交叉熵损失。

图 2(e)–(f) 展示了低重叠情况下的优化难点。对于目标概率较高但草稿概率非常

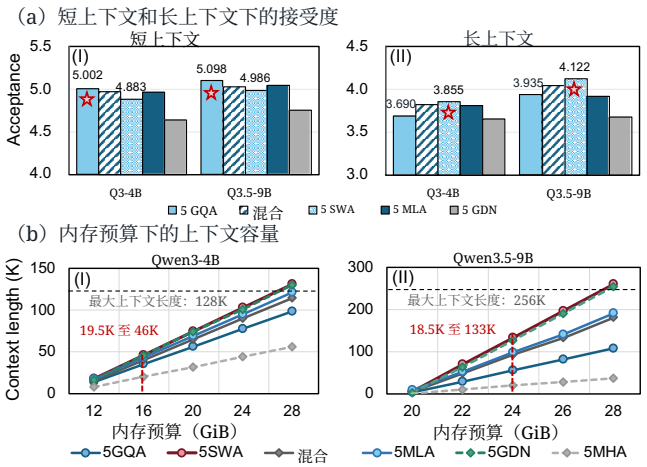


图3。（a）在卷积、RNN 提议和混合损失相匹配的情况下，不同骨干网络的接受度。（b）各内存预算下的最大上下文长度，包括权重和运行时开销。

低的词元，TV损失相对于草稿logits产生接近于零的梯度，从而减缓收敛速度。在混合损失（0.1 交叉熵+0.9TV）中，CE项在这一情况下增强了纠正梯度，而TV项则促进分布对齐。图示案例表明，混合损失在低重叠时结合了更强的梯度，并在收敛时实现了紧密对齐，这解释了其优于单独使用TV损失的原因。目标函数定义见附录A-B。

块级监督。只有当前一个前缀被接受时，后续令牌才会增加接受长度。D-PACE 和 D-PAL 损失通过自适应加权局部交叉熵和 KL 损失来考虑这种依赖关系。在*Regular*上，两者的表现均优于各自对应的局部损失，这支持了接受度感知的聚合方式。尽管 D-PAL 损失在*Long*上的表现不佳，但加入有利于长上下文草稿生成的卷积（第 VI-D 节）后，其在*Regular*和*Long*上的表现均优于混合损失。这表明互补组件可以协同作用，提升不同工作负载下的性能。

要点3：在局部匹配分布，并针对整个块优化接受长度。提议应保留全词表输出以进行分布监督，而最终目标函数在聚合局部损失时应考虑前缀接受情况。这两个层面促成了联合提议–目标设计和分层监督。

IV. 发现引导的工作负载感知设计

A. 上下文长度相关的骨干网络选择

使用 400K 个训练样本时，密集 GQA 在*Regular*下表现领先，而 SWA 在*Long* 下表现领先，且对两种目标模型均如此（图 3(a)）。在 Qwen3-4B 上，两者各自的优势分别为 2.44% 和 4.47%。这些结果表明，短上下文更适合完整历史访问，而长上下文草稿生成更适合有界历史 KV 访问；SWA–GQA 混合架构则兼顾了这两种需求。

B. 内存约束下的最大上下文长度

紧凑历史状态在相同的运行时内存总预算下可提升上下文容量（图3(b)）。将

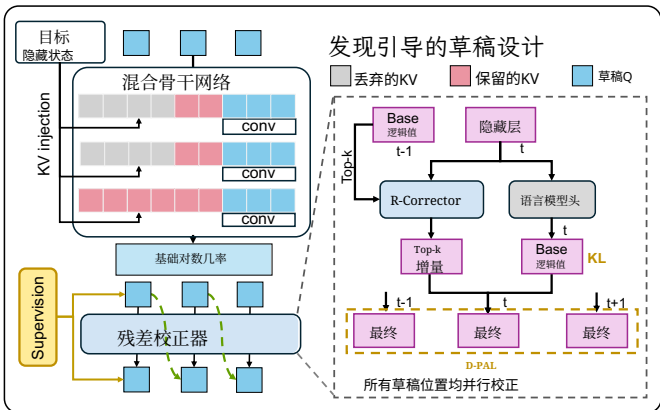


图 4. 我们的发现引导型通用草稿模型结合了混合骨干网络、残差校正器和分层监督。

五个密集 MHA 层替换为 SWA，可将 Qwen3-4B 在 16 GiB 内存下的上下文长度从 19.5K 词元扩展到 46K 词元，并将 Qwen3.5-9B 在 24 GiB 内存下的上下文长度从 18.5K 扩展到 133K（7.19×）。MLA 和混合方案可提供居中的容量，而 GDN 也支持显著更长的上下文。SWA 和 MLA 的接受长度与全注意力相当，而 GDN 带来的下降幅度更大。这些结果表明，紧凑型骨干网络能够在内存限制下支持更长上下文的工作负载，同时保持有竞争力的接受长度。

C. 均衡的通用草稿模型

我们将三项改进结合起来，构建通用型草稿*DFlux*，以平衡短上下文和长上下文的接受率以及历史状态存储。

SWA–GQA 混合骨干网络。更深的全注意力层对上下文长度不那么敏感。我们使用四个 SWA 层，后接一个带有块内卷积的密集 GQA 层。SWA 保留块内注意力，卷积混合相邻草稿位置，而最后一个 GQA 层则纳入完整历史信息。只有最后一层会保留随上下文增长的缓存。这种混合架构在保持*Regular* 性能的同时，提高了*Long* 接受率（图 3）。

残差校正器（R-Corrector）。为了在支持全词汇监督的同时保留选择器的并行性，我们提出了残差校正器。测量结果表明，前16个候选项占目标分布与草稿分布重叠部分的 99%。因此，我们将校正限制在这少数候选项上，并保留词汇表其余部分的骨干网络基准逻辑值，从而将校正工作量缩减至一个小型候选空间。

令 h_i 和 z_i^{base} 分别表示骨干网络隐藏状态和逻辑值，令 C_i 包含排名前 k 的候选项。对于前驱令牌 a 和候选项 v，校正项和最终逻辑值为

$$\Delta_i(a,v) = \left(E_a^- \odot W h_i\right)^\top E_v^+,$$
$$z_i^{\text{final}}(v \mid a) = z_i^{\text{base}}(v) + \mathbf{1}_{v \in C_i}\, \Delta_i(a,v), \tag{3}$$

其中，W 将隐藏状态映射到 r 维度，而$[E^-, E^+]$ 是前驱码本和后继码本。对于每个位置，对 $C_{i-1} \times C_i$ 中的所有配对进行评分，即可生成一个 $k \times k$校正表。所有位置的这些表都会并行计算

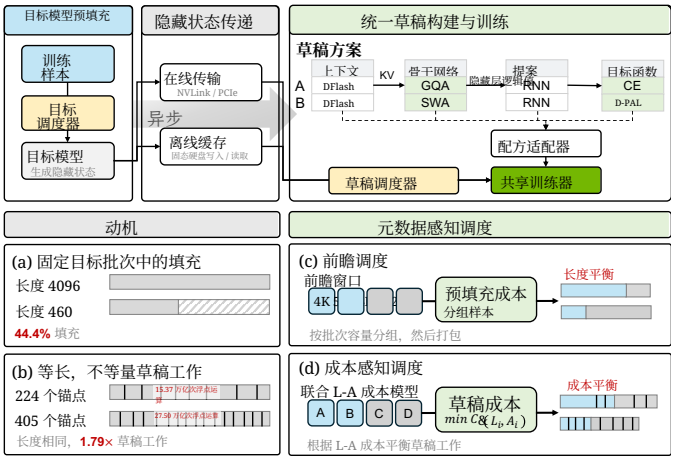


图 5. LEGOSpec 流水线（上）和元数据感知调度（下）。

。如果前驱令牌 a 不在候选集$a \notin C_{i-1}$中，我们就将校正项 $\Delta_i(a,\ v) = 0$ 设为零。通过拒绝采样，最终输出在理论上能够保留目标分布（附录 B）。

分层监督。 骨干网络输出旨在使目标词元出现在每个位置的前-k 候选集中，而最终输出旨在最大化接受长度。因此，我们分别对二者进行监督：辅助前向 KL 损失促使骨干网络分布覆盖合理的目标词元，而最终输出D-PAL损失则使用前缀依赖聚合来优化整个草稿块上的接受长度。两个分支均使用完整词表。总损失为

$$\mathcal{L}_{\text{train}} = \lambda_{\text{aux}}\,\mathcal{L}_{\text{KL}}(p, q^{\text{base}}) + \mathcal{L}_{\text{D-PAL}}(p, q^{\text{final}}), \qquad (4)$$

其中， p 是目标分布，辅助 KL 损失在有效草稿位置上取平均。我们将 $\lambda_{\text{aux}} = 0.2$ 设为以平衡损失尺度。

V. LEGOSPEC 框架设计

A. 模型组装与统一训练

LEGOSpec 通过配方适配器，将上下文、骨干网络、提案和目标函数连接到共享训练器。该适配器提供合法的锚点位置和监督布局，使组件能够替换，而无需更改训练循环。目标工作器会直接提供在线训练所需的隐藏状态，或将其存储起来用于离线训练。对于相同的目标模型和分词后的训练数据，所需目标层已缓存的配方可以复用监督信息。这样便将草稿设计与目标特征提取分离开来，并支持对不同组件组合进行受控比较。

B. 元数据感知调度

动机。现有框架会复用 SGLang [21] 和 vLLM [22] 等服务引擎进行目标模型预填充。训练每次只提交一个批次，因此基于当前请求队列进行调度会限制跨批次规划。然而，训练序列的长度和监督掩码都是已知的，因此可以进行前瞻分组和打包执行。草稿调度面临另一种负载不均衡：长度相同的样本可能有不同的锚点数量，因此计算成本也不同。图 5 显示了草稿成本的 1.79×差异，这是仅依据长度进行调度无法捕捉的。

目标侧前瞻调度。LEGOSpec 在前瞻窗口内按序列长度对即将处理的请求分组，从而减少填充，并均衡目标工作器的预填充工作。打包执行会处理有效词元，同时保留序列边界和位置。生成的隐藏状态会传输到草稿工作器以用于在线训练，或缓存起来用于离线训练。

草稿侧成本感知调度。LEGOSpec 将草稿成本模型与草稿前瞻相结合，对样本进行分组，并在草稿工作器之间分配工作。

草稿成本模型。规划器根据序列长度、锚点布局和监督掩码计算工作量。对于序列长度 L、 A 个锚点和块大小 B，查询数量为 $Q = AB$，全注意力包含 $P = B\sum_{j=1}^{A} t_j + AB^2$ 个可见 Q–K 对，其中 t_j 是锚点 j 的历史长度。主要的操作计数估算指标为

$$\begin{aligned} C_{\text{attn}} &\approx (L+Q)H^2 + PH, \\ C_{\text{FFN}} &\approx 3QHF, \qquad C_{\text{vocab}} \approx R_{\text{loss}}HV, \end{aligned} \qquad (5)$$

其中， H、 F 和 V 分别表示隐藏维度、FFN 维度和词汇表维度， R_{loss} 表示有效损失行的数量。对于 SWA，Q–K 对的计数基于窗口限制历史。对于 GDN，循环状态更新取代 Q–K 交互项。MLA 使用潜在维度估算投影成本。将各层成本与上下文、提案以及适用的教师 $\widehat{}$ 投影成本相加，得到样本权重 C_i。

对于目标批次 $\mathcal{B}$ 和 R 个草稿秩，样本分配会使估计秩成本的最大值最小化：

$$\begin{aligned} &\min_{\{\mathcal{S}_r\}_{r=1}^{R}} \quad \max_{1\le r\le R} \sum_{i\in\mathcal{S}_r} \widehat{C}_i \\ &\text{s.t.} \quad \biguplus_{r=1}^{R} \mathcal{S}_r = \mathcal{B}. \end{aligned} \qquad (6)$$

此处， $\mathcal{S}_r$ 包含分配给秩 r 的样本。LEGOSpec 使用确定性贪心算法：按成本对样本排序，将其分组成同步轮次，并将它们分配给负载最低且有空闲批次槽位的秩。

草稿前瞻。 在即将到来的更新中应用草稿成本模型，可为均衡分组和放置提供更多选择。LEGOSpec 保留窗口的样本出现记录和计划锚点，但可以跨更新重新分组样本。我们在框架效率实验中使用 16 次更新的前瞻窗口。

C. 异步监督流水线

LEGOSpec 将 CPU 批次准备和监督输入输出与 GPU 计算重叠。后台工作线程构建锚点布局、估算成本并组装即将到来的批次，同时草稿工作器在当前批次上训练。

目标工作器预填充下一批数据，同时将已完成的隐藏状态传输给草稿工作器进行在线训练，或写入存储以进行离线监督准备。在此期间

表 III 在短上下文和长上下文工作负载上的平均接受长度，取五个随机种子的平均值。

模型	方法	短上下文工作负载						LongBench-v2 领域			长代码编程
		Alpaca	MT-Bench	GSM8K	MATH-500	MBPP	LCB	MD-QA	L-ICL	SD-QA	SWE
Qwen3-4B	DFlash	3.23	3.75	5.63	5.32	4.77	3.97	2.14	2.52	2.50	1.58
	DFlash2	3.25	3.88	5.87	5.64	4.83	3.89	3.12	3.45	3.17	2.96
	DSpark	3.57	3.89	6.04	5.61	5.12	4.30	2.42	2.75	2.67	2.24
	DLite	3.56	3.88	6.12	5.98	5.14	4.04	2.74	3.11	2.98	3.03
	DFlux	**4.03**	**4.39**	**6.47**	**6.35**	**5.74**	**4.95**	**3.60**	**3.81**	**3.63**	**4.08**
Qwen3.6-35B-A3B	DFlash	3.26	3.82	5.77	5.82	5.66	4.55	3.25	3.65	3.09	3.15
	DFlash2	3.45	4.00	6.06	5.99	5.88	4.66	3.48	4.05	3.34	3.62
	DSpark	3.39	3.75	5.66	5.85	4.97	4.11	3.14	3.76	3.17	2.84
	DFlux	**4.13**	**4.59**	**6.89**	**6.54**	**6.24**	**5.09**	**3.60**	**4.16**	**3.58**	**3.98**
Gemma4-12B-it	DFlash	3.36	3.69	5.94	5.54	4.80	4.14	2.72	2.78	2.65	3.39
	DFlash2	3.72	4.05	6.33	6.01	5.28	4.65	2.60	2.81	2.58	3.44
	DSpark	3.60	3.97	6.21	5.92	5.19	4.50	3.00	3.05	3.01	4.00
	DFlux	**4.12**	**4.43**	**6.65**	**6.42**	**5.77**	**5.14**	**4.02**	**4.09**	**3.85**	**4.52**

进行离线训练时，草稿工作器预取缓存的隐藏状态并准备下一批数据，同时 GPU 计算继续进行。有界队列和可复用缓冲区限制了未完成的传输和写入操作。这种异步监督流水线将准备、传输和存储延迟隐藏在训练执行之后。

VI. 评估

A. 实验设置

训练设置与基线。我们使用 140 万个样本训练*DFlux*：其中 90 万个样本取自 Open-PerfectBlend [23], 300万个样本取自 Nemotron-Post-Training-Dataset-v2 [24], 万个样本取自 WildChat [25]。我们仅保留问题，并在 $T = 1$ 生成回复，以构建训练数据。我们使用 16 张 NVIDIA H800 GPU 训练 6 个轮次，全局批大小为 128。Adam [26] 和 Muon [27] 的学习率分别为 6×10^{-4} 和 0.01。我们使用原论文发布或由 deepseek-ai 和 RedHatAI 提供的基线检查点（附录 C）。由于没有公开检查点，我们自行训练了 Gemma4-12B-it 的 DFlash2。

评估协议。除 Qwen3-4B DFlash2 检查点沿用其原生块大小 9 外，所有模型的块大小均设为 8。接受度评估使用 $T = 1$、最大生成长度 1,024 个词元以及拒绝采样。我们使用不同的随机种子重复进行每项接受度基准测试五次，并报告平均接受长度，以降低采样波动。

基准测试。短上下文工作负载包括聊天：Alpaca [28] 和 MT-Bench [29]；数学：GSM8K [30] 和 MATH-500 [31]；代码：MBPP [32] 和 LiveCodeBench v6 (LCB) [33]。长上下文工作负载包括 LongBench-v2 [34]，中的单文档问答（SD-QA）、多文档问答（MD-QA）和长上下文学习（L-ICL）领域，以及基于真实仓库级软件问题的 SWE-bench [35] 提示。我们从两个长上下文基准测试中选取 32K 至 64K 词元的输入。

B. 与现有方法的比较

我们在 Qwen3-4B、Qwen3.6-35B-A3B 和 Gemma4-12B-it 上将*DFlux* 与 DFlash [16],DFlash2 [12], DSpark [17], 和 DLite [7]进行比较。*DFlux* 在所有评估的工作负载和目标模型上均实现了最高的接受长度（表 III），相较于最强基线，在 SWE-bench 上最高提升 34.7%，在 GSM8K 上提升 13.7%。在聊天、数学、代码和长上下文工作负载上的这些提升，证明了组件改进的有效性和泛化性。

C. 匹配的训练条件下的比较

为单独评估草稿设计带来的收益，我们使用相同的 80 K Open-PerfectBlend 样本和共享的训练超参数，对 DFlash、DFlash2、DSpark 和 *DFlux* 进行训练。*DFlux* 在两个目标模型的每项基准测试上都实现了最高的接受度（图 6），表明其收益在匹配的训练条件下依然存在。各数据集的详细接受长度见附录 D 的表 VII。

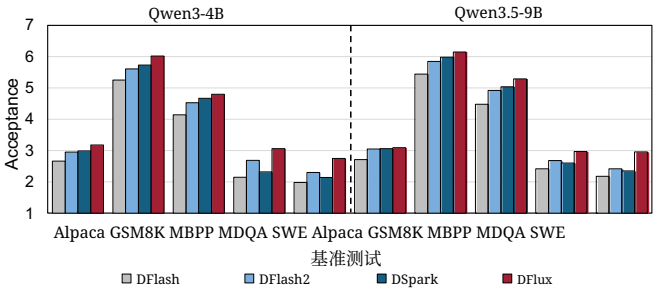


图 6. 在 Qwen3-4B 和 Qwen3.5-9B 上使用相同训练数据和共享训练超参数时，τ 在 $T = 1$ 下的平均接受长度。

D. 组件消融

表 IV 评估了 Qwen3-4B 和 Qwen3.5-9B 上的骨干网络、提案和监督改进。

骨干网络。 在固定循环神经网络和 Mixloss 的情况下，混合架构在 Qwen3-4B 和 Qwen3.5-9B 上的 Long 接受度分别比密集 GQA 高 24.5% 和 14.6%，Regular 也略有提升。SWA

表 IV 使用 DFlash 上下文的组件消融。

模型	骨干网络	提案	目标函数	Regular	Long
Q3-4B	GQA	RNN	Mix	4.472	2.450
	~~混合~~	RNN	Mix	4.480	3.051
	混合	~~选择器~~	CE	4.392	2.927
	混合	R-Corrector	Mix	4.535	3.045
	混合	R-Corrector	D-PAL	4.599	3.109
	混合	R-Corrector	~~分层~~	**4.618**	**3.123**
Q3.5-9B	GQA	RNN	Mix	4.548	2.649
	~~混合~~	RNN	Mix	4.560	3.036
	混合	~~选择器~~	CE	4.422	2.875
	混合	R-Corrector	Mix	4.578	3.032
	混合	R-Corrector	D-PAL	4.666	3.110
	混合	R-Corrector	~~分层~~	**4.687**	**3.121**

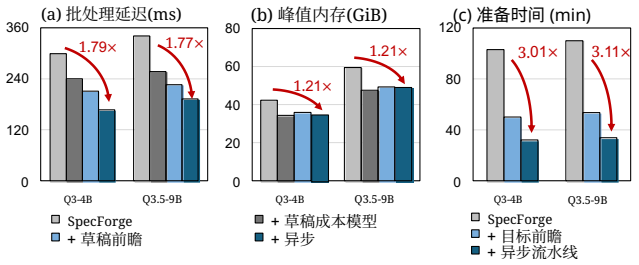


图 7. 框架效率。(a)–(b) 累积优化下的批处理延迟和峰值内存。(c) 离线监督准备时间。

限制了历史KV竞争，而块内卷积增强了块内依赖关系。

提案。采用混合损失后，R-Corrector 在几乎不改变 *Regular*接受度的情况下，提高了相较于循环神经网络的接受度。采用混合损失的 R-Corrector 在两个工作负载上也均优于采用交叉熵损失的选择器，同时支持全词汇表分布式监督，并保留并行转移评分。

目标函数。分层监督在两种模型和所有工作负载上都取得了最高接受度，优于混合和 D-PAL 损失。辅助 KL损失有助于提升骨干模型候选覆盖，而最终输出 D-PAL损失则优化接受长度；二者结合能够改进最终草稿。

E. 高并发吞吐量

我们使用 SGLang 高效实现*DFlux*，并比较其在 $T = 0$ 下的吞吐量（表 V）。平均 τ 表示平均接受长度。在 Qwen3.6-35B-A3B 和 Gemma4-12B- it 上，*DFlux* 在三个任务的每种批大小下都实现了最高吞吐量，相较自回归取得 1.57–5.62× 倍加速，吞吐量也比最强的推测基线高 4.7–24.2%。更长的接受长度反映出预测准确率有所提升，而 R-Corrector 中的并行转移评分减少了顺序提议工作。二者共同作用，使*DFlux*既能降低批大小为 1 时的解码延迟，也能提升较大批大小下的吞吐量。

F. 框架效率

a) 草稿训练： **LEGOSpec** 相比 SpecForge 实现了 1.77–1.79×倍加速，同时峰值内存降低 17.1–17.6%（图 7(a)–(b)）。草稿成本模型带来 1.25–1.33×

表 V 使用 SGLang 后端的吞吐量和接受度。

Task	方法	批大小（每秒事务数 / 加速） 1	8	32	64	平均值 τ
Qwen3.6-35B-A3B						
Alpaca	AR	131/1.00×	1033/1.00×	3740/1.00×	6113/1.00×	—
	DFlash2	298/2.27×	2001/1.94×	5817/1.56×	7820/1.28×	3.37
	DSpark	303/2.30×	2005/1.94×	5137/1.37×	7193/1.18×	3.26
	DFlux	**353/2.69×**	**2336/2.26×**	**7125/1.91×**	**9712/1.59×**	**3.85**
GSM8K	自回归	132/1.00×	1033/1.00×	3740/1.00×	5988/1.00×	—
	DFlash2	517/3.92×	3497/3.38×	10127/2.71×	13281/2.22×	5.96
	DSpark	504/3.82×	3349/3.24×	8552/2.29×	11611/1.94×	5.52
	DFlux	**602/4.57×**	**3994/3.86×**	**12058/3.22×**	**16013/2.67×**	**6.66**
MBPP	AR	132/1.00×	1033/1.00×	3565/1.00×	5677/1.00×	—
	DFlash2	500/3.80×	3374/3.27×	9255/2.60×	11938/2.10×	5.73
	DSpark	440/3.34×	2916/2.82×	7118/2.00×	9443/1.66×	4.77
	DFlux	**532/4.04×**	**3533/3.42×**	**10136/2.84×**	**13120/2.31×**	**5.88**
Gemma4-12B-it						
Alpaca	AR	47/1.00×	381/1.00×	1480/1.00×	3168/1.00×	—
	DFlash2	109/2.32×	943/2.48×	3195/2.16×	4122/1.30×	3.02
	DSpark	123/2.61×	1011/2.65×	3408/2.30×	4393/1.39×	3.41
	DFlux	**137/2.91×**	**1151/3.02×**	**3833/2.59×**	**4963/1.57×**	**3.61**
GSM8K	自回归	47/1.00×	382/1.00×	1492/1.00×	3253/1.00×	—
	DFlash2	212/4.50×	1838/4.81×	6394/4.29×	8399/2.58×	5.72
	DSpark	236/5.02×	1950/5.10×	6741/4.52×	8856/2.72×	6.29
	DFlux	**256/5.42×**	**2149/5.62×**	**7362/4.93×**	**9719/2.99×**	**6.56**
MBPP	AR	47/1.00×	378/1.00×	1465/1.00×	3114/1.00×	—
	DFlash2	160/3.43×	1379/3.65×	4614/3.15×	5876/1.89×	4.51
	DSpark	190/4.06×	1555/4.12×	5159/3.52×	6561/2.11×	5.25
	DFlux	**207/4.44×**	**1739/4.60×**	**5689/3.88×**	**7288/2.34×**	**5.55**

倍加速，平衡了草稿工作器计算；草稿前瞻增加 1.14×，而异步流水线通过重叠调度、传输与训练增加 1.17–1.26×。

b) 离线监督准备： 目标侧的前瞻头调度与异步缓存写入加速了 400K 个样本的准备工作，在 Qwen3-4B 上提升 3.01×，在 Qwen3.5-9B 上提升3.11× （图 7(c)）。分组可减少填充并平衡预填充工作，而异步写入则将存储输入输出与目标计算重叠执行。

VII. 结论

本文提出了**LEGOSpec**，这是一个用于推测草稿模型设计与训练的可组合、高效框架。借助**LEGOSpec**，我们开展了受控组件比较和组合研究，以识别各组件的影响及其相互作用。这些发现为工作负载感知的骨干网络选择和通用 *DFlux* 方案提供了指导。*DFlux*在SWE-bench和GSM8K上的接受度较最强基线分别最高提升34.7%和13.7%。其SGLang集成版本在所有测试批大小下，均在Qwen3.6-35B-A3B和Gemma4-12B-it上实现了最高吞吐量。元数据感知调度和异步执行使草稿训练速度较SpecForge最高提升 1.79×，离线监督准备速度最高提升3.11×。

致谢

本研究由国家自然科学基金（编号：62622406、62674144、62561160156）和上海市启明星计划（编号：24QB2706200）资助。

参考文献

[1]Y. Leviathan、M. Kalman 和 Y. Matias，“通过投机解码实现 Transformer 的快速推理”，载于《第 40 届国际机器学习会议论文集》，《机器学习研究论文集》系列，第 202 卷。PMLR，2023 年，第 19 274–19 286 页。[在线]。网址：https://proceedings.mlr.press/v202/leviathan23a.html [2] C. Chen、S. Borgeaud、G. Irving、J.-B. Lespiau、L. Sifre 和 J. Jumper，“通过投机采样加速大语言模型解码”，arXiv 预印本 arXiv:2302.01318，2023 年。[3] Y. Li、F. Wei、C. Zhang 和 H. Zhang，“Eagle-3：通过训练时测试扩展大语言模型的推理加速”，《神经信息处理系统进展》，第 38 卷，136 756，2026 年，136 737 页。[4] H. Wang、J. Xu、Y. Xiang、J. Pan、Y. Zhou、Y.-L. Li 和 G. Dai，“Specprune-vla：通过具备动作感知能力的自投机剪枝加速视觉-语言-动作模型”，arXiv 预印本 arXiv:2509.05614，2025 年。[5] J. Xu、J. Pan、H. Wang、Y. Zhou、J. Ye、Y. Wang 和 G. Dai，“Specontext：利用大语言模型中的投机式上下文稀疏性实现高效长上下文推理”，载于《第 31 届 ACM 编程语言与操作系统架构支持国际会议论文集》第 2 卷，2026 年，第 1832–1847 页。[6] J. Zhang、Z. Yu、S. Liu、E. J. Yu、Z. Li、D. Zhu、J. Duo、W. Xiong、Y. Song、G. Yu 等，“DFlare：扩展块扩散投机解码的草稿容量”，arXiv 预印本 arXiv:2606.02091，2026 年。[7] J. Xu、H. Wang、T. Zhao、J. Xiao、C. Yuan、J. Pan、Y. Zhou、J. Ye、Y. Wang 和 G. Dai，“DLite：面向智能体人工智能的无历史投机解码”，arXiv 预印本 arXiv:2610.23787，2026 年。[8] A. Samarin、S. Krutikov、A. Shevtsov、S. Skvortsov、F. Fisin 和 A. Golubev，“Lk损失：直接优化投机解码的接受率”，arXiv 预印本 arXiv:2602.23881，2026 年。[9] Y. Zhou、K. Lyu、A. S. Rawat、A. K. Menon、A. Rostamizadeh、S. Kumar、J.-F. Kagy 和 R. Agarwal，“Distillspec：通过知识蒸馏改进投机解码”，载于《国际学习表征会议》，第 2024 卷，2024 年，第 32 011–32 050 页。[10] A. Vaswani、N. Shazeer、N. Parmar、J. Uszkoreit、L. Jones、A. N. Gomez、Ł. Kaiser 和 I. Polosukhin，“Attention Is All You Need”，《神经信息处理系统进展》，第 30 卷，2017 年。[11] I. Beltagy、M. E. Peters 和 A. Cohan，“Longformer：长文档 Transformer”，arXiv 预印本 arXiv:2004.05150，2020 年。[12] Inco AI，“DFlash 2：持续并行起草”，2026 年。 [在线]。网址：https://inco.ai/blog/dflash2/ [13] A. Yang、A. Li、B. Yang、B. Zhang、B. Hui、B. Zheng、B. Yu、C. Gao、C. Huang、C. Lv 等，“Qwen3 技术报告”，arXiv 预印本 arXiv:2505.09388，2025 年。[14] G. Team、S. E. Abd、V. Aggarwal、R. Algayres、A. Andreev、O. Bachem、I. Ballantyne、C. Brick、V. C˘arbune、M. Casbon 等，“Gemma 4 技术报告”，arXiv 预印本 arXiv:2607.02770，2026 年。[15] S. Li、C. Wang、Y. Zhu、Y. Wang、F. Yin、S. Shi、Y. Chen、X. Dong、Q. Chen、J. Pan 等，“Specforge：面向投机解码的灵活高效开源训练框架”，arXiv 预印本 arXiv:2603.18567，2026 年。[16] J. Chen、Y. Liang 和 Z. Liu，“DFlash：用于闪速投机解码的块扩散”，arXiv 预印本 arXiv:2602.06036，2026 年。[17] X. Cheng、X. Yu、C. Shao、J. Li、Y. Xiong、Y. Qian、J. Zhu、S. Ma、X. Zhang、J. Ye 等，“DSpark：采用半自回归生成的置信度调度投机解码”，arXiv 预印本 arXiv:2607.05147，2026 年。[18] F. Liu、X. Li、K. Zhao、Y. Gao、Z. Zhou、Z. Zhang、Z. Wang、W. Dou、S. Zhong 和 C. Tian，“DART：用于快速大语言模型推理的扩散启发式投机解码”，arXiv 预印本 arXiv:2601.19278，2026 年。[19] T. Wu、Y. Yao、Z. Qi、H. Zheng、Z. Wang、H. Ma、L. Liao、H. Lakkaraju、J. Li 和 Y. Du，“D-PACE：用于并行投机起草的动态位置感知交叉熵”，arXiv 预印本 arXiv:2605.18810，2026 年。[20] T. Wu、Z. Wang、J. Geuter、Y. Yao、W. Jiang、C. Zhang、J. Li 和 M. Yu，“超越点质量：用于投机解码的基于接受率的权重与分布损失”，arXiv 预印本 arXiv:2610.28355，2026 年。[21] L. Zheng、L. Yin、Z. Xie、C. Sun、J. Huang、C. H. Yu、S. Cao、C. Kozyrakis、I. Stoica、J. E. Gonzalez 等，“SGLang：结构化语言模型程序的高效执行”，《神经信息处理系统进展》，第 37 卷，第 62 557–62 583 页，2024 年。[22] W. Kwon、Z. Li、S. Zhuang、Y. Sheng、L. Zheng、C. H. Yu、J. Gonzalez、H. Zhang 和 I. Stoica，“利用 PagedAttention 实现大语言模型服务的高效内存管理”，载于《第 29 届操作系统原理研讨会论文集》，2023 年，第 611–626 页。[23] T. Xu、E. Helenowski、K. A. Sankararaman、D. Jin、K. Peng、E. Han、S. Nie、C. Zhu、H. Zhang、W. Zhou 等，“完美融合：通过评审者混合重新定义基于人类反馈的强化学习”，arXiv 预印本 arXiv:2409.20370，2024 年。[24] D. Nathawani、S. Ding、V. Lavrukhin、I. Gitman、S. Majumdar、E. Bakhturina、B. Ginsburg 和 J. Polak Scowcroft，“Nemotron-Post-Training-Dataset-v2”，2025 年 8 月。 [在线]。可访问：https://huggingface.co/datasets/nvidia/Nemotron-Post-Training-Dataset-v2 [25] W. Zhao、X. Ren、J. Hessel、C. Cardie、Y. Choi 和 Y. Deng，“WildChat：真实环境中的 100 万条 ChatGPT 交互记录”，载于《第十二届学习表征国际会议》，2024 年。 [在线]。可访问：https://openreview.net/forum?id=Bl8u7ZRlbM [26] D. P. Kingma 和 J. Ba，“Adam：一种随机优化方法”，2017 [在线]。1412 6980 。在线。可访问：https://arxiv.org/abs/ 。[27] J. Liu、J. Su、X. Yao、Z. Jiang、G. Lai、Y. Du、Y. Qin、W. Xu、E. Lu、J. Yan 等，“Muon 可扩展用于大语言模型训练”，arXiv 预印本 arXiv:2502.16982，2025 年。[28] R. Taori、I. Gulrajani、T. Zhang、Y. Dubois、X. Li、C. Guestrin、P. Liang 和 T. B. Hashimoto，“Stanford Alpaca：指令跟随型 Llama 模型 ”，https://github.com/tatsu-lab/stanford_alpaca，2023 年。[29] L. Zheng、W.-L. Chiang、Y. Sheng、S. Zhuang、Z. Wu、Y. Zhuang、Z. Lin、Z. Li、D. Li、E. Xing 等，“使用 MT-Bench 和 Chatbot Arena 评判大语言模型评审者”，《神经信息处理系统进展》，第 36 卷，第 46 595–46 623 页，2023 年。[30] K. Cobbe、V. Kosaraju、M. Bavarian、M. Chen、H. Jun、L. Kaiser、M. Plappert、J. Tworek、J. Hilton、R. Nakano 等，“训练验证器以解决数学应用题”，arXiv 预印本 arXiv:2110.14168，2021 年。[31] Hugging Face H4，“MATH-500”，Hugging Face 数据集。 [在线]。可访问：https://huggingface.co/datasets/HuggingFaceH4 500 /MATH-J. [32] Austin、A. Odena、M. Nye、M. Bosma、H. Michalewski、D. Dohan、E. Jiang、C. Cai、M. Terry、Q. Le 等，“利用大语言模型进行程序合成”，arXiv 预印本 arXiv:2108.07732，2021 年。[33] N. Jain、A. Gu、W.-D. Li、F. Yan、T. Zhang、S. Wang、A. Solar-Lezama、K. Sen 和 I. Stoica，“LiveCodeBench：面向代码的大语言模型整体且无污染评估”，载于《学习表征国际会议》，第 2025 卷，2025 年，第 58 791–58 831 页。[34] Y. Bai、S. Tu、J. Zhang、H. Peng、X. Wang、X. Lv、S. Cao、J. Xu、L. Hou、Y. Dong 等，“LongBench v2：迈向对现实长上下文多任务的深入理解与推理”，载于《计算语言学协会第 63 届年会论文集》（第 1 卷：长篇论文），2025 年，第 3639–3664 页。[35] C. E. Jimenez、J. Yang、A. Wettig、S. Yao、K. Pei、O. Press 和 K. Narasimhan，“SWE-bench：语言模型能否解决现实世界的 GitHub 问题？ ”，载于《学习表征国际会议》，第 2024 卷，2024 年，第 54 107–54 157 页。

附录 A 受控组件研究详情

A. 骨干网络配置

表 VI 列出了受控组件研究中使用的骨干网络配置。目标模型为 Qwen3-4B。所有变体均使用五层、隐藏维度 2560 、 FFN 中间维度 9728 、 DFlash 上下文，以及秩为 256 的 RNN 提议。参数量涵盖一个骨干网络层，包括注意力、FFN 和层归一化

表 VI 参数预算相当时的骨干网络配置。

骨干网络	结构配置	参数量/层
MHA	$H_q = H_{kv} = 32,\ d_h = 128$	116.66M
GQA	$H_q = 32,\ H_{kv} = 8,\ d_h = 128$	100.93M
SWA	带有历史窗口 $W = 512$ 的 GQA	100.93M
GDN	$H_k = 16,\ H_v = 32;$ $d_k = d_v = 128$	116.87M
MLA	$H_q = H_{kv} = 32,\ r_{kv} = 512;$ $d_{\text{nope}} = 128,\ d_{\text{rope}} = 64,\ d_v = 128$	106.60M

H_q， H_{kv}， H_k， H_v 表示头数； d_h， d_k， d_v 表示头维度。对于 MLA， r_{kv} 表示潜在维度，而 d_{nope}， d_{rope} 分别表示非 RoPE 和旋转位置编码查询/键维度。

B. 令牌级目标

令 $p_i(v)$ 和 $q_i(v)$ 分别表示有效草稿位置 i 处词汇表 $\mathcal{V}$ 上的目标概率和草稿概率，并令 y_i 为记录标签。令牌级损失为

$$\ell_i^{\text{CE}} = -\log q_i(y_i), \tag{7}$$

$$\ell_i^{\text{KL}} = \sum_{v\in\mathcal{V}} p_i(v)\log\frac{p_i(v)}{q_i(v)}, \tag{8}$$

$$\ell_i^{\text{TV}} = \frac{1}{2}\sum_{v\in\mathcal{V}} |p_i(v) - q_i(v)|, \tag{9}$$

$$\ell_i^{\text{NLA}} = -\log \sum_{v\in\mathcal{V}} \min\{p_i(v), q_i(v)\}, \tag{10}$$

$$\ell_i^{\text{Mix}} = 0.1\ell_i^{\text{CE}} + 0.9\ell_i^{\text{TV}}. \tag{11}$$

NLA 损失对分布重叠取负对数，遵循直接接受损失 [8]。重叠度等于 1 减去总变差损失。交叉熵损失使用单个标签，而 KL、总变差和 NLA 损失需要目标概率。Mixloss 将标签信号与分布匹配相结合。在参考方案中，使用固定的位置衰减权重聚合局部损失；无效的监督位置会被排除。

C. 接受感知块聚合

对于包含 m 个受监督草稿位置的完整区块，将局部目标–草稿重叠写作 a_i。只有当某个位置之前的前缀通过验证时，该位置才会对接受长度作出贡献。D-PACE [19] 和 D-PAL [20] 损失使用动态位置信用来计入这一依赖关系，而不是仅应用固定的位置权重。

前缀与后缀信用构造可写为

$$u_j = \prod_{k=1}^{j} s_k, \qquad c_i = \sum_{j=i}^{m} u_j, \tag{12}$$

$$\mathcal{L}_{\text{block}} = \sum_{i=1}^{m} \text{sg}(c_i)\,\ell_i, \tag{13}$$

其中， s_k 是平滑的局部接受代理量，sg 表示停止对信用值的梯度传播。D-PACE 损失使用记录标签的草稿概率作为代理量，并在局部使用交叉熵损失。D-PAL 损失使用全词汇表重叠作为代理量；正文讨论的变体在局部使用 KL 损失。在前缀乘积之前，平滑操作将局部代理量与 1 进行插值。由此得到的信用值同时包含到达某个位置的可能性，以及该位置对后续接受位置的贡献。这将局部监督的选择与块级损失聚合区分开来。

D. 训练曲线

图 8 展示了骨干网络、提案和目标函数对比实验在训练过程中的预期接受长度。在 MLA 骨干网络训练期间，我们发现，使用默认的 Muon 和 Adam 学习率（分别为 0.01 和 6e-4）会导致性能下降，这可能是因为 MLA 未采用逐头归一化。将两个学习率都降低一个数量级后，MLA 的性能恢复到与其他骨干网络相当的水平。相比之下，其他骨干网络变体和组件未表现出类似的学习率敏感性，因此我们保留了它们的默认学习率。

附录 B 推测采样正确性

固定一个前缀，并将目标条件分布和提议条件分布分别简记为 p 和 q。提案 $Y \sim q$ 以概率 $\min\{1, p(Y)/q(Y)\}$ 被接受。定义

$$\alpha = \sum_{v\in\mathcal{V}} \min\{p(v), q(v)\}, \tag{14}$$

$$r(v) = \frac{[p(v) - q(v)]_+}{1-\alpha}, \tag{15}$$

其中 $[x]_+ = \max\{x, 0\}$ ，且仅当拒绝概率为正时才使用 r 。输出任意令牌 v 的概率为

$$\Pr(X = v \mid \mathcal{H}_i) = \min\{p(v), q(v)\} + (1-\alpha)r(v) \tag{16}$$

$$= p(v). \tag{17}$$

当 $\alpha = 1$ 时，无需进行拒绝替换。在每个已接受前缀处依次应用这一恒等式，并在整个区块均被接受后采样额外的目标词元，即可恢复目标序列分布。该论证允许使用依赖前驱的校正和零校正回退，因为二者都包含在 q 中。它假设进行精确概率评估，并采用标准的残差替换规则。在 $T = 0$ 处，贪心验证会在首次不匹配的位置输出目标参数最大值，并采用一致的并列处理策略。

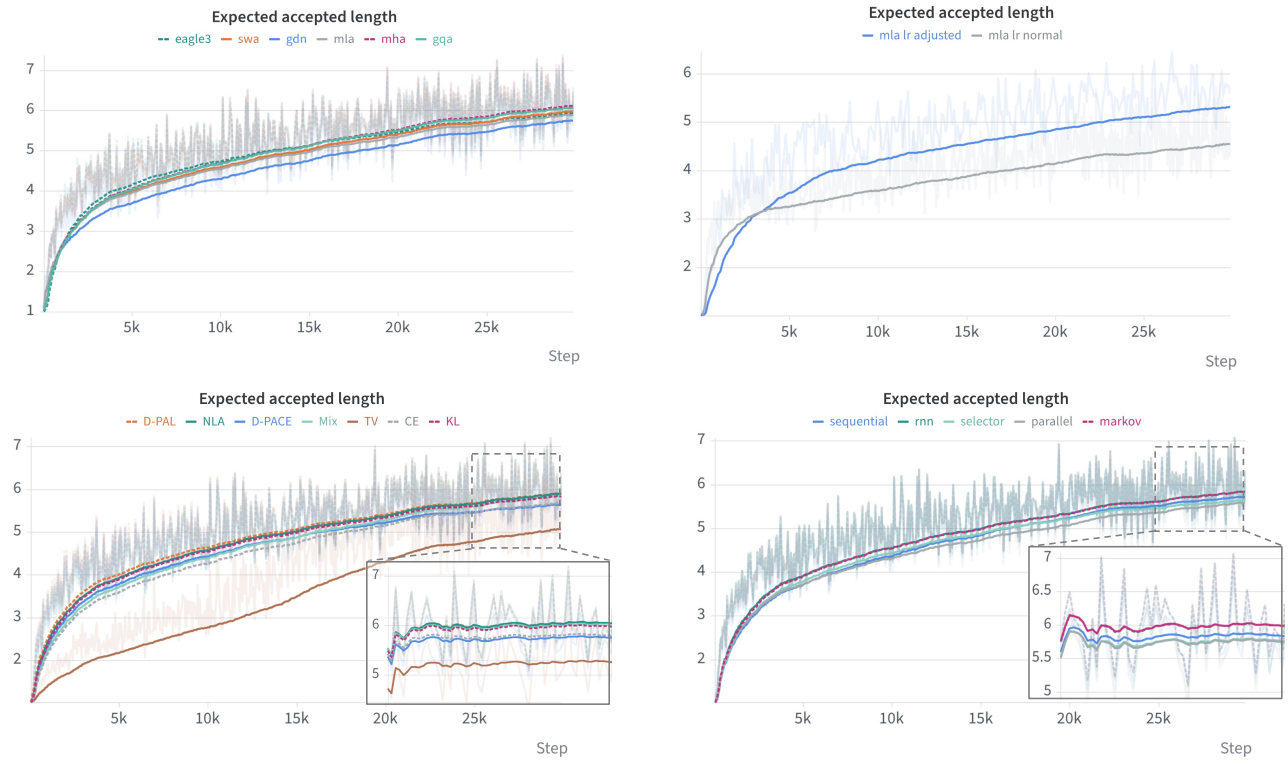


图 8. 组件研究中不同变体的预期接受长度训练曲线。

附录 C 基线检查点来源 表 VIII 列出了基线检查点的代码仓库；Gemma4-12B-it DFlash2 检查点在本工作中训练得到。附录 D 匹配的训练条件下的比较 表 VII 提供了匹配的 80K 训练条件下的详细接受结果。

表 VII 接受长度 τ 在 $T = 1$ 下（使用匹配的 80K 训练数据）。

		短上下文工作负载						LongBench-v2 领域			长代码编程
模型	方法	Alpaca	MT-Bench	GSM8K	MATH-500	MBPP	LCB	MDQA	LICL	DHU	SWE-bench
Qwen3-4B	DFlash	2.66	3.25	5.25	4.90	4.14	3.45	2.15	2.12	2.50	1.98
	DFlash2	2.95	3.55	5.61	5.30	4.53	3.81	2.69	2.73	3.06	2.30
	DSpark	2.99	3.63	5.73	5.45	4.67	4.01	2.32	2.35	2.68	2.15
	DFlux	**3.18**	**3.77**	**6.02**	**5.71**	**4.80**	**4.11**	**3.06**	**3.14**	**3.24**	**2.75**
Qwen3.5-9B	DFlash	2.71	3.08	5.44	5.04	4.48	3.34	2.42	2.37	2.87	2.18
	DFlash2	3.05	3.48	5.85	5.42	4.92	3.75	2.68	2.63	3.34	2.42
	DSpark	3.06	3.51	5.98	5.58	5.04	3.86	2.60	2.58	3.19	2.35
	DFlux	**3.09**	**3.65**	**6.15**	**5.80**	**5.29**	**4.01**	**2.96**	**2.89**	**3.60**	**2.96**

表 VIII 基线草稿的 Hugging Face 代码仓库。Gemma4-12B-it DFlash2 基线在本工作中训练。

方法	Hugging Face 代码仓库
Qwen3-4B	
DFlash	deepseek-ai/dflash_qwen3_4b_block7
DFlash2	RedHatAI/Qwen3-4B-speculator.dflash2
DSpark	deepseek-ai/DSpark_qwen3_4b_块7
DLite	SJTU-DAI-Sys/Qwen3-4B-DLite
Qwen3.6-35B-A3B	
DFlash	z-lab/Qwen3.6-35B-A3B-DFlash
DFlash2	incoai/Qwen3.6-35B-A3B-DFlash2
DSpark	RedHatAI/Qwen3.6-35B-A3B-speculator.dspark
Gemma4-12B-it	
DFlash	deepseek-ai/dflash_gemma4_12b_块7
DFlash2	aaaalex111/gemma4_12b_it_DFlash2
DSpark	deepseek-ai/DSpark_Gemma4_12b_块7