递归循环 Transformer

Recurrent Looped Transformer

Yifan Zhang

yifanzhangresearch@gmail.com

2026 年 9 月 12 日

摘要

我们提出递归循环 Transformer（Recurrent Looped Transformer，RLT），其设计围绕三项原则展开：具有无界时间深度的潜在推理、面向高效执行的模型与硬件协同设计，以及面向一致策略优化的模型与强化学习算法协同设计。因果编码器构建键值记忆，递归解码器则在每一个提示词与响应 token 之间传递最终隐藏状态和逐层滑动窗口注意力（SWA）缓存。随着序列延长，这形成了一条没有固定架构深度上限的潜在计算路径，同时每个 token 执行的模块数量保持固定。硬件协同设计将并行编码器计算与递归解码器计算分离，支持跨序列批处理、记忆复用和激活检查点训练。算法协同设计使预训练、监督微调、轨迹采样和当前策略回放采用相同的状态转移，从策略定义中消除提示词边界造成的差异。精确回放在当前参数下重建状态，而非复用过时的采样状态。因此，RLT 为硬件感知的递归执行与可靠的 RL 扩展提供了具体基础。无限深度指可持续延展的时间路径，而非单个 token 内的无限计算；实际推理收益、吞吐量和 RL 扩展能力仍有待验证。

项目主页：https://github.com/yifanzhang-pro/recurrent-looped-tranformer

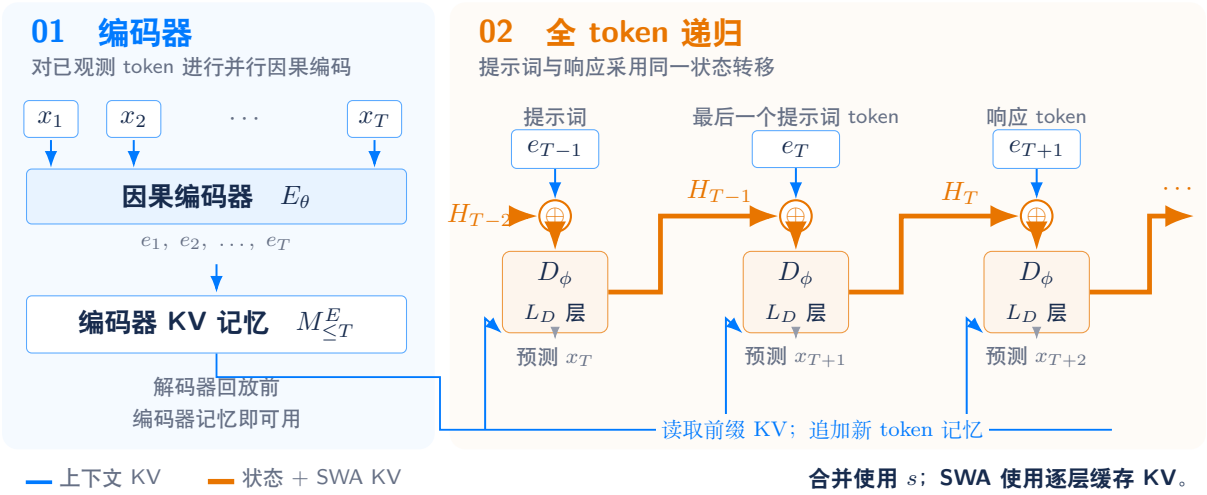

图 1: **贯穿提示词与响应的统一递归计算**。编码器构建因果 KV 记忆；解码器按顺序处理每一个 token。橙色箭头将完整解码器状态 $H_t = (s_t, C_t^D)$ 跨越提示词与响应边界传递，不做重置；其中只有 s_t 进入合并操作，而各 SWA 层读取自身缓存的 KV。蓝色箭头提供受前缀限制的编码器记忆，并随新 token 到来而扩展。图中省略了更早的提示词状态转移。

1 引言

递归循环 Transformer（RLT）围绕三个目标设计：*具有无限深度的潜在推理、面向高效训练与推理的模型与硬件协同设计，以及面向训练与采样一致性的模型与 RL 算法协同设计*。这里的无限深度意味着：随着处理的 token 增多，时间维度上的计算没有固定的架构上限。任意有限序列执行的计算仍然是有限的。

因果编码器构建上下文记忆，深层递归解码器将每个 token 的编码器表示与前一个解码器输出合并。状态在已观测提示词和生成响应中持续演化，不在两者的边界处重新启动。对于 48 层编码器和 48 层解码器，每个 token 都将 h_{t-1}^{96} 反馈到从 h_t^{48} 开始的计算中。两个堆栈可以共享兼容的权重，从而在同一骨干网络上形成两次逻辑遍历，每个 token 执行 96 次逻辑模块计算。

具有无限深度的潜在推理。 递归状态直接在 token 之间传递连续的中间计算。处理 t 个 token 后，一条状态路径穿过 tL_D 个解码器模块。其深度不受已存储层数的限制：继续处理 token 即可延长路径。这支持贯穿序列的潜在推理，同时保持每个 token 的模块数量固定。架构提供了这条路径，而如何沿着它学到有效推理是另一个问题。

模型与硬件协同设计。 源自编码器的记忆将并行特征构建与递归状态更新解耦。因此，可以采用大批量编码器内核、跨序列独立解码器转移的批处理、常驻权重和 KV 的复用，以及使用激活检查点的反向计算。这些是架构提供的硬件优化机会。完整提示词递归仍是模型的一部分，因此追求硬件效率时，不能悄然跳过其中的顺序计算。

模型与 RL 算法协同设计。 预训练、监督微调、采样和 RL 回放均由同一个依赖历史的状态转移定义。当前策略状态在当前参数下由完整历史重建，而行为对数概率始终对应生成动作的采样器。这消除了提示词与响应采用不同计算所引入的结构性失配，并为可靠的 RL 扩展建立一致的策略评估基础。数值差异、常规策略滞后和 RL 目标本身的质量仍是独立问题。

YOCO 等编码器记忆架构启发了记忆复用设计 [Sun et al., 2024]，而 Feedback Transformer 和 Recurrent Transformer 为时间反馈提供了重要先例 [Fan et al., 2020, Oncescu et al., 2026]。本文关注完整解码器递归，以及对硬件执行与 RL 回放语义的联合规定。本报告阐述这些机制，不报告实测效率或扩展结果。

2 方法

2.1 序列与状态

设 $x_{1:S}$ 为一个独立序列，且 $x_1 = \text{BOS}$。推理时，$x_{1:T}$ 是已观测提示词，之后的 token 构成续写。索引 T 标记服务边界，不表示条件模型发生改变。令 L_E, L_D 分别为编码器和解码器深度，d 为残差流宽度。本文使用列向量。编码器表示为 $e_t \in \mathbb{R}^d$，递归输出为 $s_t \in \mathbb{R}^d$。完整解码器状态为 $H_t = (s_t, C_t^D)$，其中 C_t^D 包含各解码器 SWA 层保留的键值投影。窗口大小 $W \geq 1$

包含当前 token。每次更新后，每层缓存至多保留 $W-1$ 个位置，供下一次更新使用。所有参数（包括可学习初始状态 $s_\star$）统记为 Θ；E_θ 与 D_ϕ 分别表示编码器与解码器部分。

2.2 因果编码器与记忆

对于已观测前缀，计算

$$e_{1:T} = E_\theta(x_{1:T}). \tag{1}$$

借助因果掩码，可以同时处理同一编码器层中的各个位置。编码器层之间仍按顺序计算。对解码器记忆组 $g \in \{1, \dots, G\}$，构建

$$k_t^g = \mathcal{P}_K^g(e_t, t), \qquad v_t^g = W_V^g \,\mathrm{RMSNorm}_E(e_t), \qquad M_{\le t}^g = \{(k_j^g, v_j^g)\}_{j=1}^t. \tag{2}$$

键映射包含归一化、投影和所采用的位置变换。解码器第 ℓ 层读取记忆组 $g(\ell)$；$G=1$ 表示跨层共享记忆，$G = L_D$ 则允许逐层使用不同投影。该全局记忆依赖编码器表示，而非解码器状态。参考解码器将针对该记忆的交叉注意力与针对解码器激活的因果 SWA 结合。两种存储承担不同角色：$M_{\le t}$ 提供全局编码器上下文，C_t^D 提供有界窗口内源自解码器的 KV。

2.3 每个 token 使用同一状态转移

在 BOS 之前仅初始化一次：

$$H_0 = (s_\star, \varnothing). \tag{3}$$

对每一个已观测或采样的 token，执行

$$u_t = \mathrm{Merge}(e_t, s_{t-1}), \tag{4}$$

$$H_t = (s_t, C_t^D) = D_\phi(u_t; M_{\le t}, C_{t-1}^D, t), \qquad t \ge 1, \tag{5}$$

$$p_\Theta(x_{t+1} \mid x_{1:t}) = \mathrm{softmax}\big(W_o \,\mathrm{RMSNorm}_o(s_t)\big)_{x_{t+1}}. \tag{6}$$

对 $H = (s, C^D)$，定义 $F_t(H) = D_\phi(\mathrm{Merge}(e_t, s); M_{\le t}, C^D, t)$。对于固定 token 序列，编码器特征不直接依赖解码器状态。然而，用于预测首个响应 token 的完整状态包含全部提示词转移：

$$H_T = F_T \circ F_{T-1} \circ \cdots \circ F_1(H_0). \tag{7}$$

生成从 H_T 继续。从 s_T 采样 x_{T+1} 后，对其进行增量编码，追加源自编码器的 KV，并计算 $H_{T+1} = F_{T+1}(H_T)$，包括所有解码器 SWA 缓存更新。服务边界处不会重置 H_T 的任一分量。

2.4 提示词预填充与增量解码

预填充（prefill）首先计算提示词的因果编码器表示与记忆，然后依次计算 $H_1, \dots, H_T$。即使完整提示词记忆已经可用，解码器位置 t 的注意力仍限制在 $M_{\le t}$。生成阶段使用编码器缓存 C^E 执行

$$(e_t, C_t^E) = E_\theta^{\mathrm{step}}(x_t, C_{t-1}^E) \tag{8}$$

随后追加记忆，并执行同一解码器转移。已观测 token 可以分块编码，但必须保留因果编码器缓存，且每个 token 仍须执行其解码器更新。并行编码器处理与逐步递归编码是同一因果计算的不同执行调度；对于非线性解码器递归，本文不假设存在对应的常规完全并行调度。特别地，历史解码器 KV 必须由此前的递归更新生成；标准的并行 SWA 解码器遍历通常并不等价。

2.5 合并操作与解码器模块

一种具体的门控合并形式为

$$r_{t-1} = \mathrm{RMSNorm}_s(s_{t-1}), \tag{9}$$

$$g_t = \sigma\big(W_g[e_t; r_{t-1}] + b_g\big), \tag{10}$$

$$u_t = e_t + \alpha\, g_t \odot W_s r_{t-1}. \tag{11}$$

其中 $W_g \in \mathbb{R}^{d\times 2d}$，$W_s \in \mathbb{R}^{d\times d}$，$\alpha$ 控制反馈尺度。较小的非零初始反馈尺度是一种候选初始化方式，并非已经确立的稳定性方案。使用标量门控或低秩 W_s 可以降低开销。

令 $z_t^0 = u_t$。一种具体的解码器模块依次执行因果 SWA、编码器记忆交叉注意力和 FFN：

$$q_t^{D,\ell} = \mathcal{P}_Q^{D,\ell}(z_t^{\ell-1}, t), \tag{12}$$

$$k_t^{D,\ell} = \mathcal{P}_K^{D,\ell}(z_t^{\ell-1}, t), \qquad v_t^{D,\ell} = W_V^{D,\ell}\,\mathrm{RMSNorm}_{S,\ell}(z_t^{\ell-1}), \tag{13}$$

$$b_t^\ell = z_t^{\ell-1} + \mathrm{Attn}_\ell^D\Big(q_t^{D,\ell}, \{(k_j^{D,\ell}, v_j^{D,\ell})\}_{j=\max(1,t-W+1)}^{t}\Big), \tag{14}$$

$$a_t^\ell = b_t^\ell + \mathrm{Attn}_\ell^M\Big(\mathcal{P}_Q^{M,\ell}(b_t^\ell, t), M_{\le t}^{g(\ell)}\Big), \tag{15}$$

$$z_t^\ell = a_t^\ell + \mathrm{FFN}_\ell(\mathrm{RMSNorm}_{D,\ell}(a_t^\ell)), \qquad s_t = z_t^{L_D}. \tag{16}$$

注意力算子包含输出投影；查询与键映射包含各自的归一化和位置变换。在每一层，当前 KV 都在 SWA 之前由该层输入构建，因此对当前位置的注意力不会引入循环依赖。历史解码器 KV 来自 C_{t-1}^D。更新后，在 C_t^D 中保留位置 $\max(1, t-W+2), \ldots, t$；当 $W = 1$ 时，该集合为空。预填充、采样和回放使用相同的位置元数据约定。其他子层顺序定义不同变体，必须在所有执行模式下保持一致。

2.6 何处形成循环：一种具体的权重绑定配置

参考权重绑定 RLT 取 $L_E = L_D = L$。编码器第 ℓ 层自注意力与解码器第 ℓ 层 SWA 共享兼容的查询、键、值和输出投影矩阵；其 FFN 也共享。编码器使用自身的因果上下文，解码器 SWA 则使用窗口内的解码器激活。解码器交叉注意力具有独立的查询与输出投影，以及式 (2) 中的记忆投影。各阶段专属的归一化、合并和读出仍是显式模块。因此，解码器模块在复用的注意力与 FFN 核心上增加交叉注意力；两次逻辑遍历并不意味着每个模块的 FLOPs 相等。

这是在不同注意力连接方式下复用参数，而非复制激活。任何解码器输出都不被等同于编码器输出。不绑定权重的 E_θ, D_ϕ 保留完整状态递归，但取消沿深度的参数复用。编码器记忆组共享是另一个独立维度，并不会合并或消除逐层解码器 SWA 缓存。

3 计算性质

3.1 提示词与响应的一致性

命题 1 (对服务分割位置的不变性)**.** 固定参数、*token* 序列、位置约定和独立序列初始状态。假设因果编码器执行在数学上等价，*SWA* 窗口与缓存更新相同，且解码器操作确定。对任意前缀，先执行批量编码器预填充再执行递归解码器更新，与对该前缀逐 *token* 增量处理，将产生相同状态与下一 *token* 分布。对固定 *token* 历史，移动提示词与响应的分割位置不会改变条件分布。

证明. 因果编码器的等价性保证两种调度得到相同的 e_t 与 $M_{\le t}$。二者均以 $H_0 = (s_\star, \varnothing)$ 初始化。若它们在 $t-1$ 的状态相同，则式 (5) 在 t 对相同输入执行相同操作，故状态仍相同。由归纳法可得结论。服务分割位置从未出现在转移公式中。 □

这是一种数学等价性。不同内核和精度选择仍可能引起数值差异。此外，该结论假设分词与上下文一致，且任一执行过程中都没有丢弃 token、重置状态或插入过时状态。

3.2 预填充计算量与顺序深度

令 $C_E^{\mathrm{pf}}(T)$ 为编码器预填充计算量，$C_M(T)$ 为记忆投影计算量，$C_D^{\mathrm{step}}(t)$ 为一次解码器计算量：读取 t 条编码器记忆、每层至多 W 个解码器位置，并包含合并开销。则

$$C_{\mathrm{prefill}}(T) = C_E^{\mathrm{pf}}(T) + C_M(T) + \sum_{t=1}^{T} C_D^{\mathrm{step}}(t). \tag{17}$$

对于稠密注意力以及与模型宽度成比例的 KV 宽度，粗略算术复杂度为

$$O\big((L_E + L_D)(Td^2 + T^2 d) + GTd^2 + L_D T \min(W, T)d\big). \tag{18}$$

解码器存在一条经过 T 次转移的顺序路径，每次转移包含 L_D 个模块。因此，编码器并行不意味着整个模型预填充完全并行。即使算术复杂度阶数与传统稠密 Transformer 相同，状态递归仍可能降低硬件利用率。我们接受这一成本以保留完整提示词计算；本文不声称通过减少预填充计算获得加速。

每个新 token 执行 $L_E + L_D$ 个模块，此外还有合并与记忆投影。注意力计算量仍随上下文长度增长。设有效 KV 宽度为 d_{KV}，推理缓存存储量约为

$$O\big((L_E + G)t\, d_{\mathrm{KV}} + L_D \min(t, W-1) d_{\mathrm{KV}}^D + d\big). \tag{19}$$

其中 d_{KV}^D 是解码器 SWA 的有效 KV 宽度；$O(d)$ 项对应当前递归输出。SWA 项统计保留的历史，不包含临时的当前 KV。训练激活另计。参数绑定减少权重存储，但不会消除第二次逻辑遍历，也不会消除任何一种缓存的作用。

3.3 具有无限时间深度的潜在推理

到位置 t 的可用状态路径复合了 t 次解码器转移，从序列起点穿过 tL_D 个解码器模块。对于长度为 T 的提示词和随后 n 个已处理续写 token，相应路径包含 $(T+n)L_D$ 个解码器模块。

图 2 展示了这些解析计数。该深度包含提示词，并随已处理历史增长，而每个 token 的模块数量保持固定。门控、收缩效应和可学习投影可能抑制长路径的实际贡献；结构深度本身不保证推理能力。

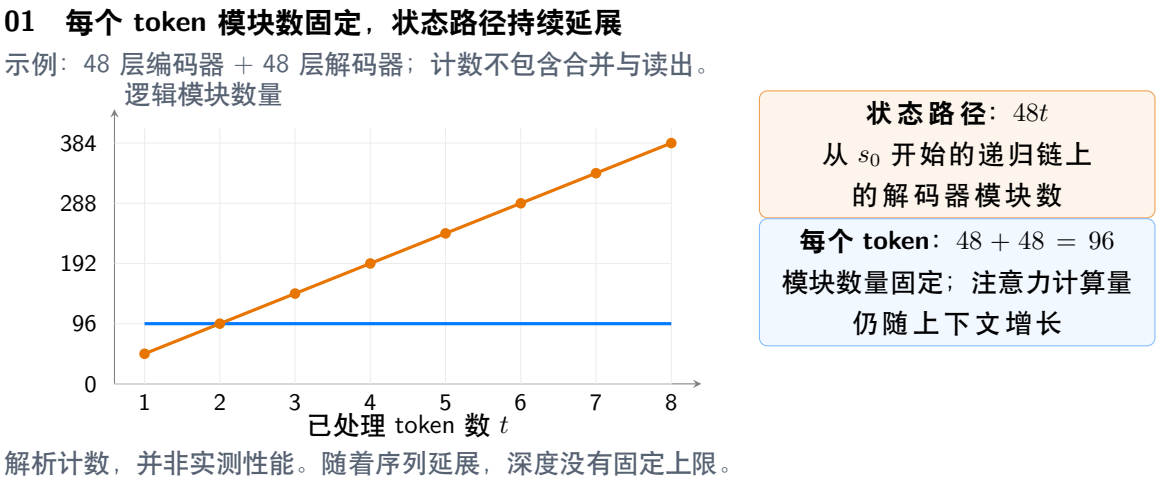

图 2: **固定每个 token 的模块数量，实现无界时间深度**。当 $L_E = L_D = 48$ 时，处理 t 个 token 后，递归路径穿过 $48t$ 个解码器模块，而每个 token 执行 96 个编码器与解码器模块。路径计数不含编码器模块，衡量的是结构深度，而非有效推理质量或实际耗时。

4 模型与硬件协同设计

4.1 围绕递归核心的并行计算

对于已知训练序列或提示词，可使用 token 并行内核计算编码器特征与记忆投影。解码器随后遵循状态依赖执行。如图 3 所示，可以将独立序列中已就绪的解码器转移组成批次：每个序列贡献其下一次状态更新，同时维护自身的编码器 KV 前缀、解码器 SWA 缓存、递归输出与位置。这提供了批次层面的并行性，并不声称递归解码器内部存在序列层面的并行性。

推理时，编码器单步计算、记忆追加、合并、解码器模块与读出形成重复的执行调度。联合调度独立请求可增大矩阵运算规模并提高权重复用。小批次或不均匀的序列长度可能限制利用率。本文不假设一般非线性解码器存在精确并行扫描。

4.2 内存访问与参数复用

对于固定 token 前缀和参数集，源自编码器的 KV 不再变化。解码器层读取这部分记忆，同时将源自解码器的 KV 追加到各层有界 SWA 缓存。这些缓存不能由编码器记忆替代。共享记忆组减少 KV 存储与投影，而使用更多记忆组则保留逐层不同的记忆变换。绑定兼容的编码器与解码器权重可减少参数存储占用，并可能有利于权重常驻，但本身并不减少模块计算次数，也不保证更低延迟。

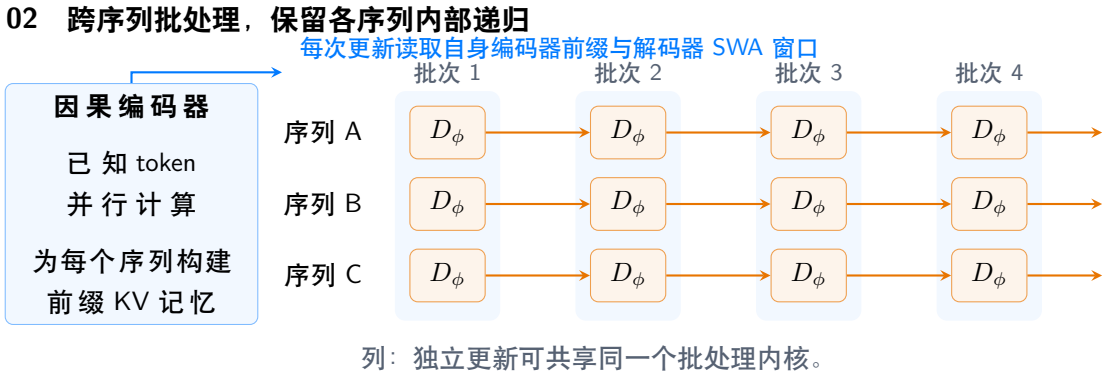

图 3: **围绕顺序核心的硬件并行**。独立序列将已就绪的解码器更新组成批次。每一行保持完整状态顺序，并读取自身的编码器 KV 前缀和解码器 SWA 窗口。已知 token 可在回放前并行编码；生成 token 需要增量编码。记忆复用、批处理和内核融合是实现目标，而非已经测得的加速。

在依赖关系和数值语义允许时，归一化、门控、状态投影与残差相加可考虑融合执行。交叉注意力只能读取有效编码器前缀记忆，SWA 只能读取有效局部窗口；读取未来条目的更快内核会改变模型。这些是实现目标，并不表示内核已经完成或已经测得带宽节省。

4.3 训练内存与执行保真度

激活检查点以重计算换取更少的激活存储，同时保留完整的时间反向传播（BPTT）。编码器批处理、跨样本解码器批处理和检查点位置，应结合序列长度与加速器内存共同选择。优化目标是在参考递归计算下获得有效训练与推理吞吐量，而不是通过近似消除递归。

Zhang et al. [2026] 将实际执行策略视为权重与执行选择的共同函数；后者包括精度、缓存构建、归约和采样变换。硬件与算法选择在回放保真度处交汇：内核精度、随机操作、位置约定和缓存生命周期必须与被评估策略一致。完整提示词递归确实产生额外成本。协同设计旨在高效执行这些计算；在获得测量结果前，硬件高效的训练与推理仍是工程目标。

5 训练目标

5.1 自回归预训练

基础目标是全序列下一 token 预测：

$$\mathcal{L}_{\mathrm{PT}}(\Theta) = -\mathbb{E}_{x_{1:S}}\left[\frac{1}{S-1}\sum_{t=1}^{S-1}\log p_\Theta(x_{t+1} \mid x_{1:t})\right]. \tag{20}$$

初始化 $H_0 = (s_\star, \varnothing)$，计算因果编码器特征，并在位置 $1, \ldots, S-1$ 上展开完整解码器状态。每个非 BOS 目标均接受监督。编码器、记忆投影、合并模块与解码器联合训练。独立文档重置递归输出、解码器 SWA 缓存、编码器缓存和位置，并使用彼此隔离的注意力掩码。必须使用完整

文档，或明确声明初始上下文约定的片段；不能在切分片段时悄然丢弃状态，却仍声称计算了完整历史似然。

5.2 监督微调

对助手目标令 $m_{t+1} = 1$，对用户、系统、工具或填充目标令其为 0。对于至少有一个选中目标的样本，优化

$$\mathcal{L}_{\mathrm{SFT}}(\Theta) = -\mathbb{E}_x \left[\frac{1}{\sum_{t=1}^{S-1} m_{t+1}} \sum_{\substack{1 \le t < S \\ m_{t+1}=1}} \log p_\Theta(x_{t+1} \mid x_{1:t}) \right]. \tag{21}$$

损失掩码不屏蔽状态更新，也不分离编码器记忆、递归输出或解码器 KV 的梯度。解码器处理所有先前上下文 token，包括用户和工具消息。助手损失的梯度可以流经这些提示词计算。状态不在助手边界重置，也不需要额外的边界适配目标来修复提示词专属的转移变化。

5.3 模型与 RL 算法协同设计

令 $c = x_{1:T}$ 为提示词，$y = (y_1, \ldots, y_N)$ 为采样响应，其中 $y_i = x_{T+i}$。策略为

$$\pi_\Theta(y \mid c) = \prod_{i=1}^{N} p_\Theta(y_i \mid c, y_{<i}). \tag{22}$$

对于不依赖 Θ 的序列奖励 $R(c, y)$，定义 $J(\Theta) = \mathbb{E}_{c,\, y \sim \pi_\Theta}[R(c, y)]$。其同策略得分函数梯度为

$$\nabla_\Theta J = \mathbb{E}_{c,\, y \sim \pi_\Theta} \left[(R(c, y) - b(c)) \sum_{i=1}^{N} \nabla_\Theta \log p_\Theta(y_i \mid c, y_{<i}) \right], \tag{23}$$

其中 $b(c)$ 是与响应无关的基线，在策略梯度中视为常量。对数概率回放期间，离散采样 token 保持固定，但求导包含其连续递归计算。架构不要求特定奖励函数、裁剪规则或 KL 正则项。

图 4 总结了回放约定。采样器记录每个动作在实际采样分布 μ 下的行为对数概率。训练器在参数 Θ 下更新时，使用这些参数重新计算因果编码器特征，并从相同序列初始化出发，沿完整提示词与已采样响应前缀重建递归输出和每个解码器 SWA 缓存。由此得到当前策略对数概率与逐 token 比率

$$r_i(\Theta) = \exp\left(\log p_\Theta(y_i \mid c, y_{<i}) - \log \mu(y_i \mid c, y_{<i})\right). \tag{24}$$

参数和采样约定相同时，训练器与采样器表示相同计算。原始模型分布不能与温度缩放或截断后的采样分布混为一谈。精确重要性采样要求在相关历史处有 $p_\Theta(\cdot \mid h) \ll \mu(\cdot \mid h)$。对于未截断 softmax 目标，使用 top-k 或 top-p 行为采样通常违反该条件。行为对数概率必须包含温度、截断和重新归一化；仅记录元数据无法恢复缺失的支持集。这里的目标是原始模型策略 p_Θ。若使用变换后的目标，必须在整个目标函数和比率中一致地替换目标概率。

依据 Zhang et al. [2026] 对执行层面的区分，在某一参数值处前向概率相同，不足以证明策略梯度相同：训练器必须对递归目标计算求导，或采用前向与反向都等价的实现。因此，RLT 同时规定完整历史前向回放及其梯度路径。仅共享架构转移，并不能证明数值内核一致。

正确的对数概率并不会使任意离策略损失无偏。对于固定提示词和序列奖励，精确轨迹重要性采样使用 $\prod_i r_i$，并受支持集与可积性条件约束。逐 token 裁剪或其他替代目标引入各自的算法选择。架构导致的状态失配与常规策略滞后是不同问题。

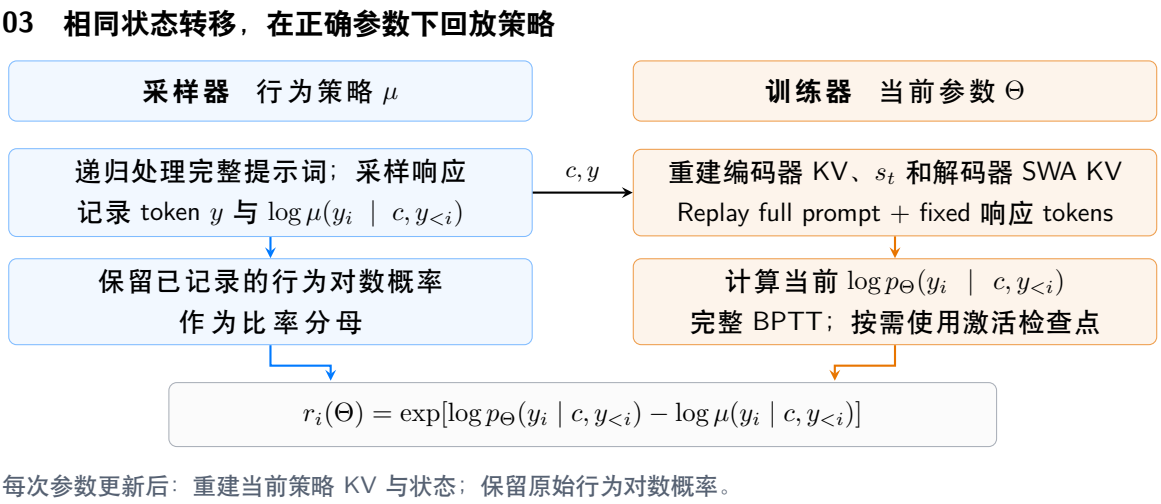

图 4: **遵循递归状态的策略回放**。采样器保留行为概率；训练器在当前参数下重建完整前缀。不能用旧递归输出或旧解码器 SWA KV 替代当前策略状态。相同参数与执行约定给出相同条件策略；离策略估计与数值差异仍是独立问题。

5.4 梯度、激活检查点与状态过时

教师强制允许对已知 token 执行并行因果编码器遍历。解码器回放在完整历史上仍然是顺序的。即使损失仅施加于响应 token，完整 BPTT 也会沿编码器记忆、递归输出和解码器 SWA KV，在提示词与响应之间传播梯度。在 `no_grad` 下运行提示词递归，若使用当前参数重算状态，可保持前向概率，但会遗漏提示词状态的导数，因此是一种梯度近似。

参考训练计算采用完整 BPTT，并按需使用激活检查点。检查点在反向传播期间，以相同参数与随机状态重算前向操作；它改变内存和计算成本，而不有意截断梯度。截断 BPTT 是可选近似：保留数值，但显式分离选定边界张量的梯度。仅分离 s_t 仍留下经由解码器 KV 和编码器记忆的梯度路径；截断规范必须明确所有保留和分离的路径。这可以在缩短梯度跨度的同时，保留当前策略前向概率。

一次回放内保持参数固定。优化器更新后，此前计算的编码器 KV、递归输出和解码器 SWA KV 通常不再是当前策略值。应从序列起点重算，或从相同参数与执行约定下产生的精确前缀检查点重算。这适用于对同一采样轨迹执行的每一轮重复优化。采样器的旧隐藏状态不能替代当前策略回放。相应地，其记录的行为对数概率仍作为实际采样动作之策略的比率分母。

6 多轮服务与缓存语义

对话被序列化为从 BOS 开始的 token 历史。新的用户消息、工具结果、角色分隔符和助手 token 均执行编码器与解码器更新。工具返回时，利用缓存编码器前缀对新观测片段编码，然后从已有状态出发，逐个处理所有新 token，推进解码器。恢复生成时不重置状态。参考策略仅在新的独立序列或显式定义的上下文重置处重置。

精确前缀快照包含编码器缓存 C_t^E、源自编码器的交叉注意力记忆 $M_{\leq t}$、完整解码器状态 $H_t = (s_t, C_t^D)$、token 与位置元数据、SWA 窗口约定和模型版本。兼容的固定权重快照可复用于推理，因为状态与服务分割位置无关。复现采样输出还需要采样器随机状态。仅从文本恢复时，需要回放以重建隐藏状态；权重更新会使复用旧状态不再满足精确当前策略计算的要求。

删除、编辑或截断前缀会改变条件历史。应从有效的更早检查点重算，而不能保留包含已删除内容的状态。同样，追加一个会改写更早 token 的模板，并不是仅追加的缓存操作。在多轮 RL 中，外部提供的 token 更新完整状态，但不是采样策略动作，因此不附加动作重要性比率因子。其状态更新在完整 BPTT 中仍保持可微。

7 相关工作

源自编码器的记忆。 YOCO 为上层交叉解码器构建可复用 KV 记忆，并支持预填充提前退出 [Sun et al., 2024]。DeepSeek-V4.1-Flash 从最终编码器状态投影得到解码器全局 KV，并单独处理解码器 SWA [DeepSeek-AI, 2026]。RLT 保留源自编码器的记忆，但让每个提示词 token 都经过递归解码器。因此，它不继承这些设计在整个提示词阶段跳过解码器的机制。

时间反馈。 Feedback Transformer 将已经处理的历史表示提供给未来计算 [Fan et al., 2020]。Recurrent Transformer 从各层输出构建该层持久 KV，并通过精确分块调度改善内存数据搬运 [Oncescu et al., 2026]。RLT 则将前一个最终解码器输出反馈到下一个解码器输入；其全局注意力记忆源自编码器，局部 SWA KV 源自解码器。提示词与响应均承载递归。区别在于反馈位置与记忆表示，而非声称 RLT 预填充完全并行。逐层 RT 的分块结果并不会自动成为适用于此种完整解码器递归的内核。

沿深度复用。 Universal Transformers 沿深度共享计算 [Dehghani et al., 2018]，而递归深度潜在推理通过迭代递归模块扩展计算量 [Geiping et al., 2025]。RLT 的状态跨 token 推进，同时其权重绑定配置额外共享兼容的编码器与解码器权重。单独的权重绑定或时间递归均不足以确立新颖性或质量提升。

8 结论

RLT 结合三项设计原则：具有无界时间深度的潜在推理、模型与硬件协同设计，以及模型与 RL 算法协同设计。连续解码器状态将计算贯穿每个提示词与响应 token，同时保持每个 token 的模块数量固定。编码器记忆分离在递归核心周围提供并行计算、记忆复用和激活检查点

机会。共享状态转移与精确当前策略回放消除训练器和采样器之间由提示词边界引起的结构性失配，为可靠的 RL 扩展提供基础。计算定义已经明确；实际推理质量、硬件效率和扩展表现仍需未来验证。

参考文献

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. Technical report, DeepSeek-AI, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash/resolve/main/DeepSeek_V41_Tech_Report.pdf. Sections 2.2 and 3.2.2; publicly available from the official DeepSeek model repository.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018. URL https://arxiv.org/abs/1807.03819.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of transformers with feedback memory. *arXiv preprint arXiv:2002.09402*, 2020. URL https://arxiv.org/abs/2002.09402.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025. URL https://arxiv.org/abs/2502.05171.

Costin-Andrei Oncescu, Depen Morwani, Samy Jelassi, Alexandru Meterez, Mujin Kwun, and Sham Kakade. The recurrent transformer: Greater effective depth and efficient decoding. *arXiv preprint arXiv:2604.21215*, 2026. URL https://arxiv.org/abs/2604.21215.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. *arXiv preprint arXiv:2405.05254*, 2024. URL https://arxiv.org/abs/2405.05254.

Yifan Zhang et al. Reliable RL scaling requires accounting for Prefill–Decode kernel mismatch. Technical report, Pretraining-RL-Science project, August 2026. URL https://github.com/yifanzhang-pro/Pretraining-RL-Science/blob/master/Prefill_Decode_Kernel_Mismatch.pdf. Dated August 6, 2026; revised August 24, 2026.

附录

A 参考执行调度

完整解码器状态为 $H_t = (s_t, C_t^D)$，初始状态为 $H_0 = (s_\star, \varnothing)$。编码器续接状态与源自编码器的交叉注意力记忆单独维护。所有调度均使用式 (16) 中的模块顺序与窗口约定。

A.1 递归提示词预填充

1. 以 $x_1 = \text{BOS}$ 和 $H_0 = (s_\star, \varnothing)$ 开始一个独立序列；初始化编码器缓存与位置。
2. 对 $x_{1:T}$ 执行并行因果编码，并构建编码器 KV 记忆。
3. 对 $t = 1, \ldots, T$，计算 $H_t = D_\phi(\text{Merge}(e_t, s_{t-1}); M_{\le t}, C_{t-1}^D, t)$。各解码器层仅读取允许访问的历史 SWA 条目和当前 KV。绝不向更早查询暴露未来编码器或解码器 KV。
4. 从 s_T 预测首个响应 token。保留 H_T、编码器缓存、记忆与位置元数据，供下一次更新使用。

历史解码器 KV 由此前的递归更新产生。并行因果编码器预填充，并不会使普通并行 SWA 解码器遍历与该递归等价。

A.2 生成与新的外部输入

从 s_t 预测的分布中采样 x_{t+1}，随后通过更新编码器缓存和记忆，并将完整解码器状态推进至 H_{t+1}，来消费该 token。所得状态预测 x_{t+2}。每个被消费的 token 恰好执行一次递归更新，并在每个解码器 SWA 层插入一次 KV。

当用户或工具提供多个 token 时，可基于已有前缀，使用因果批处理计算其编码器表示。解码器仍从保存的完整状态出发，按 token 顺序更新。外部 token 同时更新 s_t 与 C_t^D。

受长度限制的生成可能推迟消费其最后输出的 token。此时缓存 API 必须单独记录该待处理 token，并在任何后续 token 之前消费它。缓存表示已消费前缀，不包含待处理 token。这样可以避免恢复时重复更新或漏掉更新。

A.3 预训练与 SFT

1. 使用因果编码器计算 $e_{1:S-1}$，并构建记忆。
2. 初始化 $H_0 = (s_\star, \varnothing)$；展开所有位置 $1, \ldots, S-1$，包含每次逐层 SWA 缓存更新。
3. 预训练累积所有有效下一 token 损失，SFT 则仅累积助手目标损失。所有上下文 token 都执行状态更新。
4. 按每个样本选中目标的数量归一化，再按式 (20) 与式 (21) 对样本取平均。对完整计算进行反向传播，必要时使用激活检查点。

没有选中目标的样本不参与损失平均。另一种按 token 加权的批次归一化定义了不同的样本权重，应明确说明。

打包在一起的独立序列使用独立的递归状态、解码器 SWA 缓存、编码器缓存、位置和注意力掩码。两种注意力机制均不得跨越文档边界。损失掩码不能替代注意力掩码或完整文档状态重置。训练中的缓存操作必须保留自动微分依赖，或支持等价重计算。

A.4 当前策略 RL 回放

1. 读取采样轨迹 token 历史、动作掩码、实际行为对数概率和采样元数据。行为概率包含所有采样变换与重新归一化。
2. 在整个前向回放和反向传播期间固定当前参数值。使用目标策略的位置、SWA 和执行约定，且不禁用参数梯度。
3. 重算已知历史的编码器表示。从 $H_0 = (s_\star, \varnothing)$ 出发，遍历所有提示词 token，重建递归输出与每个解码器 SWA 缓存。
4. 按顺序回放后续 token。在消费每个采样动作之前，从前一状态读取其当前策略对数概率。若采样了 EOS，也应纳入。消费为后续预测所需的所有中间外部 token。
5. 由动作对数概率、奖励或优势，以及所需行为比率构建选定的 RL 损失。不给外部用户或工具 token 赋予策略动作因子。
6. 反向传播、更新参数，并在下一次精确当前策略回放前，使依赖参数的编码器和解码器缓存失效。

动作掩码只选择策略损失项，不得禁用穿过外部 token 更新的梯度追踪。回放所有助手轮次及其间的外部上下文。在 `no_grad` 下回放提示词，或分离提示词 KV 的梯度，可以保持前向值，但不属于参考完整 BPTT 计算。

声称精确重要性采样时，必须在相关历史处覆盖目标支持集。记录已采样动作的概率，不能修复未采样动作缺失的支持集。逐 token 加权或轨迹级加权属于 RL 算法选择，并非回放调度的必然结果。

确定性的策略前向计算（例如禁用 dropout）可避免内部随机计算带来的歧义。否则，策略必须规定如何对这些随机性进行条件化或边缘化；一次任意随机回放通常并不等于边缘动作概率。仅匹配调度不能确立不同内核或精度设置下的数值相等。

B 因果性与梯度路径

命题 B.1 (因果性)**.** *在确定性执行下，若编码器因果、编码器记忆受前缀限制，且解码器 SWA 因果，则 $H_t = (s_t, C_t^D)$ 仅依赖 $x_{1:t}$ 与包含 $s_\star$ 在内的参数。*

证明. 编码器因果性意味着每个 e_j 仅依赖 $x_{1:j}$，因此 $M_{\le t}$ 仅依赖 $x_{1:t}$。初始状态不包含未来 token 信息。假设 H_{t-1} 仅依赖 $x_{1:t-1}$。下一次更新使用该状态、e_t 与 $M_{\le t}$。在每个解码器层，当前 KV 由因果可用的层输入构建，且 SWA 不读取大于 t 的位置。追加当前 KV 和淘汰旧条目都不引入未来信息。因此，s_t 与 C_t^D 均仅依赖 $x_{1:t}$，归纳完成。 □

固定教师强制下的编码器特征，用固定槽位表示解码器缓存，并在窗口尚未填满时使用有效性掩码，定义

$$\mathcal{J}_t = \frac{\partial H_t}{\partial H_{t-1}}. \tag{25}$$

则有

$$\frac{\partial H_t}{\partial H_j} = \mathcal{J}_t \mathcal{J}_{t-1} \cdots \mathcal{J}_{j+1}, \qquad j < t, \tag{26}$$

其中

$$\mathcal{J}_t = \begin{pmatrix} \dfrac{\partial s_t}{\partial s_{t-1}} & \dfrac{\partial s_t}{\partial C_{t-1}^D} \\ \dfrac{\partial C_t^D}{\partial s_{t-1}} & \dfrac{\partial C_t^D}{\partial C_{t-1}^D} \end{pmatrix}. \tag{27}$$

仅包含 $\partial s_t / \partial s_{t-1}$ 的乘积通常会遗漏经过解码器 KV 的路径。仅靠归一化不能为这些雅可比矩阵的乘积提供界。

编码器记忆提供对历史编码器信息的访问。解码器 SWA 进一步提供近期解码器激活的缓存投影。两者都不是包含所有先前解码器状态的无限制档案。被淘汰条目仍可能通过此前消费过它们的状态或保留激活，影响后续计算。

完整参数导数还包含编码器特征、记忆、每次转移和解码器 KV 投影对参数的依赖。令 B_T^E 表示响应续接所需的全部编码器侧边界张量，包括编码器续接 KV 和源自编码器的交叉注意力记忆。对于响应损失 $\ell(\Theta, H_T(\Theta), B_T^E(\Theta))$，链式法则给出

$$\frac{d\ell}{d\Theta} = \frac{\partial \ell}{\partial \Theta} + \frac{\partial \ell}{\partial H_T}\frac{\partial H_T}{\partial \Theta} + \frac{\partial \ell}{\partial B_T^E}\frac{\partial B_T^E}{\partial \Theta}. \tag{28}$$

第一项固定边界参数；其余项计入产生这些边界张量的前缀计算。特别地，

$$\frac{\partial \ell}{\partial H_T}\frac{\partial H_T}{\partial \Theta} = \frac{\partial \ell}{\partial s_T}\frac{\partial s_T}{\partial \Theta} + \frac{\partial \ell}{\partial C_T^D}\frac{\partial C_T^D}{\partial \Theta}. \tag{29}$$

即使前向概率保持不变，分离 s_T、解码器 KV 或编码器侧边界张量也会删除相应梯度路径。这些操作均不是完整 BPTT。

C 缓存状态的精确性

只有当完整缓存与当前参数、已消费 token 前缀、位置、初始化、窗口语义和策略执行设置完全匹配时，缓存前缀才可替代用于概率评估的前向回放。它包含递归输出、每个解码器 SWA 缓存、编码器续接状态、源自编码器的记忆和所需元数据。随机计算必须按策略定义处理。

已经分离梯度的缓存不提供完整梯度训练所需的导数图。必须保留或重算前缀计算图，使梯度到达所有依赖参数的边界张量。旧权重下构建的缓存通常既不满足当前策略数值要求，也不满足完整梯度要求。

一次更新内的激活检查点在相同参数和随机状态下重算激活。可变缓存实现必须在重计算时恢复检查点内容，且不得覆盖反向传播仍需使用的激活。

截断 BPTT 则在保留数值的同时切断选定梯度依赖。完整解码器状态的梯度分离为

$$\widetilde{H}_t = (\mathrm{stopgrad}(s_t), \mathrm{stopgrad}(C_t^D)). \tag{30}$$

仅分离 s_t 仍可能留下经过解码器 KV 的路径；仅分离解码器 KV 则留下经过递归输出的路径。编码器侧边界张量还可能提供跨越同一边界的额外路径。任何截断方案都必须说明哪些张量被分离梯度。

SWA 淘汰限制未来直接访问旧 KV，但其本身不是停止梯度操作。完整 BPTT 仍对淘汰前消费过这些条目的计算求导。因此，推理缓存大小并不能限制完整 BPTT 的激活存储量。

在更改权重后处理后续分块，同时保留早期状态或 KV，是另一种独立的过时状态近似。这些区别同样适用于预训练、SFT 和 RL。