

隐状态推理与循环 Transformer

从问题、机制到证据与研究设计

内部研究读本

文献检索截止：2026 年 9 月 19 日

阅读说明

本书面向熟悉基础机器学习、希望系统理解隐状态推理与循环 Transformer 的读者。两条路线围绕同一问题展开：模型怎样增加有效的内部计算？它们在状态所在位置、参数共享方式、监督来源与预算分配上并不相同。Coconut 的连续状态反馈与循环深度语言模型提供了两个具体入口 [9, 12]。

阅读时请区分三种陈述：论文在给定条件下报告的结果、对机制的解释，以及本书提出的综合判断与研究设想。概念图辅助解释机制，不表示未经核验的实验数据。引用年份按所列来源版本记录；预印本与正式发表版本可能不同。成本比较分别讨论参数量、计算量、串行步骤、缓存、延迟与训练代价。

本书的来源、科学审读与版本记录见随附研究目录；逐页视觉验收状态以对应 PDF 哈希的验收记录为准。本书是中文内部研究读本，检索截止为 2026 年 9 月 19 日，不宣称穷尽该领域或经过出版同行评审。建议依次阅读第 1 至 4 章的问题与连续反馈、第 5 至 8 章的循环机制与停机、第 9 至 12 章的理论、成本与研究设计。术语速查位于书末；版本与全文路径见随书的来源映射。

目录

阅读说明	i
第一章 阅读入口：推理发生在哪里？	1
1.1 从一个看似简单的问题开始	1
1.2 先沿计算图追踪，再给方法命名	1
1.3 两条坐标轴，比单一分类树更有用	2
1.4 “多算”与“有效”之间缺少哪几步？	3
1.5 怎样把一个模糊比较变成可回答的问题	3
1.6 带着问题阅读全书	4
第二章 从显式轨迹到内部计算	6
2.1 文字同时承担了几件事？	6
2.2 计算图中，新增位置不等于新增反馈轮次	6
2.3 暂停是否有效，取决于怎样学会暂停	8
2.4 不展示的文字，仍然是文字：Quiet-STaR	8
2.5 上下文和缓存：过去的结果如何继续发生作用	10
第三章 怎样学会内化推理？	11
3.1 撤掉过程以后，学习信号去了哪里？	11
3.2 先学会读教师，再学会产生教师状态	11
3.3 不指定每个状态，逐步撤掉语言支架	12
3.4 增加连续步骤以后，为什么还需要监督？	14
3.5 CODI：在终端状态建立桥梁	14
3.6 三种监督，留下三种待检验的假设	16
第四章 连续思维怎样向前推进？	17
4.1 去掉词汇接口以后，还剩什么？	17
4.2 Coconut 的实验回答了什么？	17
4.3 潜在搜索：好的解释需要多强的证据？	19
4.4 压缩一条已知轨迹，和发现新的计算不同	20

4.5	软思维并不总是“一条更短的链”	20
4.6	连续接口打开了选择，也留下了预算问题	21
第五章	循环深度的来路：让同一条规则再运行一次	23
5.1	输入有多长，并不决定需要想多久	23
5.2	ACT：把停机纳入可学习系统	23
5.3	UT：保留全局通信，重复修订整段表示	25
5.4	共享权重为什么还不等于学到算法	25
5.5	从“复用层”走向“训练更新规则”	28
5.6	用一个极小例子检查“迭代”是否成立	28
第六章	从循环更新器到预训练语言模型	29
6.1	下一词损失怎样监督不可见的计算	29
6.2	Huginn：前奏、共享核心与终曲	29
6.3	训练深度是一种数据分布	31
6.4	Ouro：多轮读出与退出分布一起训练	32
6.5	共享参数之后，缓存仍然是问题	32
6.6	怎样从训练配方判断模型学到了什么	33
第七章	两条路线的交汇：同是连续状态，计算图却不同	35
7.1	共同问题不代表同一种答案	35
7.2	把“状态”从一个向量扩展为可访问的记忆	35
7.3	连续反馈也不只有一种接口	37
7.4	计算可组合，监督却不能自动迁移	37
7.5	中间状态可读，不等于中间状态因果充分	39
7.6	一个让两条路线公平相遇的任务	40
第八章	何时停止：自适应计算与过度迭代	41
8.1	多做一步，可能是在离开正确答案	41
8.2	先把轨迹质量与读出策略分开	41
8.3	三类停止信号解决不同问题	43
8.4	训练时在哪一轮停止，也会改变学到的轨迹	43
8.5	真实模型的深度曲线应当怎样读	45
8.6	从停止阈值到可以部署的决策规则	46
第九章	理论的边界：能表示、能学会与能泛化	47
9.1	先问是哪一种保证	47
9.2	计算模型的资源不能藏在形容词里	47

9.3 循环可以逼近函数，但逼近什么函数	48
9.4 从构造走向训练，需要补上哪些条件	50
9.5 把定理变成可以使用的阅读工具	50
第十章 成本与可信评估：让每一份额外计算有账可查	53
10.1 为什么一个更短的答案仍可能更贵	53
10.2 把一条推理请求拆成计算图	53
10.3 公平比较需要不止一个切面	55
10.4 训练成本不能从推理故事中退出	56
10.5 可信效果来自怎样的测试	56
10.6 一份足以复核的最小报告	56
第十一章 扩展与机制审计：状态里到底发生了什么	59
11.1 从语言向多模态迁移，究竟增加了什么	59
11.2 层级递归是计算调度，不是架构阵营	59
11.3 可解码的中间量，尚不是被使用的中间量	62
11.4 怎样做有分辨力的因果实验	62
11.5 不把不可见误写成神秘	64
第十二章 从综述到研究程序：让新问题能够被证伪	65
12.1 可以带走的结论与仍然缺少的桥梁	65
12.2 研究一：压缩与新增计算可以怎样分离	65
12.3 研究二：共享规则、时间编码与深度外推	67
12.4 研究三：停机应预测可修复性，而不只是信心	67
12.5 研究四：多模态循环是否真的重新利用证据	69
12.6 执行顺序与可积累的失败	69
术语速查	71
术语索引	75

插图

1.1	三条计算路径的概念示意：显式思维链经词汇接口反馈，连续链直接反馈向量，循环深度重复共享内部模块。历史缓存与每轮输入注入省略；图中步数不代表等量成本。	2
1.2	两个独立问题：中间信息保存在哪里，计算次数由什么决定。同一载体可以搭配不同预算策略。	4
1.3	阅读从载体和学习信号出发，经连续链与循环深度两路汇合，最后接受成本、泛化与机制检验。	5
2.1	左侧暂停位置在一次前向前已知；右侧每一整组蓝色堆叠表示一次完整模型前向，新词元决定下一次输入。层内注意力与输入嵌入细节省略；图不比较实际时间。 .	7
2.2	Quiet-STaR 中的隐藏思考仍是离散词元。两个不同文本前缀各自生成 rationale 并支持后续预测；图中两行互不读取对方思考，混合头细节省略。	9
2.3	当前输入向量与历史 KV 缓存是不同对象：左侧是新位置的输入，右侧是各层为过去位置保存的键和值。此图比较对象，不绘制完整数据流；层数和位置数仅为示意。	10
3.1	隐式蒸馏的三个阶段：先让学生读取教师状态，再训练预测状态的 emulator，最后连接优化。测试时无真实教师轨迹，学生仍可访问原问题；图中省略该并行输入和层内注入细节。	12
3.2	逐步撤除语言前缀，每个阶段仍依靠保留后缀与答案训练。淡出的词元不是已被证明逐一存入某个隐状态。	13
3.3	CODI 共享权重的两种任务在答案前状态上对齐。虚线是教师到学生的监督目标，stop-gradient 只作用于该目标。文字损失符号表示监督，教师实际还监督轨迹，连续状态逐步生成细节省略。	15
4.1	词汇接口与连续接口的概念对照。连续输入可直接来自上一隐状态，也可经投影；最终答案仍由语言读出。历史缓存和多轮细节省略。	18
4.2	三种多路径概念：从一个状态读出候选、实际生成多条独立轨迹、显式维护搜索节点。右下 frontier 仅示意可存储的前沿对象，不是上图某一时刻的精确队列；左图不证明标准 BFS。	19
4.3	软思维构造与后续离散推理是两个分支位置。各起点均与同一问题一起输入助手，图中问题连接简写；两处 Main model 复用相同权重。分支数量仅作示意，均需计算，最后聚合答案。	21

5.1	两类步数的概念对照：左侧每次更新读取新输入，右侧对同一输入连续更新。相同蓝色表示共享更新规则；图中不表示速度或准确率比较。	24
5.2	Universal Transformer 的简化展开：同一组位置经过共享的注意力与变换块反复修订。图中省略位置和轮次编码、残差与归一化；不表示各轮能并行执行。 . . .	26
5.3	固定数据与变化状态分开：每轮更新都能读取同一个问题条件，同时修改当前候选状态。此图是迭代算法的解释性类比，不证明模型执行了梯度下降。	27
6.1	Huginn 宏观结构的简化展开：前奏产生固定条件，每轮共享核心同时接收固定条件和变化状态，终曲在选定终点读出。图示两轮仅用于解释数据流，不代表实际训练或测试轮数。	30
6.2	前向计算与梯度窗口的概念示意：所有已选步骤都执行前向更新，截断反向传播只覆盖后段。图中六步与三步窗口仅为教学例子，不是 Huginn 的实际超参数；正文所述主实验反向窗口为八轮。	31
6.3	共享权重不等于共享缓存：上图按深度保存并读取不同 KV，下图显式复用选定缓存。两者是不同设计选择，可能改变可见信息与性能；箭头表示读取缓存，不表示缓存生成过程。	33
7.1	增加计算的两种地址组织：追加潜在位置会增加可访问历史；循环深度则在固定位置集合上更新状态。颜色仅区分示意状态，不代表实测激活或质量。	36
7.2	三类连续接口：直接隐状态反馈、经词概率和词嵌入混合的反馈、共享核心内部的状态更新。中间通路示意一次残差更新；多轮累计依正文方法定义，图中不表示并行离散搜索。	38
7.3	待执行的二维预算实验：分别改变潜在位置数与核心轮数，在固定训练预算下记录准确率、FLOPs、延迟和缓存。空心格点代表待测配置，不是实验结果或推荐最优点。	39
8.1	人为构造的轨迹说明读出选择的重要性：正确状态可能早于最终轮次出现。Oracle 使用标签，是诊断上界；图中 Policy 的成功仅属于该例，不能推出实际策略总能选对。	42
8.2	三种监督终点：固定终点、抽样终点和多深度加权损失。图中的步骤数量为示意；下行橙色边表示损失计算和汇总，不是推理状态更新，也不意味着三种方法具有相同训练成本。	44
8.3	深度评估应分开记录训练阶段与梯度窗口、验证集上的深度或门控选择、测试时的固定策略及最大预算。训练配置存在来源冲突时，应先核对再解释“未见深度”泛化。	45
9.1	概念示意：存在解、训练找到解与新输入上保持正确需要不同证据；问号处是尚需补充的条件。	48
9.2	概念示意：精度、注意力选择规则与串行步骤共同限定理论模型；并列最大值平均为一个注意力结果，分格不代表具体位数。	49
9.3	概念示意：逐项放宽门控、掩码与量化条件，再分别验证学习行为；减号表示移除该条件，未表示实验已成功。	51

10.1	概念示意：共享一份权重仍可对应多次前向计算；存储、工作量和实际需要 分别核算，四次调用仅为示意。	54
10.2	概念示意：固定参数、训练计算、推理计算与延迟是四类不同的比较实验，图中 不表示这些条件能同时配平。	55
10.3	概念示意：同一份逐请求轨迹支撑效果、成本和失败模式的联合审计；路径长短 不对应实测结果。	57
11.1	概念示意：初始读取、重复读取固定视觉表示与重新编码图像是不同操作；新视 觉证据进入持续更新的状态链。三行是机制对照，不表示 MCOUT 实现了全部三 种方式。	60
11.2	概念示意：HRM 调用高低两个 Transformer 模块，TRM 复用一個网络更新潜在 状态和答案表示。图仅示模块调用与复用关系，省略多状态融合、输入注入、梯 度与精确循环次数；颜色深浅不编码模块身份。	61
11.3	概念示意：探针只读取状态，激活替换则把干净运行的状态移入扰动运行并检查 模型输出；旁路可能同时影响答案。Readout 在左上指探针，在左下指主模型后 续读出，符号节点不表示已恢复正确答案。	63
12.1	概念示意：压缩、规则复用与下一步是否有益，分别需要状态置换、深度外推和 边际收益等不同检验；图中未给出实验结果。	66
12.2	概念示意：独立扫描任务依赖深度和循环预算，并在跨步状态置换时保持或更换 步号条件。网格为空白实验设计；右侧 Label 为计算块附带的步号条件，不是生 成输出。	68
12.3	概念示意：先判断核心轨迹是否存在可修复错误，再测试调度器与分布迁移；没 有收益时返回相应机制研究。箭头表示研究顺序，非已获得的成功结果。	70

表格

1.1	用于阅读的机制坐标。表中描述是指定代表方法的设计，不是互斥的领域定义。 .	3
4.1	CCoT 原文的一个局部成本对照。数据仅对应其 LLaMA2-7B-chat、GSM8K 微调设置，不用于跨论文排行榜。	20
7.1	计算接口的机制比较，依据各论文方法节；不是性能排名。	37
10.1	本书建议的成本账本。各列是测量口径，不是任何模型的实测结果。	54

第一章 阅读入口：推理发生在哪里？

1.1 从一个看似简单的问题开始

一位同学拿到这样一道题：仓库有三箱零件，每箱十二个，检验后发现四个不能用，还剩多少个？他可以写出“先算总数，再减去坏件”，也可以只报出三十二。对旁观者而言，前者提供了中间步骤，后者没有；但输出的长短不足以判断两人究竟做了多少计算。语言模型也有类似问题。直接给出答案，不意味着没有处理问题；输出一长段解释，也不意味着每句话都承担了必要且正确的推导。

这个教学例子不是实验。它用于分开本书最关心的三个对象：**问题要求的计算、模型实际执行的计算，以及最后展示的文字**。三者可能重合，也可能相差很远。讨论“隐状态推理”时，如果只盯着第三个对象，很容易把隐藏文字当作连续计算，把答案简短当作成本降低，把内部表示可被读出当作算法已经被证明。

早期的思维链提示研究提供了一个有力入口：在提示样例中加入中间步骤，能够改变足够大语言模型在若干算术、常识和符号任务上的表现；这些模型在该研究中没有为实验重新微调。但原论文也明确指出，可读的步骤并没有完整刻画支撑答案的神经计算，正确的答案与正确的推理路径也不能互相保证 [22, 第 2、3、6 节]。因此，本书从思维链出发，却不预先把自然语言规定为推理的唯一载体。

1.2 先沿计算图追踪，再给方法命名

设输入问题为 x ，模型输出答案为 y ，中间生成的文字为 $z_1, \dots, z_T$ 。一种显式思维链计算可以抽象为

$$p_{\theta}(y, z_{1:T} | x) = \left[\prod_{t=1}^T p_{\theta}(z_t | x, z_{<t}) \right] p_{\theta}(y | x, z_{1:T}). \quad (1.1)$$

这里 θ 是模型参数， $z_{<t}$ 表示已经生成的中间文字，最后一项为答案序列的条件概率。关键不只是存在中间文字，而是这些文字重新进入后续条件：模型可以用自己刚写下来的结果继续计算。若答案本身包含多个词元，最后一项仍按通常的自回归方式展开。

隐式方法可能改变这条链上的不同接口。逐步内化方法在训练过程中缩短 z ，最终让模型直接生成 y ，没有因此自动获得一个可在测试时任意延长的内部循环 [8, 第 3 节]。Coconut 则保留逐步推进的机制，却将最后隐状态直接送作下一位置的输入向量，跳过中间的词汇解码 [12, 第 3 节]。这两者都可以不展示完整语言思维链，但计算结构已经不同。

另一路方法把循环放在一次词元预测内部。循环深度语言模型先编码输入，再反复应用一个共享模块，最后读出词汇概率。用本书的简化记号写为

$$e = P(x), \quad s_{k+1} = F_{\theta}(e, s_k), \quad p(y_1 | x) = \text{softmax}(C(s_K)). \quad (1.2)$$

Where does extra computation happen?

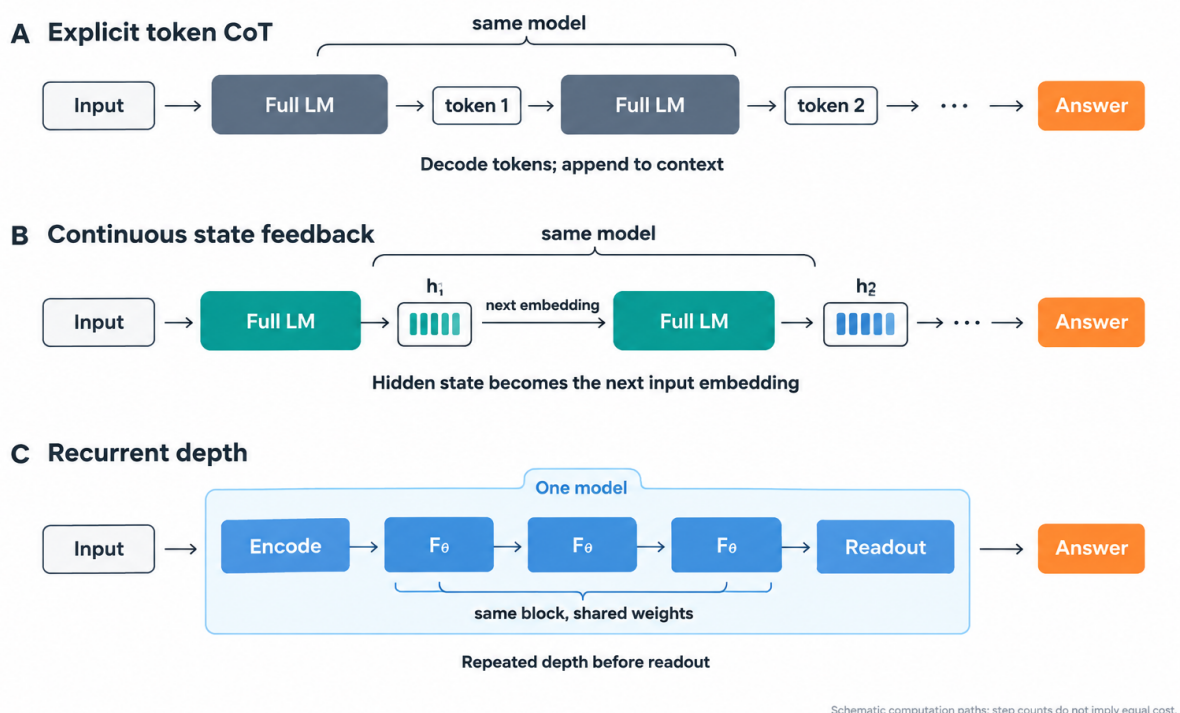


图 1.1: 三条计算路径的概念示意：显式思维链经词汇接口反馈，连续链直接反馈向量，循环深度重复共享内部模块。历史缓存与每轮输入注入省略；图中步数不代表等量成本。

来源与边界：概念性教学图；依据 [9, 12, 22] 及本章证据整理，不构成新增实验结果。

P 是输入编码部分， s_k 是第 k 轮状态， F_θ 是共享循环模块， C 是输出读出， K 是本次使用的循环次数。状态可覆盖当前序列的多个位置，而不只是一个向量。该式依据 recurrent-depth 的 prelude/core/coda 结构改写，保留每轮重新注入输入的特点；不是所有循环模型都必须采用同样的初始化和模块划分 [9, 第 3.1 节及图 2]。

三条路线可以画在同一张计算路径图中：上方是“状态到词元，再到嵌入”的反馈，中间是直接的连续向量反馈，下方是在固定序列位置上反复执行共享模块。请把图中的箭头理解为数据依赖，不要把箭头数量直接读成运行时间。已有概念图若用于这里，应保留这三个接口的不同位置。

1.3 两条坐标轴，比单一分类树更有用

本书采用两个独立问题来定位方法。第一，**状态在哪里、下一步能读到什么？** 候选答案包括已经生成的词元、连续位置表示、层间激活，以及循环模块携带的状态。第二，**计算预算怎样决定？** 它可能固定于模型结构，固定为若干隐状态步，也可能由输入相关的停机规则决定，或者在测试时由外部调节。

这张表有意没有给出一个“是否推理”的二元标签。所有神经语言模型都有隐状态，因而“使用隐状态”本身不能定义一个有区分力的新类别。本书所说的隐状态推理，主要指围绕中间表示的计算载体、训练监督或推理预算作出明确设计，使原本可能以文字表达的计算可以部分发生在内部。这个工作定义服务于比较，不宣称解决了“什么算推理”的哲学问题。

表 1.1: 用于阅读的机制坐标。表中描述是指定代表方法的设计，不是互斥的领域定义。

代表设计	中间载体	主要增加的计算位置	应单独核对的条件
显式思维链	词汇中的离散词元	自回归序列	提示、训练及生成长度
暂停位置	固定特殊词元的激活	输入序列中的额外位置	是否在训练中学会利用
逐步内化	普通前向中的隐状态	训练过程重塑既有计算	不等于新增测试时循环
连续反馈	下一位置的连续输入	依赖前一步输出的状态链	梯度、缓存及步数安排
共享深度循环	循环核心的激活	同一模块的重复应用	输入注入、深度分布、停止

来源与边界：分别依据 Deng et al. [8], Geiping et al. [9], Goyal et al. [10], Hao et al. [12], Wei et al. [22]的方法部分综合。

同理，“latent”也不一定意味着向量连续、过程不可读或计算便宜。Quiet-STaR 在观察到的文本位置后生成不直接展示的 rationale，但这些 rationale 仍是从词汇中采样出的离散词元 [26, 第 4 节]。把显示界面关掉，不会改变词汇采样这条计算边。判断方法类别时，最可靠的起点是其前向过程和训练目标，而不是论文标题。

双栏机制图的两栏同一行没有一一对应关系。图表达“一个方法可以同时有多个属性”。例如，连续表示既可能来自固定位置的助手模型，也可能来自自回归反馈；深度循环既可采用固定预算，也可结合停机机制。图中不宜把方法排成从低级到高级的阶梯，更不能让读者误以为较晚发表的方法必然覆盖较早方法的能力。

1.4 “多算”与“有效”之间缺少哪几步？

回到仓库题。如果模型第一轮编码了箱数，第二轮形成总数，第三轮执行扣除，这听起来很像一个迭代算法。但我们实际上还不知道各轮是否承担了这些功能。仅仅观察到准确率随着步数增加而提高，最多支持“这组模型在这些条件下能够利用更多计算”；它尚未告诉我们中间状态表示了什么，更未说明测试分布之外仍能按同一规则执行。

这里至少有三层证据。性能证据比较答案，机制证据考察状态与行为之间的关系，算法或理论证据进一步限定可表达的过程及其资源条件。三者有关联，却不能互相替代。读出器从一个状态中恢复出“三十六”，说明这个读出器能利用该状态中的信息；只有更进一步的干预和反事实实验，才有望判断这部分信息是否用于后续扣除。第十一章将专门讨论这种区别。

成本也必须沿实际图核算。参数共享减少了需要存储的一组不同权重，不会让重复执行的乘法自动消失；没有自然语言中间输出，仍可能有多个串行前向过程；从六十个词元缩短为六个连续位置，可能缩短推理，却同时增加训练时反向传播的依赖长度。Coconut 的方法部分明确讨论了连续位置逐个计算对训练并行性的限制 [12, 第 3 节]。因此，任何“有效”的结论都应同时说明所测任务、所用预算和对照条件。

1.5 怎样把一个模糊比较变成可回答的问题

假设有人提出：“模型甲只输出答案，模型乙会写过程，所以甲更有效率。”我们首先需要知道甲在答案前是否执行连续反馈，乙的中间文字有多长，两者使用什么模型和硬件。接着还要确

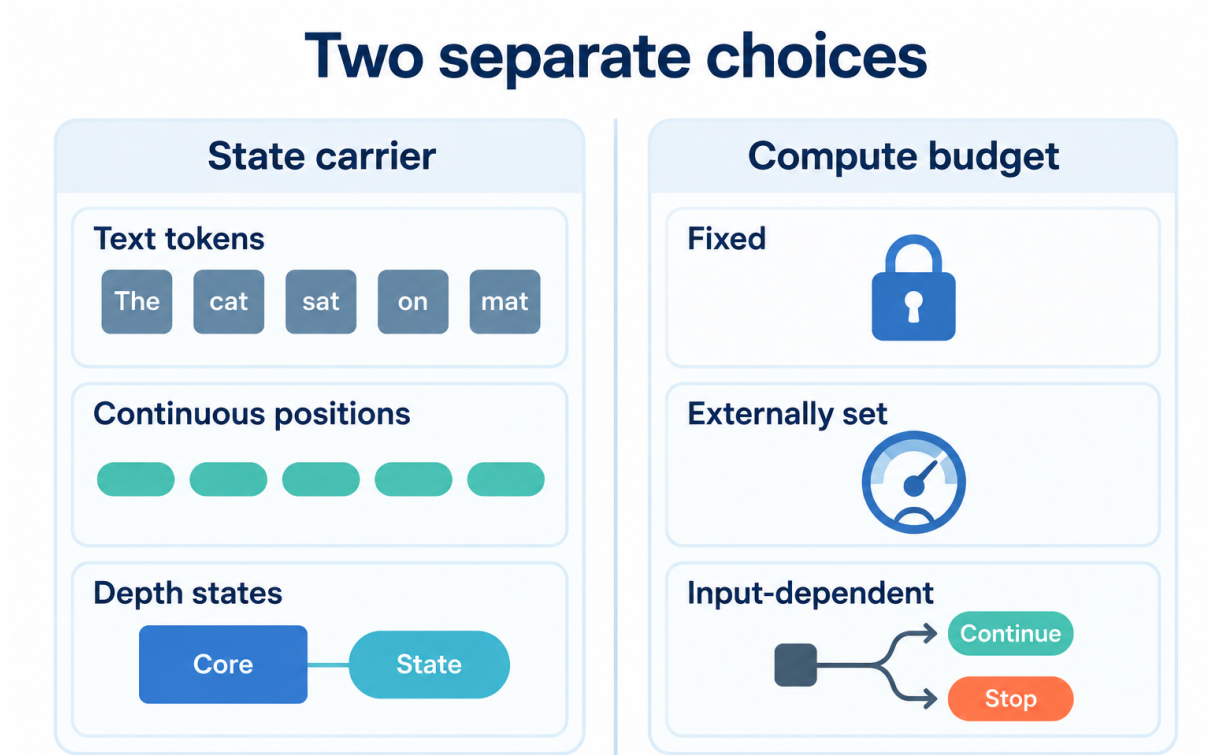


图 1.2: 两个独立问题：中间信息保存在哪里，计算次数由什么决定。同一载体可以搭配不同预算策略。
来源与边界：概念性教学图；依据 [8, 9, 12] 及本章证据整理，不构成新增实验结果。

认是否比较同一组题目、同一答案判定规则，以及甲是否为了直接回答而接受了额外蒸馏。这些信息一旦缺失，表面简洁的结论就混合了不同成本。

一个更明确的比较可以从固定模型开始：保留同一问题与输出判定，分别记录内部步骤、输出词元、延迟和准确率，再说明训练是否一致。这仍不是所有研究都必须采用的唯一实验设计。若研究问题是参数共享是否节省存储，控制条件应围绕参数与计算展开；若问题是部署时怎样分配预算，就应比较相同延迟约束下能够达到的答案质量。实验设计服务于问题，而不是预先规定某一方法获胜。

同样重要的是记录失败。如果增加循环只改善简单题，却使长问题更差，那么“平均更好”不应遮住适用范围的收缩。如果读取中间结果只在最后回答正确的样本上进行，就需要另报失败样本的机制表现。把这些问题提前带入阅读，有助于在后续章节区分方法的承诺、实验真正测到的东西，以及仍需要检验的空白。

1.6 带着问题阅读全书

本书的前四章讨论内部计算如何从显式轨迹中形成。第二章先分开文字的内容、位置和反馈作用；第三章讨论教师表示、训练课程和终端对齐怎样提供学习信号；第四章进一步区分连续推进、轨迹压缩与多路径探索。接着，第五章到第八章转向共享模块为什么可能支持迭代、怎样扩展到语言模型，以及什么时候应停止。

最后四章把两条路线放回同一套证据要求中：理论说明什么，成本应该怎样比较，机制解释如何被检验，以及哪些研究问题值得继续推进。这个顺序不是论文影响关系的年表。一个早期问

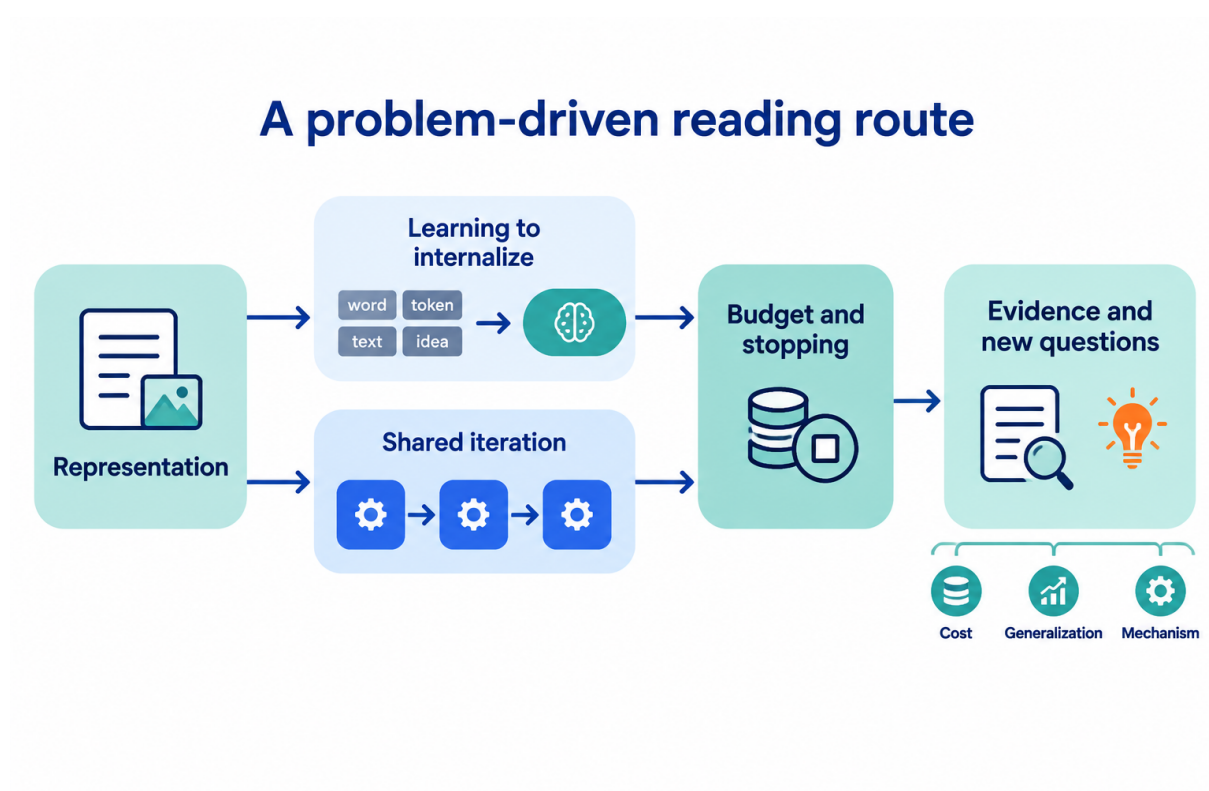


图 1.3: 阅读从载体和学习信号出发，经连续链与循环深度两路汇合，最后接受成本、泛化与机制检验。
来源与边界：概念性教学图；依据 [9, 12, 22] 及本章证据整理，不构成新增实验结果。

题可能在较晚的方法中仍未解决，一项后来的实验也可能要求我们重新理解较早的解释。

读者可以从一个具体任务带着三条记录阅读：模型在何处保留中间结果，训练怎样让它学会更新，评价怎样证明多算值得。比如关注部署延迟，可优先串联第三章、第八章和第十章；关注算法泛化，则应把第五章与第九章、第十一章一起读。无论选择哪条路线，都应保留对原文实验条件的追踪。方法名字是入口，真正需要带走的是可以比较、可以质疑，也可以重新实验的具体问题。

第二章 从显式轨迹到内部计算

2.1 文字同时承担了几件事？

让模型“把过程写出来”看似只是改变回答格式，实际上同时改变了监督、记忆和计算。以本书构造的两跳规则题为例：已知“物件甲属于类乙”“类乙属于类丙”，询问甲是否属于丙。模型可以先写出“甲属于乙”，再把这句话与第二条规则结合。已经写出的结果既是人可读的解释，也是后续预测能够重新访问的上下文。若训练数据还标注了每一步，文字又承担了学习目标。

这三种作用不能由一次准确率提升全部识别。显式思维链的优势可能来自更好的任务分解示范，也可能来自新增的自回归计算机会，或二者共同作用。原始 CoT 提示研究通过方程提示、无语义的额外输出及“先给答案再解释”等消融尝试分辨这些因素；结果依赖于任务与模型条件，不能简化为“只要多生成词元就有效”[22, 第 3.3 节]。

也应区分 prompting 与训练。给一个已训练模型补充带推理的示例，是改变当前条件上下文；用大量带步骤数据优化权重，是改变模型本身。二者都能让模型生成轨迹，却对应不同的数据费用和泛化问题。在理解后面的隐式训练方法时，若把提示实验中的发现直接当成小模型经过蒸馏后的能力界限，比较就已经失去共同起点。

2.2 计算图中，新增位置不等于新增反馈轮次

设一个 Transformer 有 L 层，当前输入有 N 个位置。对于一个已给定的输入序列，各层可以在满足注意力掩码的条件下批量计算所有位置。即使采用因果掩码，第 ℓ 层某位置读取的也是上一层对应位置及其允许访问的前缀表示。若再附加 M 个已知特殊词元，并不需要等待模型先预测第一个特殊词元，才能知道第二个应是什么。

这与自回归生成 M 个中间词元不同。后者第 $t+1$ 步的输入内容依赖第 t 步的输出；不仅在序列上增加位置，也把多次整网前向串联起来。把一个前向中的层数记为 L ，可以用 L 与约 ML 的串行算子组合来理解区别，但这只是计算图层面的示意，不是硬件延迟公式。批处理、缓存和内核调度会改变实际时间[10, 第 6 节]。

图中应把两种箭头分开：层间箭头从第 $\ell-1$ 层到第 ℓ 层，反馈箭头则从一次前向的末端回到下一次前向的输入。额外的空位置可以扩大可用的中间工作空间，却不会仅凭位置数量自动获得后一种依赖。这也是为什么“隐状态步数”在不同论文中不能直接相加、比较或换算。

暂停词元提供了一个干净的观察窗口。Pause-training 在预训练文本的随机位置插入可学习的特殊词元，不要求模型预测这些暂停词元本身；在下游微调与推理中，则在输入后附加固定数量的暂停位置，再读取目标答案。模型可以学习利用这些位置产生更合用的中间表示，而不必在其上写出词汇意义明确的理由[10, 第 3 节]。

但“没有新信息”不等于“没有新计算”。把一张纸上的已知数字抄到草稿区，不会增加题

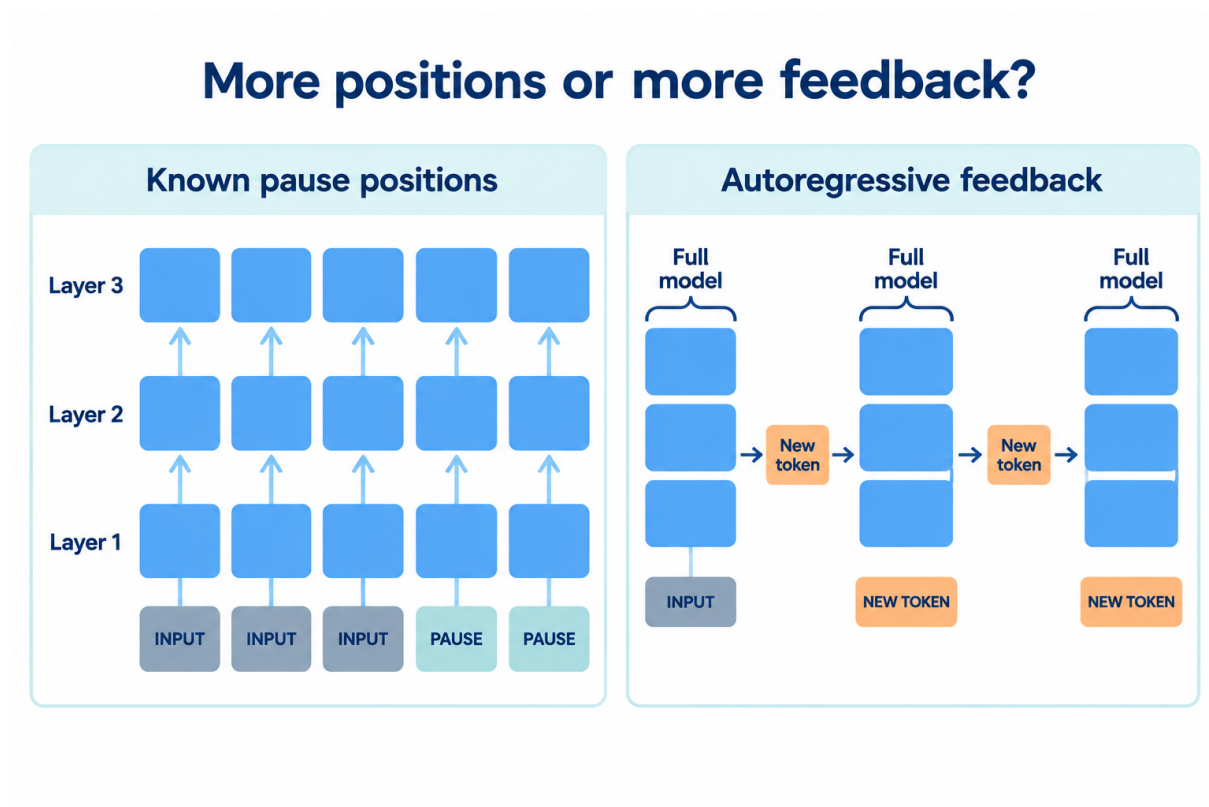


图 2.1: 左侧暂停位置在一次前向前已知；右侧每一整组蓝色堆叠表示一次完整模型前向，新词元决定下一次输入。层内注意力与输入嵌入细节省略；图不比较实际时间。

来源与边界：概念性教学图；依据 [5, 10] 及本章证据整理，不构成新增实验结果。

目事实，却可能让后续操作更容易组织。暂停位置可以作为这个直觉的类比；不能进一步把草稿区中的内容指定成乘法、搜索或符号缓存，因为论文的主要实验并没有逐一确证这些内部机制。原文自身也把“增加计算宽度”作为解释假说，而非完整的机制定理。

还应分开训练时已知的轨迹与推理时待生成的轨迹。监督训练拿到完整正确序列时，可以把所有词元一次送入模型，通过因果掩码同时学习多个下一词元目标；这通常称为 teacher forcing。模型看到的历史词元来自数据，而不是它本轮自己预测的结果。真正生成时，后续输入却必须等待前一步选择。Coconut 方法部分对连续状态训练的讨论正好凸显这个差异：连续思考没有预先给定的词元序列，训练也需先算出前一步状态，再取得下一步输入 [12, 第 3 节]。

因此，“显式长链能批量训练”与“显式长链要串行生成”可以同时成立；“连续链更短”与“连续链训练并不更容易并行”也可以同时成立。若只用推理时的词元压缩比估计训练加速，就会漏掉这一依赖方向的改变。

2.3 暂停是否有效，取决于怎样学会暂停

Pause-training 的实验设计同时改变预训练是否使用暂停、下游微调是否使用暂停，形成四种组合。它测试了两种模型规模与多类下游任务，而不是仅向普通聊天模型输入一串标点。两阶段都使用暂停的模型在多数所测任务上获益；只在普通预训练模型的微调阶段加入暂停，收益更混合，在部分任务上还会下降 [10, 第 4.1 至 4.3 节]。

这个结果带来一个比“暂停有用”更有价值的问题：模型此前学会的计算组织是否允许它利用新位置？若预训练一直鼓励紧接当前上下文产生下一个真实词元，那么增加暂停位置既增加资源，也改变分布。学习如何使用这些位置本身需要训练；新增位置不一定会被模型自动解释成“再想一会儿”。

论文还比较了训练和测试时暂停数量不同的情形。结果支持一定范围内的鲁棒性，却不支持无限增加暂停能够单调改善答案；在所测条件中，更多暂停可能降低表现，完全去掉暂停也可能造成显著退化 [10, 第 5.1、5.2 节]。这是本书第一次遇到训练预算与测试预算不匹配的问题。它并非循环深度特有，而是所有改变内部计算路径的方法都可能面对分布变化。

研究成本时还需要保留一个细节：原文预训练比较固定总词元预算，暂停位置占据其中一部分；下游加入暂停则确实提供了额外计算。因而“参数几乎不增加”不意味着“所有阶段计算完全一样”，固定处理位置数也不等于固定真实文本数据量 [10, 第 4.1、4.3 节]。如果复现实验时删除这些区别，仅保留准确率一列，就很难解释收益究竟来自哪里。

2.4 不展示的文字，仍然是文字：Quiet-STaR

暂停位置不需要生成有内容的中间句子。Quiet-STaR 则选择另一个方向：让模型在文本的许多位置生成有内容、但不作为正常输出呈现的短 rationale，并根据它们对后续真实文本预测的帮助来学习。它试图从普通文本中获得“什么时候应该想到什么”的训练信号，而不是完全依靠人工标注推导 [26, 第 3、4 节]。

方法可以分为三个有依赖关系的过程。首先，对多个文本前缀生成候选 rationale；其次，把有 rationale 与没有 rationale 的预测通过可学习的混合机制结合；最后，奖励那些使后续真实词元更容易预测的 rationale。这里的 rationale 是离散词元序列，训练使用相应的策略梯度信号。为了避免刚开始生成的不稳定思考破坏原模型，混合头为基础预测保留通道 [26, 第 4.1、4.3、4.4 节]。

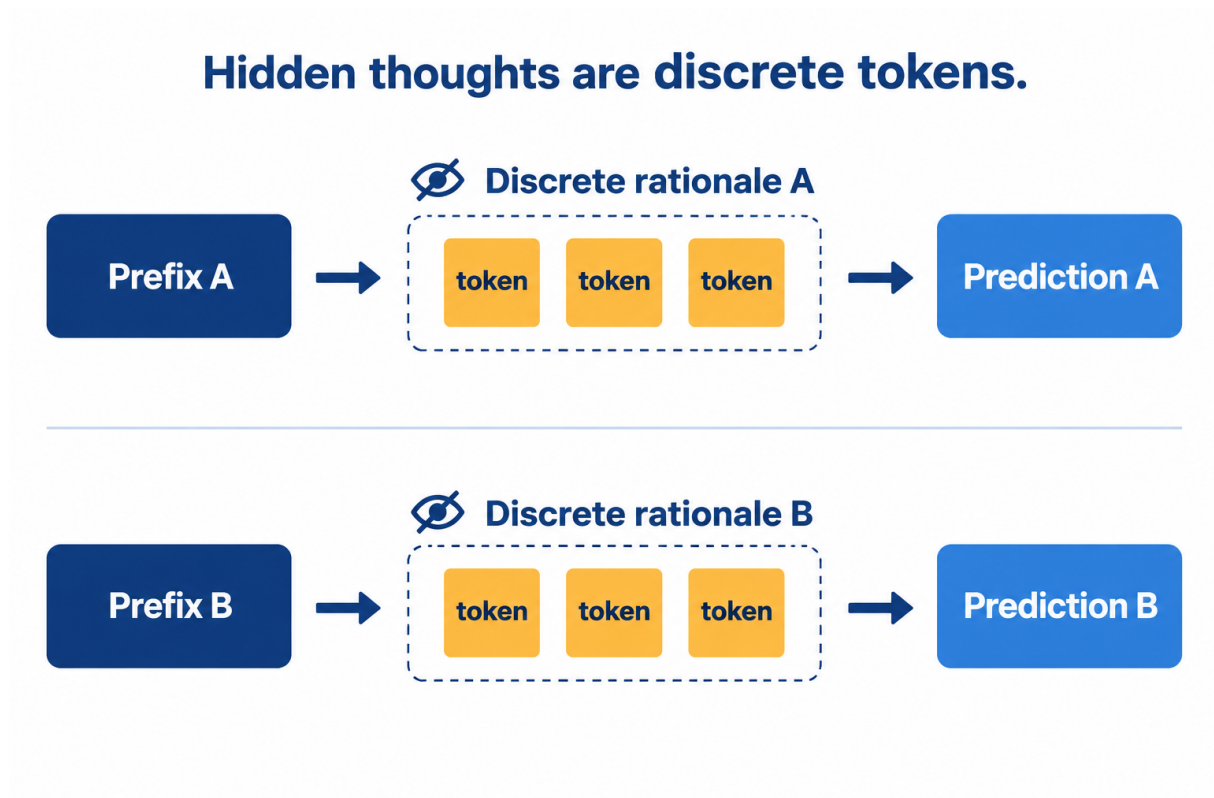


图 2.2: Quiet-STaR 中的隐藏思考仍是离散词元。两个不同文本前缀各自生成 rationale 并支持后续预测；图中两行互不读取对方思考，混合头细节省略。

来源与边界：概念性教学图；依据 [26] 及本章证据整理，不构成新增实验结果。

可以用下式概括这种教学直觉，而不把它当作原文全部实现细节：

$$R(r; x) = \log p_{\theta}(x_{i+1:i+q} \mid x_{\leq i}, r) - b_i. \quad (2.1)$$

r 是某个候选 rationale， q 指纳入评价的后续真实词元数量， b_i 是针对该位置候选的基准。奖励表达的是“这段中间文本对预测未来是否比参照更有帮助”，不是人类对其逻辑正确性的直接评分。原文还采用混合预测、teacher forcing 和截去负奖励等具体处理；因此不能用这一概括式推断无偏性或最优性 [26, 第 4.4 节]。

不同前缀的候选可以通过特殊注意力掩码并行组织，但每条 rationale 内仍有逐词生成的依赖。所谓并行并没有取消思考长度的成本，也不能让不同反事实分支任意互相读取。图中保留“观察文本主轨道”和“候选 rationale 支轨道”，有助于解释为什么用户看不到文字并不代表模型没有生成文字。

原文以 Mistral 7B 为起点，在 OpenWebMath 为主的训练及 C4 补充实验中考察语言预测与下游问答，也研究了内部 rationale 与显式 CoT 的组合。一个值得保留的结论是二者可以共存：模型可以先在内部产生离散 rationale，再输出面向读者的推理文字。与此同时，论文明确承认每个位置生成思考会带来显著额外开销，其实现没有动态决定何时开始或结束 rationale [26, 第 5、6 节]。因此，本书将它视为内部离散计算的重要参照，而非连续隐状态方法的同义词。

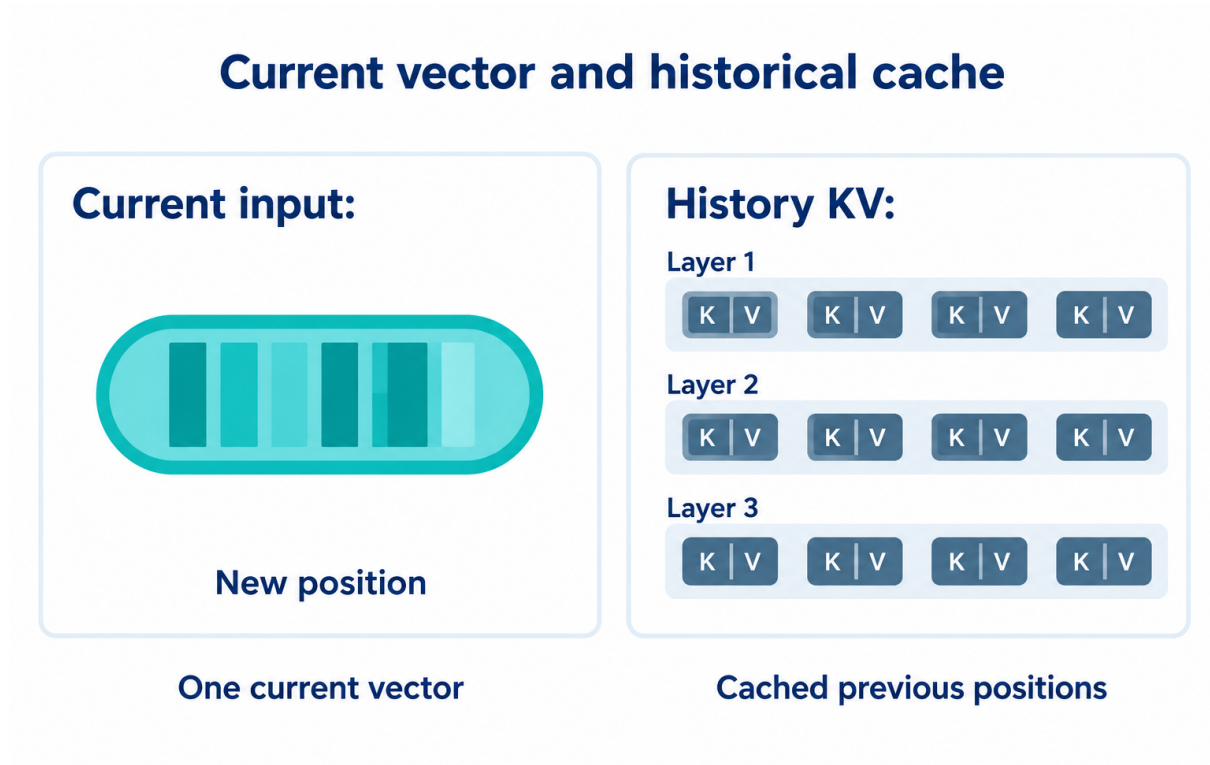


图 2.3: 当前输入向量与历史 KV 缓存是不同对象: 左侧是新位置的输入, 右侧是各层为过去位置保存的键和值。此图比较对象, 不绘制完整数据流; 层数和位置数仅为示意。

来源与边界: 概念性教学图; 依据 [12] 及本章证据整理, 不构成新增实验结果。

2.5 上下文和缓存: 过去的结果如何继续发生作用

无论中间载体是文字还是连续向量, 下一步必须能够利用过去的计算, 才可能构成有用的多步过程。标准自回归推理常用 KV cache 保存历史位置在各注意力层的键和值, 以免每一步重新计算全部前缀。这里保存的是模型特定层、特定上下文中的表示, 而不是一份可独立理解的自然语言笔记。

连续反馈方法也可以使用历史位置的缓存。改变新位置的输入, 从词元嵌入变成前一步的隐状态, 并不会自动消除历史注意力; 随着连续位置增加, 缓存也可能增长。Coconut 方法部分明确讨论使用 KV cache 避免重复计算, 同时保留逐步前向的串行依赖 [12, 第 3 节]。因此, “只保留一个向量”的画法若忽略实际历史缓存, 可能使概念图低估系统代价。

示意图把当前输入向量与每层历史 KV 列为不同对象, 不表示完整缓存更新算法。前者决定本步从哪里出发, 后者提供可访问的上下文。两者共同影响信息流, 却可能具有不同的内存与更新时间。第六章会进一步看到, 当循环进入深度维度后, 缓存还必须回答“属于哪一轮状态”的问题。

至此, 可以重新理解自然语言轨迹的地位。它是可检查的学习与推理接口, 也带来逐词生成、语法组织和对特定表述形式的依赖。去掉它并不是去掉一个装饰层, 而是同时改变学习目标、工作空间和计算路径。下一章的难点由此产生: 当不再要求模型逐字写出中间答案时, 训练究竟拿什么教会它完成那些步骤?

第三章 怎样学会内化推理？

3.1 撤掉过程以后，学习信号去了哪里？

假设一个学生已经会跟着示例完成三步计算。把所有中间步骤一次删掉，只保留问题和答案，给他的不是同一个任务的缩短版本，而是一个监督更稀疏的问题：错误答案不能直接指出哪一步出了错。对神经模型也一样。原本在每个思维链词元上存在的交叉熵信号消失后，模型必须自行找到能够支持答案的内部表示，而这些表示不一定容易通过答案损失学得。

本章讨论三种不同的帮助方式。第一种让学生模仿教师的内部表示，明确指出它应在某处包含什么信息；第二种逐步撤掉语言步骤，使学习问题连续变化；第三种同时保留显式任务和隐式任务，在答案前的关键表示上建立对齐。它们并非互斥，却分别回答了“学什么”“怎样过渡”和“在哪里对齐”三个问题。

尤其需要区分内化训练与推理时增加内部计算。前者可以只改变已有网络参数，使其直接回答；后者还需要新的计算路径，例如连续状态反馈或共享深度循环。将一个教师的长过程压入固定深度学生，不会自动赋予学生在测试时延长该过程的能力。这个区别在早期隐式 CoT 蒸馏和逐步内化工作中尤其清楚 [7, 8, 第 3 节]。

3.2 先学会读教师，再学会产生教师状态

隐式 CoT 的早期蒸馏方案把问题拆成三个阶段。首先训练学生利用教师的中间隐状态预测答案；其次训练一个 emulator，仅从问题预测那些教师状态；最后把 emulator 和学生连接起来，联合优化答案任务 [7, 第 3.1 至 3.3 节]。这比笼统地说“蒸馏思维链”更具体：教师在训练时可以读完整中间过程，实际推理时则必须由 emulator 替代这种额外信息。

教师状态具有两个轴：层和中间词元位置。设教师有 L 层，思维链有 T 个位置，记 $h_{\ell,t}^{\text{teach}}$ 为第 ℓ 层、第 t 位置的状态。论文选择跨层、跨位置的一组状态，而非复制整个 $L \times T$ 矩阵。其位置选择可写为

$$t_\ell = \min(\lfloor 1 + \Delta(\ell - 1) \rfloor, T), \quad \ell = 1, \dots, L. \quad (3.1)$$

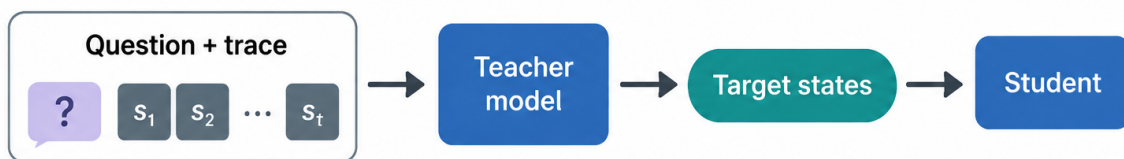
Δ 控制沿中间序列前进的间距；随着层数增加，目标中可以包含更靠后的计算信息。这个选择是一种经过实验的监督接口，并不证明自然语言每一步都天然对应某一层 [7, 第 3.1 节]。

一个最简单的状态回归目标是

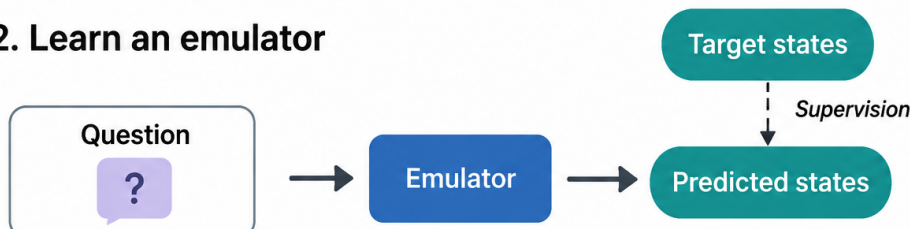
$$\mathcal{L}_{\text{state}} = \sum_{\ell=1}^L \left\| \hat{h}_\ell(x) - h_{\ell,t_\ell}^{\text{teach}} \right\|_2^2, \quad (3.2)$$

其中 $\hat{h}_\ell(x)$ 是 emulator 的预测。这个式子揭示了一个常被忽略的障碍：同一题目可能存在多个有效过程。如果教师沿不同过程产生不同状态，单个平方误差预测器可能趋向这些状态的均值，

1. Read teacher states



2. Learn an emulator



3. Couple and optimize

Inference needs no teacher trace.



图 3.1: 隐式蒸馏的三个阶段：先让学生读取教师状态，再训练预测状态的 emulator，最后连接优化。测试时无真实教师轨迹，学生仍可访问原问题；图中省略该并行输入和层内注入细节。

来源与边界：概念性教学图；依据 [7] 及本章证据整理，不构成新增实验结果。

而均值未必对应任何有效的后续计算。原文因此引入混合表示，并使用中间词元监督混合分量，减轻多个推理路径混在一起的问题 [7, 第 3.2 节]。

这里的图区分训练与测试：训练阶段的教师可以读取完整轨迹，测试阶段不能把真实中间结果继续送给学生。若图中只画“教师向学生传递知识”的箭头而不标注可见信息，就会掩盖方法最难的部分，即怎样从问题重建足够有用的表示。

实验并没有消除这种难度。原文在多位数乘法和扩增后的 GSM8K 上比较了显式、隐式及直接回答模型；结果显示隐式蒸馏可以明显改善直接回答，但在自然语言数学题上仍落后于显式 CoT。推理还需 emulator，因此它比显式长链快，并不意味着比普通直接回答没有额外开销 [7, 第 4、5 节及表 3]。这是一种经过监督的性能与速度折中，而不是已经把任意长推理无损装入固定层数的证明。

3.3 不指定每个状态，逐步撤掉语言支架

逐步内化采用更直接的训练路径：先让模型学习完整思维链，再从链的开头逐渐移除词元。设原链为 $z_{1:m}$ ，当前已删除的前缀长度为 s ，目标变成

$$\mathcal{L}_s(\theta) = -\log p_\theta(z_{1+\min(s,m):m}, y \mid x). \quad (3.3)$$

当 $s = 0$ 时，模型学习完整过程；当 $s \geq m$ 时，只剩直接回答。中间阶段并不告诉模型被删掉的每个词元应该被编码在哪里，而是要求它在失去这段条件后仍能产生剩余步骤和答案 [8, 第 3 节]。

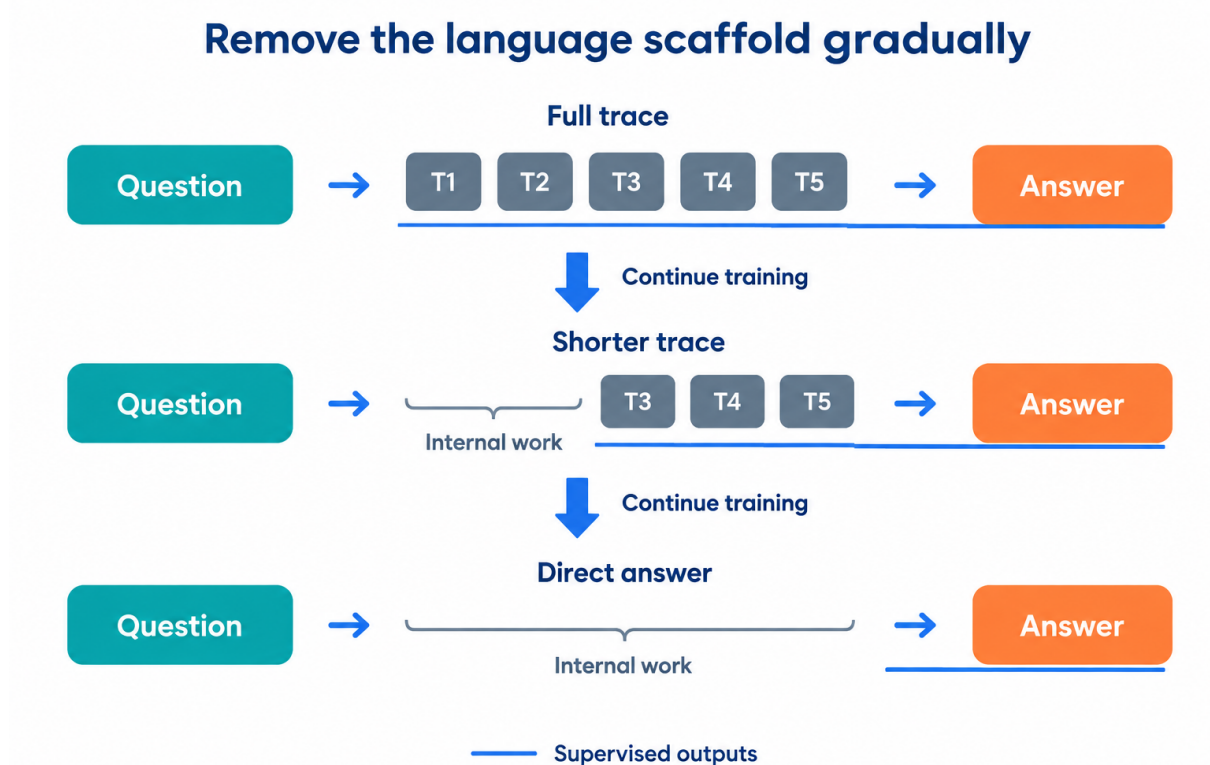


图 3.2: 逐步撤除语言前缀，每个阶段仍依靠保留后缀与答案训练。淡出的词元不是已被证明逐一存入某个隐状态。

来源与边界：概念性教学图；依据 [8, 12] 及本章证据整理，不构成新增实验结果。

这个目标不是把已经训练好的模型在测试时删去文字便完成了压缩。每一次删除都改变了训练分布与条件概率任务。模型需要经过优化才能补偿原先由语言前缀提供的信息。把一个“先求总数，再减去坏件”的例子逐步改成“……再减去坏件”，会迫使前面的计算由内部完成；但也可能诱导模型学习与原过程不同的捷径。这是本书的教学解释，不是关于具体神经机制的实验结论。

原文报告了两个稳定训练的处理：删除更多词元时重置优化器状态，以及对删除数量加入小的随机偏移，让模型偶尔提前见到稍难的阶段。其消融还显示，删除方向和删除速度会影响训练稳定性；在所测乘法设置中，从末端删除或一次删除得更快都不如完整方案 [8, 第 3、6.2 节]。

课程图最容易犯的错误，是把文字淡出画成信息已经无损搬到隐状态。图中显示：每个阶段都有一新训练问题，保留下来的语言后缀和答案提供损失；内部如何补偿则是需要学习的未知部分。箭头可以表示训练顺序，不能表示每个词元被逐个储存到了指定神经元。

这种方法在乘法任务上展示了强有力的内化能力，但不同任务存在不同极限。论文在 GSM8K 上仍观察到显式 CoT 的准确率优势；在更难的乘法设置中，完全删去所有步骤并不总是成功，保留部分链的中间检查点可以提供速度与准确率的折中 [8, 第 5、6.1 节]。这里被比较的是不同训练阶段的模型，不能误说成同一模型已经学会按每个测试问题动态选择内化比例。

更快的测试也可能以更长训练为代价。原文局限部分讨论了逐步删除需要反复微调、过快删除可能不收敛的问题 [8, 第 8 节]。因此，若任务会运行很多次，额外训练成本也许可以摊薄；若任务很快变化，重新内化的费用可能改变实际价值。这个部署判断需要工作负载，而不能由单个推理速度比值决定。

3.4 增加连续步骤以后，为什么还需要监督？

Coconut 将课程与新的连续计算位置结合：初始阶段学习普通思维链，随后以若干连续状态替换前面的语言步骤，在剩余文字和答案上施加损失。连续状态可通过反向传播接受学习信号，但原文并没有要求它逐字重建被删掉的句子 [12, 第 3 节]。这允许中间表示服务于未来预测，而不必忠实复制原文表述。

可微分只是梯度能够经过，并不保证梯度足以找到好解。Coconut 的消融比较了没有课程、没有连续思考以及用暂停位置代替连续思考等情况；结果显示课程本身有重要作用，部分合成任务上的高准确率也不能全部归因于连续链 [12, 第 4.3、4.4 节及表 1]。如果只比较完整方法和“直接回答”两个端点，就很难知道收益来自训练支架、增加位置，还是前后状态的连续依赖。

这正引出 CODI 的问题：能否不逐项约束全部连续步骤，而在一个对回答关键的位置监督最终形成的状态？CODI 同时训练共享模型的教师任务和学生任务。教师生成显式 CoT，学生执行连续状态反馈，再在答案前的指定位置对齐两者的多层激活 [20, 第 3.1 至 3.4 节]。

3.5 CODI：在终端状态建立桥梁

CODI 的总目标包含三个部分：

$$\mathcal{L} = \alpha \mathcal{L}_{\text{teacher}} + \beta \mathcal{L}_{\text{student}} + \gamma \mathcal{L}_{\text{KD}}. \quad (3.4)$$

教师项学习语言轨迹和答案，学生项学习在连续思考之后生成答案，对齐项则让学生在答案前关键位置的表示靠近教师。 α, β, γ 是权重，不能理解成三种信号天然等量或彼此独立。原文选择答案提示中冒号位置的各层状态，以停止梯度后的教师状态为目标，并使用按教师激活尺度规范化后的 L1 距离 [20, 第 3.4、3.5 节]。

“停止梯度”只说明对齐损失不沿目标分支反向传播，不意味着整个教师在训练中永远固定。因为两种任务共享模型，教师任务仍通过自己的目标更新参数。这一点与独立预训练、冻结不动的教师不同。原文消融也把独立教师、学生同时学习显式任务、移除 L1 对齐等条件分开比较 [20, 第 4.4 节及表 2]。

图中的两条分支共用权重标记，却保留不同输入。教师分支能看到语言过程，学生分支只有问题和自己产生的连续状态；停止梯度的符号仅放在对齐目标处。若把两支画成独立且冻结的网络，读者会误解这项方法为什么称为 self-distillation。

监督位置也可能携带捷径。CODI 在训练数据处理中去掉最后一个直接得出答案的 CoT 步骤，试图避免教师状态只复制已经出现的最终结果。其消融支持这一处理在所测条件下有益，但不能由此证明所有剩余教师信息都是算法性推理 [20, 第 3.5、4.4 节]。终端状态是一个方便的学习接口，不是一份经证明完备的推理摘要。

该版本实验涉及 GPT-2 与 LLaMA3.2-1B-Instruct、数学与常识数据，以及若干分布外数学集。正文报告的相对优势也具有模型依赖：例如分布外表格中，GPT-2 的 CODI 超过该表的显式基线，而 LLaMA-1B 的显式 CoT 在所列三项上仍更强 [20, 第 4.1 至 4.3 节及表 1]。因此，不能把“某设置接近或超过显式 CoT”改写成连续思考已经普遍无损替代文字思维链。

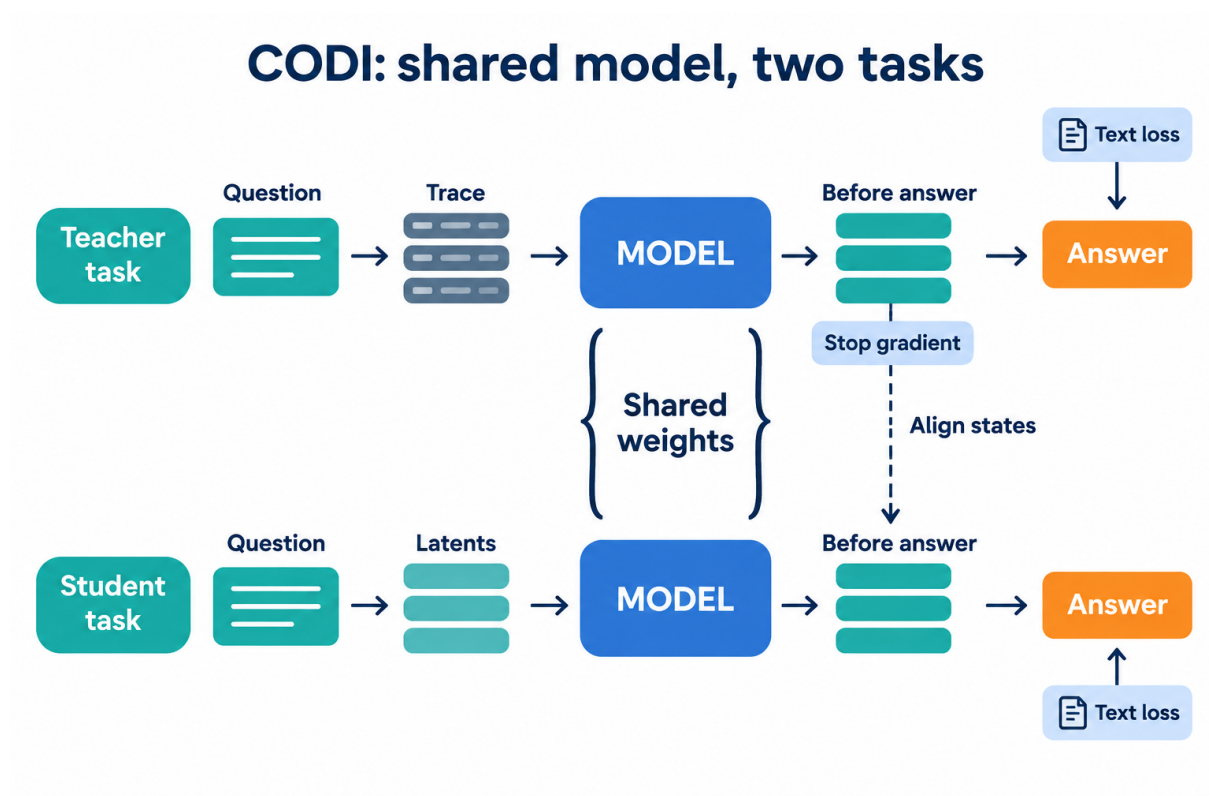


图 3.3: CODI 共享权重的两种任务在答案前状态上对齐。虚线是教师到学生的监督目标, stop-gradient 只作用于该目标。文字损失符号表示监督, 教师实际还监督轨迹, 连续状态逐步生成细节省略。

来源与边界: 概念性教学图; 依据 [20] 及本章证据整理, 不构成新增实验结果。

3.6 三种监督，留下三种待检验的假设

状态蒸馏依赖所选教师表示确实包含对学生有用的信息；逐步内化依赖课程能把任务平稳地转移到没有语言支架的条件；终端对齐依赖选定位置的接近与答案能力之间存在足够强的联系。它们都可能成功，也都可能受目标歧义、优化不稳定和分布变化限制。

本章的综合判断是：内化首先是一个怎样学习内部计算的问题，尚不是一个内部计算已被完整解释的结论。读者可以在每项工作旁写下两个不同的问题：训练究竟提供了什么额外信息？测试时究竟还执行哪些额外步骤？前者帮助我们理解能力从哪里来，后者把我们带到下一章，讨论连续状态怎样真正向前推进。

第四章 连续思维怎样向前推进？

4.1 去掉词汇接口以后，还剩什么？

在通常的自回归生成中，最后隐状态经过输出头变成词汇概率，再由选出的词元查表得到下一步输入。连续反馈删去了这段离散选择。设当前最后隐状态为 h_t ，词嵌入表为 E ，输出矩阵为 W ，则普通生成与一种连续反馈可以分别概括为

$$u_{t+1} \sim \text{softmax}(Wh_t), \quad e_{t+1} = E(u_{t+1}), \quad e_{t+1} = h_t \quad (\text{直接连续反馈}). \quad (4.1)$$

这里 e_{t+1} 是下一位置送入网络的向量，不是已经输出的答案词元。直接连续反馈这一项保留对前一步结果的依赖，却不要求该结果先落到某一个词汇项上。Coconut 的核心接口正是这种反馈；它用特殊标记界定语言模式和潜在模式，连续阶段结束后再恢复普通词元生成 [12, 第 3 节]。

这个改动有两种不同意义。计算上，它让前向过程在连续状态之间串接；表征上，它允许一个中间向量不必等同于某个具体词的嵌入。但“连续”并不等于“无限容量”：向量维度、数值精度、训练分布和后续读出都仍有限。更不能从跳离散选择直接推出模型同时精确执行了所有可能分支。是否形成有用的多路径表示，必须另外检查。

对比图标出词汇选择与重新嵌入发生的位置，并保留最终答案读出；输入问题与历史缓存在图中省略，不能据此理解为计算时不再访问它们。CODI 在反馈中加入投影 MLP 与归一化，而 Coconut 的基本形式直接使用末端隐状态；这说明“连续链”是一类接口选择，并非只有一种数值参数化 [20, 第 3.3 节]。

4.2 Coconut 的实验回答了什么？

Coconut 的课程让连续位置逐渐接替前面的语言步骤，之后在剩余文字上训练。该版本主要实验使用 GPT-2，在 GSM8K 及两个逻辑任务上比较完整 CoT、直接回答、逐步内化和多个消融。论文通常在推理时使用与最后训练阶段相符的固定连续步数；虽然讨论了分类器停机选项，主结果不能被描述为已经实现输入自适应的无限内部思考 [12, 第 3、4.1 至 4.3 节]。

比较结果时应避免只挑最有利的一列。该版本表 1 中，Coconut 在 GSM8K 上好于直接回答，却低于显式 CoT；在合成逻辑任务上，它可以超过显式 CoT，但逐步内化及若干不含连续反馈的变体也表现很强。ProsQA 中逐步内化的结果甚至高于完整 Coconut [12, 表 1 与第 4.4 节]。这说明实验同时触及了训练目标、任务结构与可用计算，不能把合成任务的全部增益都归给连续状态的容量。

原文还探索用更多连续位置替换每个语言步骤。初始小范围内的改善是一个有价值信号，但附录报告增加到更大替换数后出现性能回落和更大方差，并将其与阶段切换的训练不稳定联系起来 [12, 图 3、附录 C]。从这里得到的研究问题是怎样训练一个能稳定利用更长状态链的模型，而不是把“长度可以设大”误当成“能力必然随长度增长”。

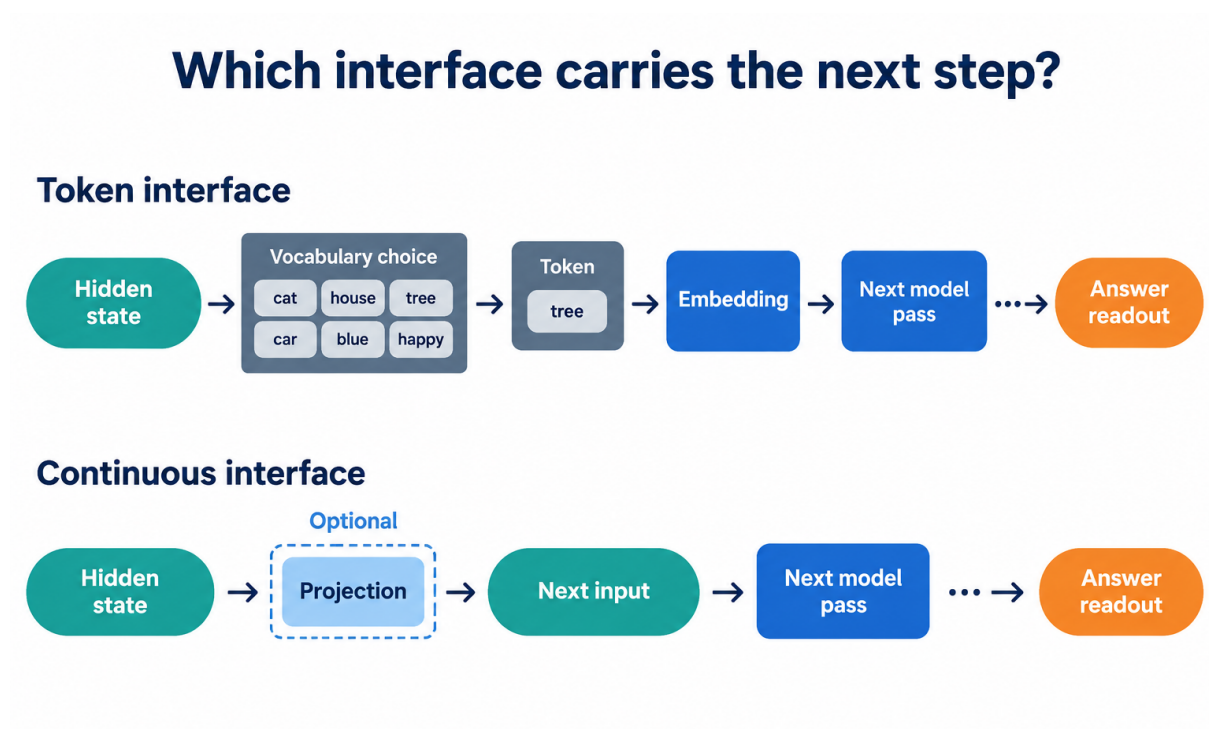


图 4.1: 词汇接口与连续接口的概念对照。连续输入可直接来自上一隐状态，也可经投影；最终答案仍由语言读出。历史缓存和多轮细节省略。

来源与边界：概念性教学图；依据 [12, 20] 及本章证据整理，不构成新增实验结果。

Three meanings of multiple paths

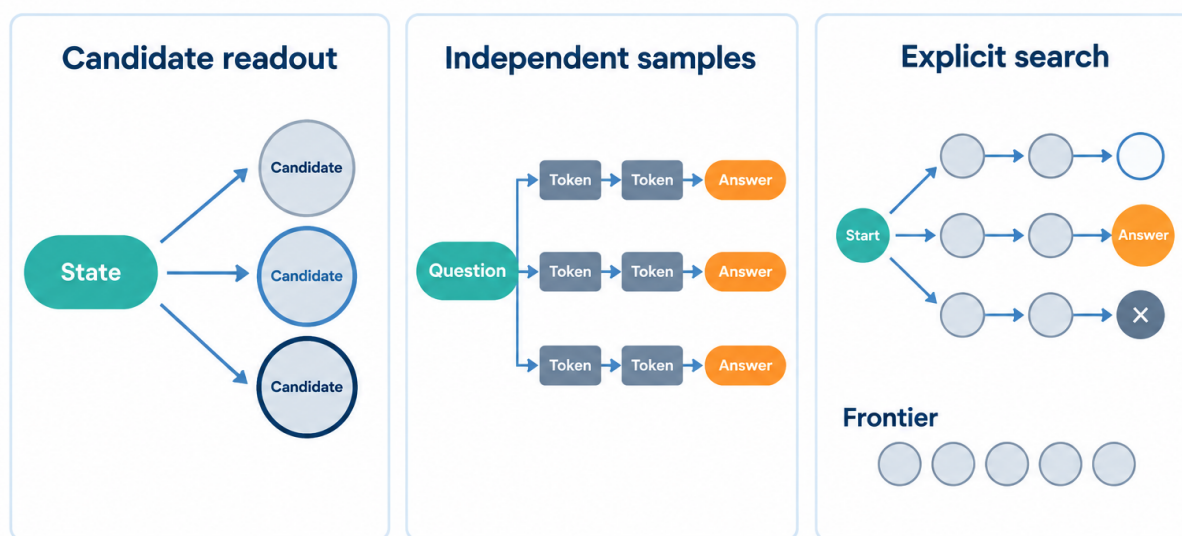


图 4.2: 三种多路径概念: 从一个状态读出候选、实际生成多条独立轨迹、显式维护搜索节点。右下 frontier 仅示意可存储的前沿对象, 不是上图某一时刻的精确队列; 左图不证明标准 BFS。

来源与边界: 概念性教学图; 依据 [12, 24] 及本章证据整理, 不构成新增实验结果。

4.3 潜在搜索: 好的解释需要多强的证据?

Coconut 在 ProsQA 上进一步研究连续阶段切换回语言阶段后的行为。为支持不同切换位置, 分析用模型混合了不同训练阶段的数据; 这一点与只使用最终阶段的基本设置不同。研究者检查最终答案, 也检查输出的部分路径能否由图中的合法路径补全 [12, 第 5.1、5.2 节]。

其核心观察是: 在连续状态之后切回语言输出时, 若干候选节点可以同时具有较高概率, 随后分布更集中于能通向目标的节点。作者把这一模式解释为一种类似搜索的过程 [12, 第 5.3 节及图 7、8]。这种解释有启发性, 因为连续状态确实不必像贪心选择一个语言词元那样过早承诺某个显式分支。

但这里读出的概率仍然是特定探测接口下的行为。它并未列举向量内部保存的全部分支, 也未独立证明模型逐步执行了标准广度优先搜索。原文自己的描述也强调它会偏好和剪除候选, 并非均匀展开所有前沿节点。因此, 本书将“潜在搜索”作为得到部分行为支持的机制解释, 保留其与严格算法等价之间的距离。

本书构造一个无实验数字的教学图: 从起点可以走向多个分支, 其中一条是死路。连续表示也许暂时保留对不同分支的倾向; 显式多次采样会产生多条实际轨迹; 外部树搜索则维护可枚举节点和扩展规则。图中以三个小面板对照这些对象。它们都涉及多个可能性, 却有不同状态占用、分支成本和可审计程度。图中的“也许”不是修辞, 而是证据层级的界线。

进一步验证需要有针对性的干预。例如, 在不改题目其他部分的条件下改变一条边, 检查对应候选的表示与最终行为是否按预期变化; 或干预候选相关子空间, 观察是否选择性影响相应

表 4.1: CCoT 原文的一个局部成本对照。数据仅对应其 LLaMA2-7B-chat、GSM8K 微调设置，不用于跨论文排行榜。

推理形式	答案精确匹配率	平均解码时间（秒）
直接答案	0.089	0.33
CCoT, 保留比例 $r = 0.05$	0.151	0.49
CCoT, 保留比例 $r = 0.10$	0.179	0.78
完整显式链	0.315	8.10

来源与边界: Cheng and Durme [5, 第 4.4、5.1、5.2 节及表 2]; 单张 Nvidia A100 的墙钟测量, 表中未给出这些条目的方差。

路径。这些是本书提出的研究设计方向, 并非上述论文已经完成的实验。第十一章将讨论为什么仅画出注意力或解码概率仍不足以完成因果审计。

4.4 压缩一条已知轨迹, 和发现新的计算不同

CCoT 从另一个问题出发: 完整 CoT 的隐状态序列中, 是否有较小的子集足以帮助解码答案? 它先利用带完整轨迹的训练数据取得目标状态子集, 再训练模块直接预测这些压缩状态, 最后让答案解码模块学会使用它们。实际推理不应先完整生成原链再压缩, 否则生成原链的主要串行代价已经付出 [5, 第 3.3、4 节]。

设原链长度为 m , 保留比例为 r , 目标连续位置数为

$$k = \lceil rm \rceil, \quad 0 < r < 1. \quad (4.2)$$

这个式子描述训练目标的压缩程度, 不是测试时已知正确轨迹长度。原文为实际推理额外训练结束预测器, 并设定长度上限。其连续位置由某个中间层的前一步表示反馈产生, 与总是使用末层状态的基本 Coconut 形式又不同 [5, 第 4.1 至 4.4 节]。

这里有一组特别值得区分的实验对象: 从真实完整轨迹取得子集后解码, 与从问题本身预测子集后解码。前者若能保留性能, 说明存在有用压缩; 后者还要解决怎样产生压缩表示。不能把前者的无损性当成后者已经无损的证据。CCoT 的实验恰好让这种差距可见。

这组数字说明的不是某种普适压缩倍数, 而是该设置下的一条折中: 压缩状态比直接回答更准确, 比完整链快, 但仍明显损失准确率。原文还观察到增大保留比例后的收益会趋于平台, 并讨论反馈近似误差的可能影响 [5, 第 6.1 节]。这些结果提醒我们, 能够表示压缩状态、能够学会预测它、以及能够在多步反馈中稳定使用它, 是三个不同问题。

4.5 软思维并不总是“一条更短的链”

SoftCoT++ 展示了另一种连续表征用途: 助手模型读取问题和若干占位词元, 将相应隐状态经投影送入主推理模型, 主模型随后仍生成显式推理文字。该工作描述的 SoftCoT 基础设置固定助手和主模型, 只训练投影。因而这里的连续向量是引导后续推理的软提示, 不能仅凭名称把它视为 Coconut 那种逐步末状态反馈 [24, 第 3.1、3.2 节]。

连续表征若对同一输入是确定的, 不能直接像词汇概率分布那样反复采样。SoftCoT++ 因

Two places to create diversity

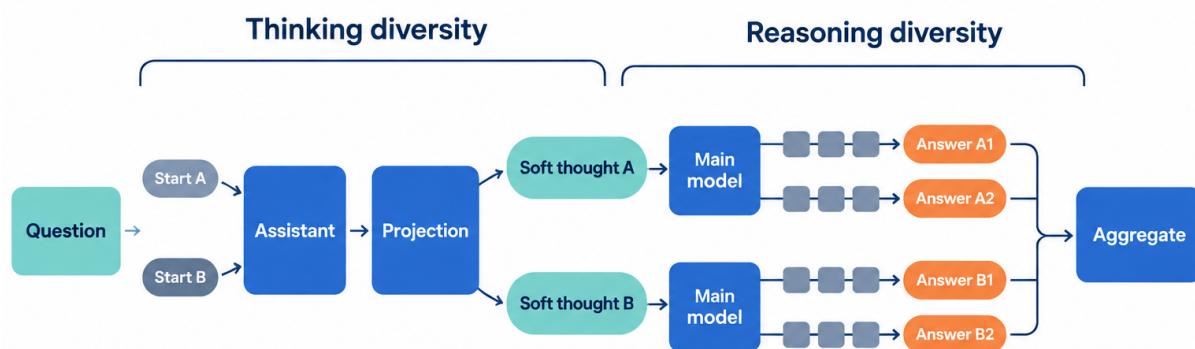


图 4.3: 软思维构造与后续离散推理是两个分支位置。各起点均与同一问题一起输入助手，图中问题连接简写；两处 Main model 复用相同权重。分支数量仅作示意，均需计算，最后聚合答案。

来源与边界：概念性教学图；依据 [24] 及本章证据整理，不构成新增实验结果。

此用不同初始化特殊词元产生多个软思维表示，并加入鼓励差异的对比目标。其研究重点是增加不同思维起点的数量，而不是只把同一条连续递推运行得更长 [24, 第 3.3 至 3.5 节]。

两层分叉图可以清楚表达这种区别：第一层在连续提示的构造处产生多个候选，第二层在每个候选之后采样离散语言推理。合并答案时还需要聚合规则。原文表 1 按十条链作比较，表 2 进一步组合两层扩展；这些不是一次前向凭空获得多个免费答案 [24, 第 4 节、第 5.1 至 5.3 节]。

比较这类方法的收益时，仅固定“链数”也未必固定总成本。助手模型、投影、主模型预填充、后续生成长度和能否批处理，都进入实际预算。原文在两个约八十亿参数的主模型系列和若干推理任务中报告改进，但其局限仍包括固定规模模型和对软思维分布的初步探索 [24, 第 4.2、5.4 节]。本书据此讨论一种多路径机制，不把这些结果推广成“连续空间多样性总能提升准确率”的定律。

4.6 连续接口打开了选择，也留下了预算问题

本章的四个问题可以重新串起来。Coconut 说明怎样让下一步直接依赖前一步的连续结果；潜在搜索分析追问这些结果可能保留怎样的候选信息；CCoT 研究怎样预测较短的有用轨迹表征；SoftCoT++ 则把连续表示当作多条后续推理的不同起点。它们共享连续向量这个载体，却分别改变串行更新、监督目标和分支结构。

因此，下次遇到“软思维”“连续推理”或“压缩思维链”，可以先问：这一个向量来自哪里？下一步究竟重复了哪个算子？中间结果是被教师规定、由答案目标间接塑造，还是作为多样化提

示使用？然后再问，需要多少次前向、保留多少缓存、怎样停止、怎样选出最终答案。回答完这些问题，方法之间的联系才会比名称更清楚。

连续链通过增加序列中的计算步骤来获得内部演化。另一条路线不新增同样意义上的序列位置，而是反复使用深度中的模块。下一章将从迭代算法与参数共享的来路展开，讨论为什么一个重复的算子可能学会逐步解决问题，以及为什么“可以重复运行”仍远远不等于“已经学会可靠迭代”。

第五章 循环深度的来路：让同一条规则再运行一次

5.1 输入有多长，并不决定需要想多久

考虑把两个二进制数相加。输入长度给出了数字的数量，却没有直接指定计算怎样展开：网络可以一次建立所有位之间的关系，也可以逐步传播进位。后者重复使用相似的局部规则，因此很自然地引出一个问题：能否让模型学习一个可重复执行的更新器，而不是为每一轮计算配置一套新参数？这里的关键不是拟人化的“思考”，而是如何把计算预算从参数数量和输入长度中分离出来。

普通序列 RNN 在读到新输入时更新状态。深度循环则允许输入暂时不变，让表示经历下一轮更新。两种循环可以同时存在。用 t 表示输入位置、 r 表示内部计算轮次，则 $s_{t,r}$ 有两个索引；删去其中任意一个，都会掩盖模型实际在重复什么。这也是阅读 ACT 与 Universal Transformer 时最先需要建立的坐标系。ACT 原文明确把一个输入步拆成若干内部更新；UT 则把递归放在整段序列表示的修订方向上 [6, 11]。

图中左右对照新输入推进与同一输入内的重复计算。对一个位置继续计算，不等于读入了新词；对所有位置并行更新，也不等于各轮之间能够并行。读者随后遇到的“时间”“步数”和“深度”，都应先放回这两个坐标，再讨论速度。

5.2 ACT：把停机纳入可学习系统

Adaptive Computation Time 的出发点是：同一个 RNN 可以在输出之前，多做几次状态变换。内部第一个更新读取上一输入步传来的状态，其余更新读取本步上一轮状态；输入附加一个二值标记，区分“新输入到来”和“仍在处理同一输入”。所有内部步骤共享状态转移与输出参数 [11, 第 2 节]。

停机单元为每轮给出 $a_{t,r} \in (0, 1)$ 。这里采用本书符号，略去输入位置下标时，ACT 的核心可写成

$$N = \min \left\{ n : \sum_{r=1}^n a_r \geq 1 - \epsilon \right\}, \quad R = 1 - \sum_{r=1}^{N-1} a_r, \quad (5.1)$$

$$p_r = \begin{cases} a_r, & r < N, \\ R, & r = N, \end{cases} \quad \bar{s} = \sum_{r=1}^N p_r s_r. \quad (5.2)$$

最后一轮使用剩余质量 R ，保证权重相加为一。原方法还以相同权重混合输出，而不是每次随机抽取某一轮。直觉上，网络既学习怎样更新，也学习哪些内部状态应当对本次输出贡献更多。ACT 加入与 $N + R$ 有关的 ponder cost；这个成本的系数决定准确率和计算量之间的取舍 [11, 式 (5)–(10)、第 2.1 节]。

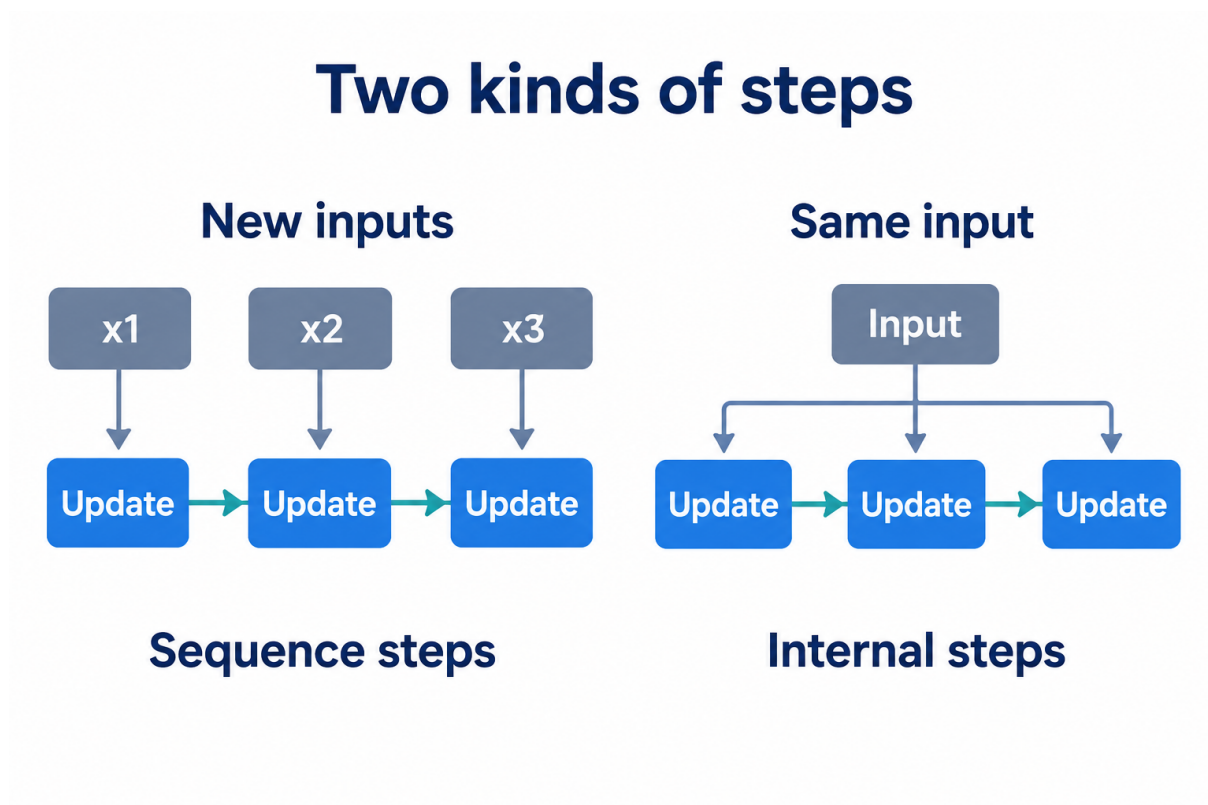


图 5.1: 两类步数的概念对照：左侧每次更新读取新输入，右侧对同一输入连续更新。相同蓝色表示共享更新规则；图中不表示速度或准确率比较。

来源与边界：概念性教学图；依据 [6, 11] 及本章证据整理，不构成新增实验结果。

混合状态是一项建模选择，不能直接理解成“多个离散推理答案投票”。状态向量的线性组合未必对应离散算法中合法的中间配置，原文也讨论了这种近似线性假设。另一项容易漏掉的限制是，停机目标中的步数并非实测毫秒：每一步代价相近、执行能及时退出时，它才可能近似计算开销。ACT 在奇偶性、逻辑运算、加法和排序等合成任务上展示了作用，而在字符级 Wikipedia 语言建模上没有获得同等幅度的性能改善 [11, 第 3 节]。该语言实验按原文第 3.5 节报告的是训练集误差，未使用验证集早停，因而也不能作为留出集泛化收益的证据。这个早期结果已经提醒我们：学会分配计算与在开放语言上获得可靠收益，是两个问题。

5.3 UT：保留全局通信，重复修订整段表示

Universal Transformer 将自注意力的全局信息交换与共享深度结合起来。输入 $x_1, \dots, x_n$ 首先得到嵌入矩阵 $H^0 \in \mathbb{R}^{n \times d}$ 。在第 r 轮，自注意力让各位置读取上一轮的表示，再由共享变换更新各位置。省略残差和归一化细节，可概括为

$$H^{r+1} = F_\theta(H^r + P^{r+1}), \quad (5.3)$$

其中 P^{r+1} 同时编码序列位置与更新轮次。原文给出了带残差、LayerNorm、自注意力以及逐位置前馈或可分离卷积的完整方程。参数共享不意味着每轮接收同样的信息：状态改变了，轮次编码也改变了 [6, 第 2.1 节、式 (4)–(7)]。

UT 的“parallel-in-time”应当谨慎理解：在一个修订轮次内，输入位置可并行处理；第 $r+1$ 轮仍依赖第 r 轮。解码器还保留自回归因果约束，并读取编码器的最终表示。模型因此并没有消除生成下一个词的顺序依赖。其动态版本把 ACT 用到每个位置；停止更新的位置保留状态，其他位置继续修订 [6, 第 2.1–2.2 节]。

教学图展示三列状态矩阵和两个相同颜色的更新块；颜色相同表示参数共享，位置间注意力位于更新块内部。这样才能看清“同一组位置被更新”与第四章“新增连续位置”的区别。

UT 的实验覆盖算法任务、bAbI、LAMBADA 与机器翻译，但收益并非处处来自自适应停机。原文报告动态停机帮助若干结构化任务，却使机器翻译结果略有下降 [6, 第 1、3 节]。因此可以把它看作“深度共享加上可选动态计算”的研究起点，不能把全部改进归因于停机机制。关于计算普适性的讨论还带有资源与模型假设；第九章将单独解释为什么表达能力不等于训练可达性。

5.4 共享权重为什么还不等于学到算法

共享参数给不同深度施加了共同约束，但共享网络仍可能只在训练深度的终点读出正确答案。它可以把轮次编码当作阶段编号，也可以在隐状态中保留一个内部计数器，形成“第六轮才能读”的计算。这样做仍可能是有效的固定深度网络，却未必允许第七轮、第八轮继续改善。

Subformer 提供了一个很有用的对照。它把中间层做 sandwich 式共享，保留首尾独立层，并结合较小词嵌入与自注意力投影来提高参数效率。论文比较不同共享方案，在翻译、摘要和语言建模上检验表现 [19, 第 2–3 节]。这里研究的是有限架构中的共享位置、宽度与参数预算；由此不能推出测试时任意增加共享层数就会形成可靠的推理延伸。参数压缩与可外推计算可以相遇，但逻辑上互不包含。

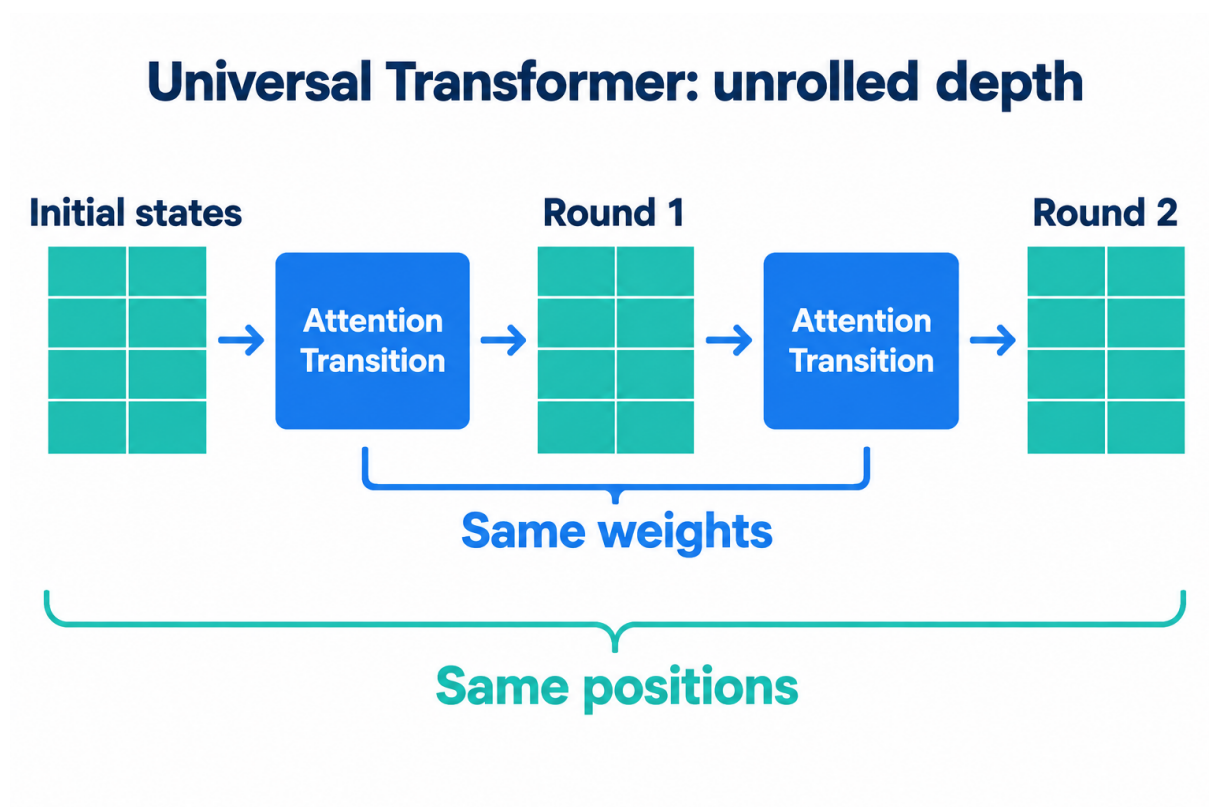


图 5.2: Universal Transformer 的简化展开：同一组位置经过共享的注意力与变换块反复修订。图中省略位置和轮次编码、残差与归一化；不表示各轮能并行执行。

来源与边界：概念性教学图；依据 [6] 及本章证据整理，不构成新增实验结果。

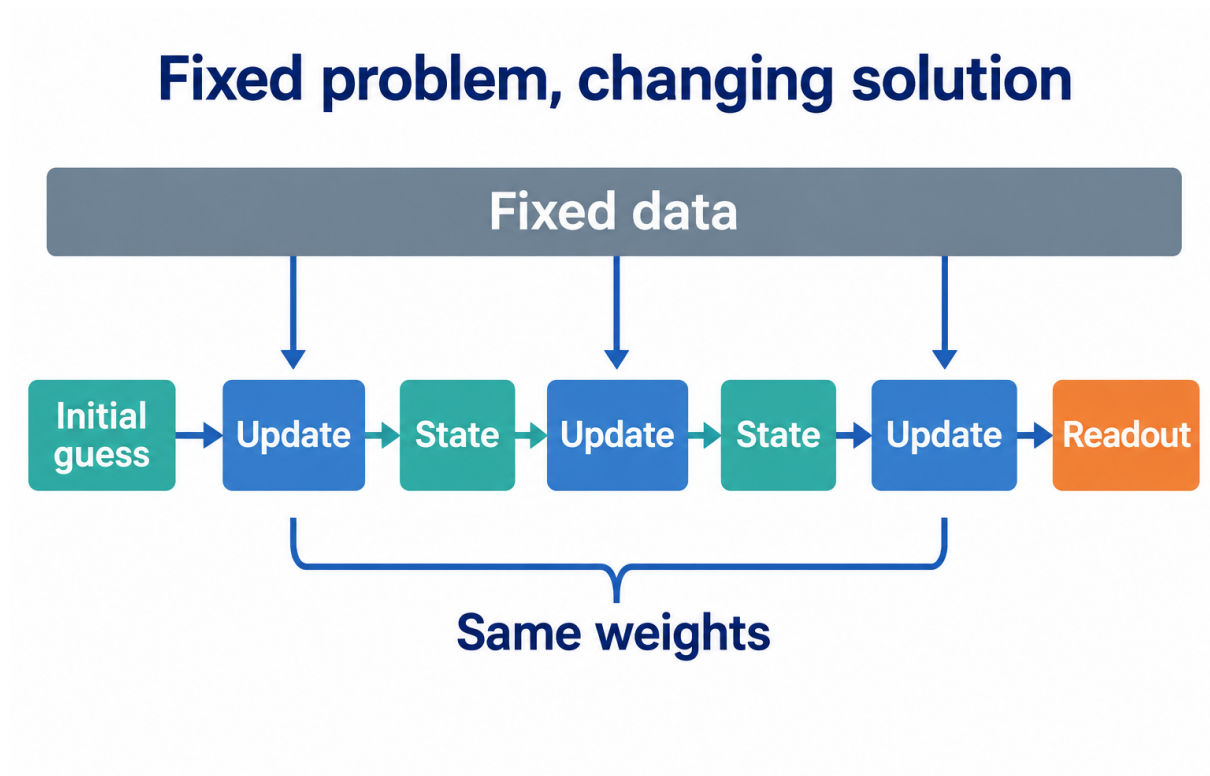


图 5.3: 固定数据与变化状态分开：每轮更新都能读取同一个问题条件，同时修改当前候选状态。此图是迭代算法的解释性类比，不证明模型执行了梯度下降。

来源与边界：概念性教学图；依据 [25] 及本章证据整理，不构成新增实验结果。

为了理解更强的迭代偏置，可看一个明确的算法模板：最小化线性回归平方误差时，梯度下降反复计算

$$w_{r+1} = w_r - \eta X^\top (X w_r - y). \quad (5.4)$$

每一步都需要当前解 w_r 和不变的数据 (X, y) 。在适当步长和凸二次目标条件下，这个例子有明确收敛性质；它在此只解释架构动机，并不证明 Transformer 内部真的执行了该式。Yang et al. [25] 将类似思路用于上下文学习实验：模型从函数采样得到的示例预测新点，使用共享块、重复输入注入和覆盖后段迭代的训练损失。原文研究线性函数、稀疏线性函数、决策树及两层网络生成的数据，不是开放域问答。

该方法从零状态出发，把提示表示每轮重新加入状态。原文的线性回归消融显示，单纯把输入作为初值的权重绑定版本，在训练轮次之外更容易恶化；重复注入版本则表现出更稳定的迭代行为 [25, 第 4 节、图 3]。这给出一个可检验的机制假说：把问题条件留在独立通道中，可能有助于更新器反复求解同一个问题。然而它不是普遍定理。无输入注入的网络也可能通过自身状态保留输入；注入输入的网络也可能学到错误更新。

图中固定数据通道与可变候选解通道保持分离，展开时固定数据每轮进入共享更新器。这是解释性类比，不是论文对真实网络内部算法的确认。

5.5 从“复用层”走向“训练更新规则”

另一条相邻路线是 Deep Equilibrium Models：它把输出隐状态定义为 $z^* = f_\theta(z^*, x)$ 的解，使用求根器及隐式微分，而非保存任意长的显式展开轨迹 [2, 第 1、3 节]。它强调最终平衡点；有限深度循环模型则可能利用尚未收敛的中间状态。两者共享输入条件化的更新形式，却不能因为都出现固定点方程，就视作同一种训练方法。

5.6 用一个极小例子检查“迭代”是否成立

不妨把共享模块想成一个反复修改候选解的编辑器。对于输入“查找从节点甲到节点丁的路径”，第零轮状态可以只编码起点；后续轮次可能扩展可达集合，也可能只调整最终分类器所需的特征。这两种行为在训练集准确率上可能相近，但对更长路径的要求不同。前者需要让更新规则适用于更大范围，后者只需在常见图形上形成可分表示。这个例子是本书构造的诊断场景，并不描述某个模型已经学到的算法。

要区分二者，可以同时改变最短路径长度和干扰分支数，而不要只增加总输入长度。增加干扰节点主要改变需要选择的信息；增加最短路径长度才直接增加某种局部传播算法需要的轮数。若模型在较长输入上失败，我们由此能进一步问，它是无法选择信息，还是缺少迭代预算。这样的任务分解延续了上下文函数学习研究将函数类别、样本数量和循环轮次分别改变的思路，但属于本书提出的实验设计 [25, 第 3-5 节]。

还可以从某个中间状态重新启动同一个更新器。如果它只有在初始状态连续运行精确次数后才成功，而从合理的一部分状态继续运行就失败，“学到可重复规则”的解释便需要收窄。反之，即使这一测试通过，也只能说明在所构造的状态范围内具有可继续性，不能证明任意状态都会收敛。这种审慎并不是降低循环模型的研究价值；它把模糊的算法直觉变成可以被反驳的命题。

最后要避免一个常见的对照错误：把一个较窄的共享块循环十次，与同宽的十套独立块比较，然后将参数较少和运行较快同时归为胜利。前者大致执行十次块计算，后者也是十次；真正改变的是权重存储和参数自由度。若要测试同参数预算，可以让独立模型变窄；若要测试同算力预算，可以匹配执行层数和宽度。两种匹配回答不同问题，不能互相替代。这也解释了为什么本章把“共享”“迭代”和“效率”分开命名。

至此，问题从“能否少存几套参数”推进为“怎样训练一个在不同运行长度上仍有用的状态转移”。至少需要同时检查初始化、输入保留、轮次信息、读出位置和训练覆盖的深度。若只报告参数量减少，就没有回答模型是否学会继续计算；若只展示一个稳定固定点，也没有回答固定点是否正确。下一章把这些要求带进语言预训练：当任务不再给定一套可监督的算法步骤，下一词损失是否足以教会共享模块分配和使用更多计算？

第六章 从循环更新器到预训练语言模型

6.1 下一词损失怎样监督不可见的计算

上一章的算法任务有清楚的问题边界：给出一串数，预测一个结果。语言预训练面对的则是长序列中的大量下一词预测；某些位置依赖局部搭配，某些位置可能需要组合多个事实。如果每个位置都运行同样深的独立层，简单位置也要支付全部计算。如果把中间层共享，参数量可以下降，但共享块还需要学习在早期状态和后期状态上都做有用的工作。

这使 recurrent-depth pretraining 的核心不只是“把 Transformer 接成环”。训练分布必须包含不同成熟程度的隐状态，读出头必须解释它们，优化过程还必须能处理重复更新产生的长依赖链。更深的前向计算、需要保存的训练激活和实际反向传播长度可以分别控制，不能混成一个“深度”数字。Huginn 与 Ouro 为这些问题提供了两种具体答案，下面将按其数据流而不是共同标签比较 [9, 28]。

6.2 Huginn：前奏、共享核心与终曲

Huginn 的 recurrent-depth 架构把语言模型分成 prelude、recurrent core 和 coda。用本书统一符号，它的主干是

$$\begin{aligned} e &= P(x), & s_0 &\sim \mathcal{N}(0, \sigma^2 I), \\ s_r &= R_\theta(e, s_{r-1}), & p &= C(s_K). \end{aligned} \tag{6.1}$$

前奏 P 把 token 序列映射为固定条件 e ，共享核心反复更新 s_r ，终曲 C 将最终状态变成词分布。这里特意用 C 表示读出：所读 v1 全文第 3.1 节公式末行的符号与其文字说明不一致，本书依照同节文字和架构图解释为 coda，而不沿用该处疑似笔误 [9, 第 3.1 节、图 2]。

核心入口把 e 和当前状态拼接，再线性映射回隐层宽度。因此输入不是只在第一轮出现，而是每轮直接可用。随机初态是训练设计的一部分，不是生成文本时的采样温度。上式沿用第 3.1 节的宏观正态分布记法；大型运行的第 4.1 节进一步指定了截断正态初态，复现时应使用该具体初始化配置。原文的动机是降低对特定初态与运行路径的依赖，但“设计希望获得路径独立”与“所有输入都已证明收敛到唯一解”应当分开。实际模型的因果自注意力、门控 MLP 和归一化共同构成非线性更新器，不能直接套用凸优化收敛定理 [9, 第 3.1–3.2、4.1 节]。

展开图把前奏输出作为固定条件，分别注入两个共享核心轮次；变化的状态沿主线传递，选定终点接入终曲。独立随机初态与固定输入条件不同；两轮仅供教学，不代表实际模型的循环次数。

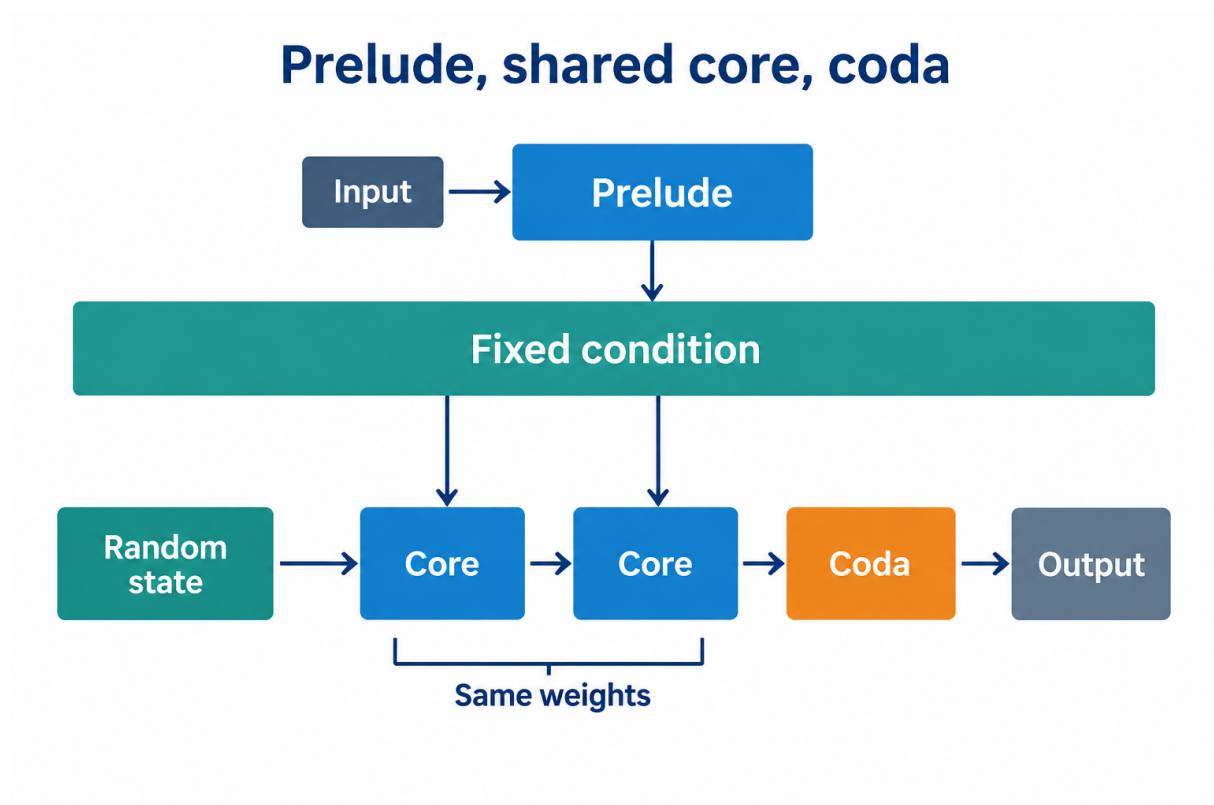


图 6.1: Huginn 宏观结构的简化展开：前奏产生固定条件，每轮共享核心同时接收固定条件和变化状态，终曲在选定终点读出。图示两轮仅用于解释数据流，不代表实际训练或测试轮数。

来源与边界：概念性教学图；依据 [9] 及本章证据整理，不构成新增实验结果。

Forward work and gradient window

Sample depth

Short depth

Long depth

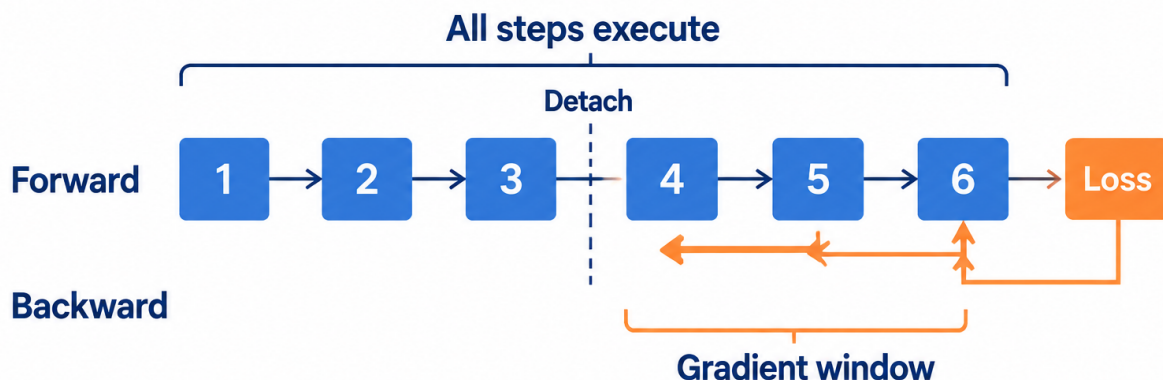


图 6.2: 前向计算与梯度窗口的概念示意: 所有已选步骤都执行前向更新, 截断反向传播只覆盖后段。图中六步与三步窗口仅为教学例子, 不是 Huginn 的实际超参数; 正文所述主实验反向窗口为八轮。

来源与边界: 概念性教学图; 依据 [9] 及本章证据整理, 不构成新增实验结果。

大型实验的层数配置为 $(l_P, l_R, l_C) = (2, 4, 2)$, 隐藏宽度为 5280, 约 35 亿参数; 循环 32 次时, 执行路径包含 $2 + 4 \times 32 + 2 = 132$ 层变换 [9, 第 3.2、4.1 节]。这是有效执行深度, 不是 132 套独立参数。它说明参数存储与计算量能够分开, 也同时说明“只有 8 个实际层”并不意味着推理便宜。一个请求仍要顺序运行 32 个核心块。

6.3 训练深度是一种数据分布

Huginn 对训练轮次采样, 优化下一词损失在输入与轮次上的期望:

$$\mathcal{L}(\theta) = \mathbb{E}_x \mathbb{E}_{K \sim \Lambda} [-\log p_\theta(x_{t+1} \mid x_{\leq t}, K)]. \quad (6.2)$$

论文采用 log-normal Poisson 分布, 既经常抽到较浅计算, 也偶尔抽到较长展开。大型运行中, 为了减少分布式设备互相等待, 每个 micro-batch 采用跨 worker 同步的同一个采样轮次。数学目标中的逐样本期望, 与工程实现中的同步采样粒度, 是两个需要分别记录的层次 [9, 第 3.3、4.1 节]。

更关键的是, 前向运行 K 次不等于反向穿过全部 K 次。主实验把反向窗口设为最多最后 8 轮 (采样轮数不足 8 时只覆盖实际运行的轮次); 较早的前向状态仍影响结果, 但其历史链在梯度计算中被截断。前奏仍能通过后段每轮的输入注入得到梯度。这个设计限制了激活保存和反向计算负担, 却没有把长前向运行变成免费计算 [9, 第 3.3 节]。如果复现实验只写“训练 32 轮”, 读者无法判断它究竟是固定 32 轮、均值附近 32 轮, 还是只对其中一段求导。

教学图分为深度选择和展开路径两部分；上方标记表示可选深度，不是概率分布曲线。下方六步前向与末三步梯度窗口仅为教学示例，不替代正文所述 Huginn 实验末八轮反传设置。未求导的前向段仍实际执行。

原文的大型训练是一个受预算限制的主要运行，实际预训练量报告为 795B tokens，标题性表述常取整为 800B；数据混合包含网页、代码、数学及指令数据。单次核心前向基线另行训练 180B tokens[9, 第 4.1–4.2 节]。因此“没有专门构造潜在推理轨迹监督”是合适的说法，“完全没有接触指令或推理相关数据”则不准确。同样，比较这两个模型也不是固定训练 token 数的纯架构消融。循环模型展示了可随推理轮次改变的行为，但不能据此独立识别架构、数据组成与训练预算的全部贡献。

6.4 Ouro：多轮读出与退出分布一起训练

Ouro 的基本定义是对 token 嵌入反复应用同一组 Transformer 层，再使用语言模型头读出。按其第 3.1 节定义，初态由嵌入给定；它没有采用式(6.1)中随机状态加独立前奏条件的同一宏观结构。读论文时不宜把 Huginn 的输入注入、初始化和终曲配置自动移植到所有 LoopLM[28, 第 3.1 节]。

Ouro 为不同轮次构造退出概率 $q_\phi(r | x)$ ，将各轮语言模型损失按退出概率加权，并减去退出分布的熵正则项：

$$\mathcal{L} = \sum_{r=1}^{K_{\max}} q_\phi(r | x) \mathcal{L}_r - \beta H(q_\phi(\cdot | x)). \quad (6.3)$$

较高熵鼓励训练关注多个深度，避免过早把概率质量集中在少数轮次。它与“每次随机抽一个深度，只监督这个终点”的训练估计器不同，也与 ACT 对状态做加权平均不同：对损失加权不等于先把状态平均再计算损失 [28, 第 3.2–3.3 节]。

模型族包含 1.4B 和 2.6B 参数版本，论文报告约 7.7T tokens 的多阶段训练，并另外区分基础模型与经过推理 SFT 的 Ouro-Thinking[28, 第 4 节]。把前者和后者混在一条“循环效果”结论中，会把后训练监督也算给架构。与其他开源模型的基准比较同样包含数据和训练 token 数差异；这些结果说明一种完整训练配方具有竞争力，不能替代严格等预算的架构因果对照。

这里有一项应公开保留的来源冲突：v1 第 5.3 节把 5–8 轮称作超出最大训练 4 轮的外推，但第 4 节训练配方表 5 列出前期 8 轮、后期 4 轮。因而本书使用“相对于该节采用的 4 轮评估基准”描述外推区间，不断言模型从未训练过 8 轮。精确复现必须进一步核对发布代码、阶段 checkpoint 和实际运行配置。这类冲突不影响读懂共享结构，却直接影响关于未见深度泛化的结论。

6.5 共享参数之后，缓存仍然是问题

注意力层的权重相同，不意味着输入相同。在不同循环深度，同一个历史 token 的表示已经变化，投影出来的 key/value 也可能不同。因此朴素实现仍需按深度保留缓存。Ouro 专门区分 prefill 与自回归 decoding 的缓存复用实验；其第 5.4.2 节报告，复用哪一轮的缓存会显著影响结果，不能把参数共享直接翻译成缓存等比例减少 [28]。

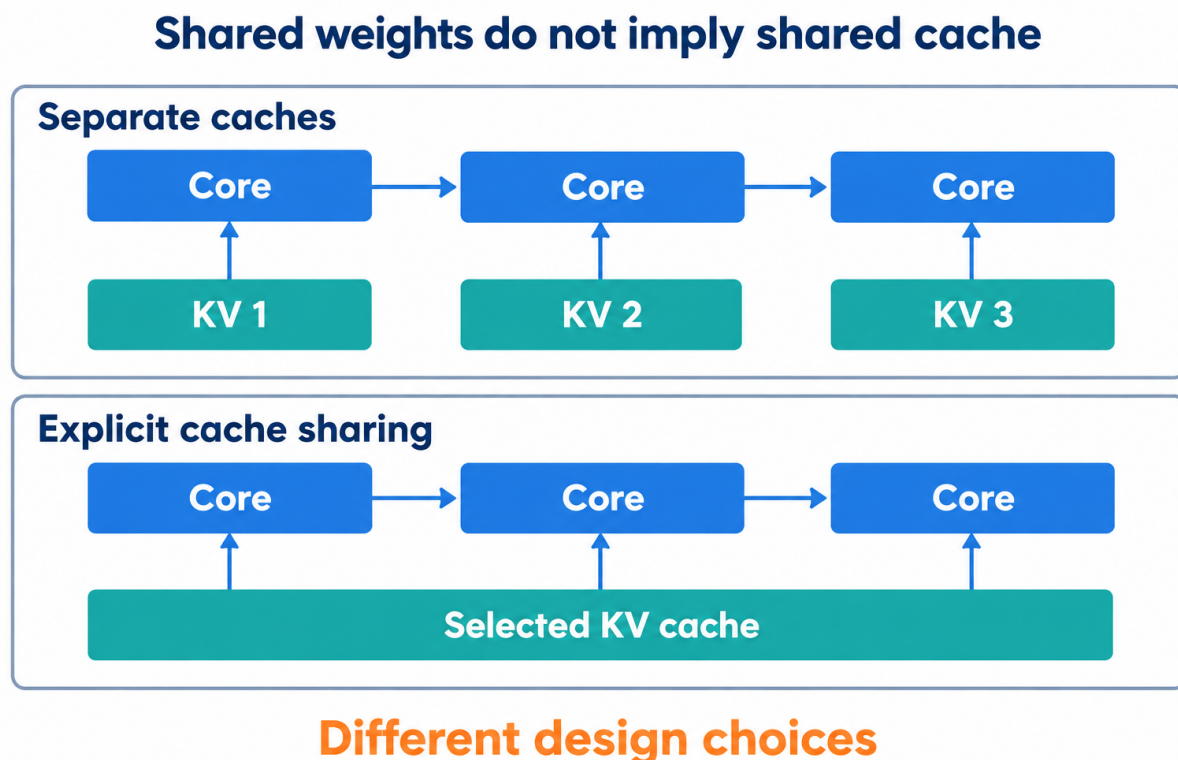


图 6.3: 共享权重不等于共享缓存：上图按深度保存并读取不同 KV，下图显式复用选定缓存。两者是不同设计选择，可能改变可见信息与性能；箭头表示读取缓存，不表示缓存生成过程。

来源与边界：概念性教学图；依据 [1, 28] 及本章证据整理，不构成新增实验结果。

Mixture-of-Recursions 更进一步，把每个 token 执行多少轮与缓存可见性一起设计。expert-choice 逐层选择继续运行的 token，后续候选是前一轮存活集合的子集；token-choice 则在开始时为 token 指定递归深度。它分别研究按递归保存活跃 token 的 KV，以及共享首轮 KV 的方案 [1, 第 2.2 节、图 2]。此外，expert-choice 的全序列 top- k 选择可能使用未来位置的信息；论文采用辅助路由或辅助损失来处理训练与因果推理之间的差异。token-choice 避开这种选择泄漏，却需要处理负载不均衡，二者不能只按平均轮数互换 [1, 第 2.2.1 节、表 2]。这不是把一个独立停机按钮加到原模型上：token 退出以后，其他位置还能读取什么，已经成为模型计算语义的一部分。

图中上下对照“相同权重、不同深度缓存”和“明确规定的跨深度缓存共享”。后者只复用指定缓存，不是无条件等价替换；从 KV 指向核心的箭头表示读取缓存，不表示构建缓存。第十章会进一步区分权重、KV、训练激活与真实延迟账本。

6.6 怎样从训练配方判断模型学到了什么

设有两个都标注“32 轮”的训练运行。运行甲始终在第 32 轮计算损失；运行乙从一组轮数中采样，平均值接近 32。若二者在第 32 轮测试接近，而运行乙在较浅轮数更好，最直接的解释是乙接受了更多浅层读出训练。若乙在较深轮数更好，还需要判断这些轮数是否落在采样长尾中。所谓测试时“增加计算”，可能只是转向训练中不常见但确实见过的预算，未必是严格的深度外推。这是由式(6.2)产生的评价要求，不是对任何一种采样方案的否定。

因此复现表最好给出训练轮数分布的实际直方图，而不只给目标均值；同时记录梯度窗口、最大前向轮数与截断边界。理论上非零概率的深度，可能在有限训练中从未抽到；反过来，一个看似超出常用深度的测试点，可能已经被长尾采样覆盖。原文给出分布定义是必要条件，实际运行日志则能进一步确定经验覆盖。对大型训练只能完成一次的研究，还需要承认随机种子和数据混合未被充分消融，而不是以完整配方的成功代替对每个设计项的独立证明 [9, 第 3.3、4 节]。

读出位置也会影响“中间轮次有用”的结论。若每个深度有专用输出头，模型可以把同一信息放在不同坐标中；若多个深度共用输出头，则必须兼顾表示在不同阶段的可读性。Ouro 的多深度损失明确训练了这些中间读出，而只监督单一末轮的模型没有同样的直接约束 [28, 第 3.1–3.3 节]。所以，某模型在浅层读出较差，并不充分证明其浅层没有任务信息，也可能是信息尚未对齐最终读出。

在工程上还应把训练吞吐和单请求延迟分开。重复使用参数有机会摊薄参数加载或分布式通信成本；但单个状态从第一轮走到第三十二轮仍有串行依赖。大批量训练可能因此更容易保持设备忙碌，而交互式单样本生成未必同比例加速。Huginn 的同步深度采样与 MoR 的跨深度批处理，分别从训练和推理调度侧处理这种差异 [9, 第 4.1 节][1, 第 2.1 节]。架构的优势必须放回执行场景，才有可比较的意义。

从这些模型能得到的稳健结论是：预训练目标可以直接监督循环后的下一词预测，循环内部不必逐轮产生可读文字；但深度分布、梯度窗口、读出和缓存约定必须共同决定。下一章回到另一条路线：如果连续状态不是留在相同位置反复更新，而是反馈成下一次输入，这两类计算在哪些方面相同，又在哪些方面仍然不可互换？

第七章 两条路线的交汇：同是连续状态，计算图却不同

7.1 共同问题不代表同一种答案

隐状态反馈与循环深度都允许模型在给出可见答案之前执行额外计算。这个共同目标容易让人把它们画成同一个环：模型输出一个向量，再把向量送回模型。然而，一个箭头隐藏了至少三个问题：向量写入哪个位置，过去的向量是否仍可访问，下一步调用的是整个模型还是共享核心？本章将这些问题展开。以下比较是本书根据计算定义所作的综合，不宣称相关论文存在未经原文确认的直接继承关系。

以 Coconut 为例，潜在阶段把前一个位置的最终隐状态作为下一个位置的输入嵌入。模型没有先把它离散化成一个词，再查词嵌入表；但序列上仍多出一个计算位置，先前位置仍属于可注意的历史 [12, 第 3 节]。Huginn 则在固定的一组位置上反复更新状态矩阵，完成选定轮次后再通过终曲生成词分布 [9, 第 3 节]。两者都使用连续值，区别首先发生在状态的地址与生命周期，而不是向量是否叫“思维”。

7.2 把“状态”从一个向量扩展为可访问的记忆

对追加连续位置的模型，可把一步记为

$$(u_{j+1}, \mathcal{K}_{j+1}) = G_\theta(u_j, \mathcal{K}_j; x), \quad (7.1)$$

u_j 是进入本次计算的连续嵌入， $\mathcal{K}_j$ 表示既有历史的 KV 等缓存。这个式子是解释性抽象，不是某篇论文的逐层实现公式。它强调：若只把 u_j 当作完整状态，就会漏掉网络仍能读取的过去。Coconut 原文使用 KV cache 避免重复计算已处理前缀，并指出连续步骤仍具有顺序依赖 [12, 第 3 节]。

固定位置上的深度循环则可抽象为

$$S_{r+1} = F_\theta(S_r; E, C_r), \quad (7.2)$$

其中 E 可表示反复注入的输入条件， C_r 表示该深度可访问的历史缓存。不是所有模型都具有独立的 E 通道，也不是所有模型都共享 C_r 。对于整段 prefill， S_r 可以是整个序列的状态矩阵；对于增量解码，当前 token 的更新依赖历史 token 的缓存。这一抽象既容纳第六章的结构，也提醒读者不要把推理时的单 token 更新画成每轮重新计算整个前缀。

上下对照图分别展示新增连续位置并保留旧位置，以及同一组位置跨深度更新。色彩是概念标记，不量化激活，也不表示相同内存、注意力可见性或单步成本。

一个具体例子有助于理解：若问题需要记住三个中间量，追加位置的模型可以让它们在不同历史槽位留下可读取的表示；固定位置循环模型则可以更新原有槽位，在当前状态中维护这

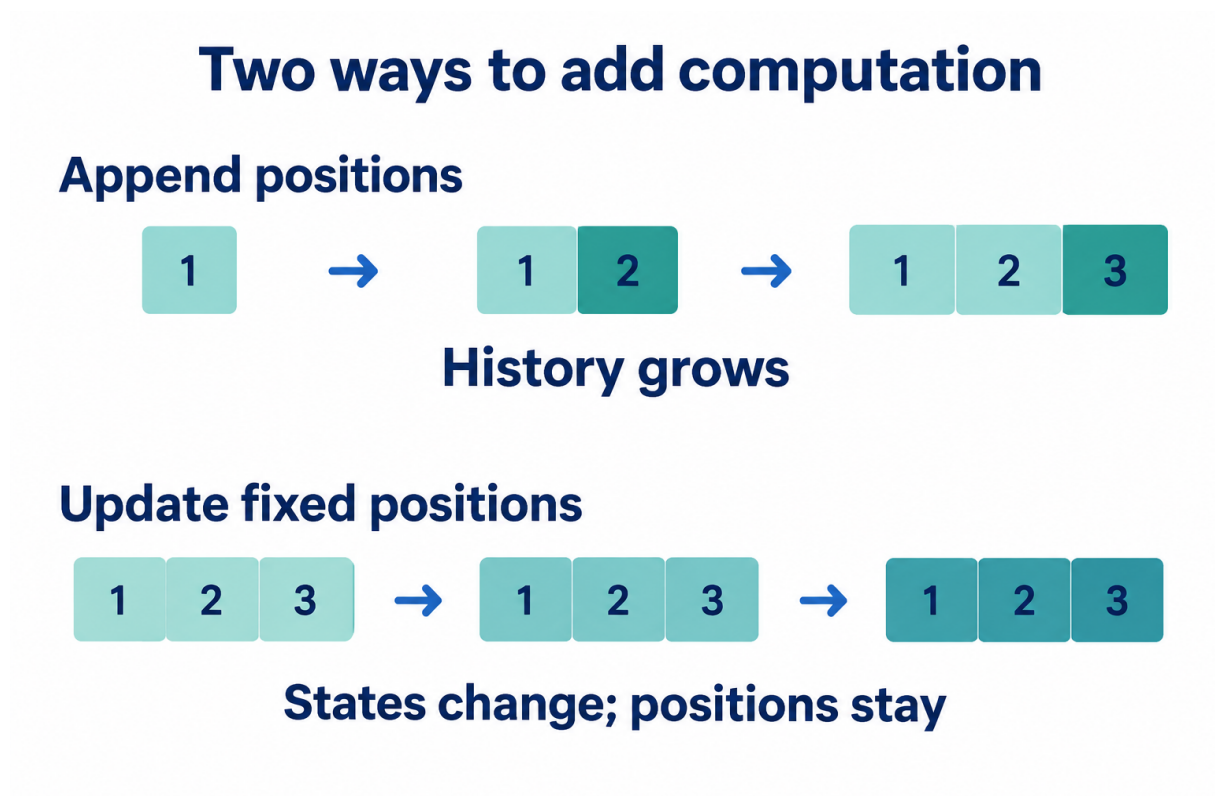


图 7.1: 增加计算的两种地址组织：追加潜在位置会增加可访问历史；循环深度则在固定位置集合上更新状态。颜色仅区分示意状态，不代表实测激活或质量。

来源与边界：概念性教学图；依据 [9, 12] 及本章证据整理，不构成新增实验结果。

些量及其关系。后者未必信息容量更小，因为它可能具有较宽状态和多个输入位置；前者也未必显式保存了三个语义变量，因为潜在表示不受这种离散解释约束。比较时应改变状态预算并做干预，不能仅用“一个向量”与“三个 token”判断机制强弱。

7.3 连续反馈也不只有一种接口

Coconut 反馈的是最终隐状态。PonderLM 则从预测的词分布构造加权词嵌入，并将其加入输入嵌入后再次运行语言模型 [27, 第 2 节、图 2]。若 $V \in \mathbb{R}^{|\mathcal{V}| \times d}$ 为词嵌入矩阵，则这种接口包含

$$p_r = \text{softmax}(z_r), \quad v_r = p_r^\top V. \quad (7.3)$$

原文式 (7) 的多轮更新是累积残差：若 E_0 是原始输入， v_r 是第 r 次产生的加权嵌入，则 $E_s = E_0 + \sum_{r=1}^s v_r$ ，不是每轮仅保留最新一次 v_r 。式(7.3)写的是完整词表分布；实际实验还采用 top-100 候选近似来节省混合开销，见原文第 2 节脚注 1。下面的凸包判断针对公式中归一化的概率权重，而不把截取候选的实现细节默认为完全相同。概率非负且相加为一，因此 v_r 位于词嵌入的凸包内。这是线性代数结论：它说明新增的反馈分量受到词表坐标的约束，整个残差输入 E_s 则不受同一个凸包约束，却不说明一次计算只代表一个词，也不说明连续状态天然实现了离散搜索树。对词分布取期望会把多个候选融合在一起；是否能保留后续计算需要的区分，取决于模型和任务。

这一点也使“语言空间”与“连续空间”的二分变得过粗。Coconut 的接口跳过词表投影；PonderLM 经过词表分布再回到连续嵌入；Huginn 的核心更新则在其专门的隐状态空间进行，最后再读出词分布。三者可以都不生成可见中间文本，却在信息瓶颈、可解释接口与每轮计算代价上不同 [9, 12, 27]。

比较对象	反馈接口	位置如何变化	训练信号的关键区别
Coconut	末层隐状态成为下个输入	潜在阶段追加位置	用课程替换语言步骤，监督余下文本
Huginn	共享核心接收状态与固定条件	既有位置沿深度更新	随机轮次上的下一词损失
PonderLM	词概率加权嵌入并作残差反馈	在输入表示上重复处理	预训练学习 pondering
Ouro	共享层栈的输出状态	既有位置沿深度更新	多深度损失及退出分布

表 7.1：计算接口的机制比较，依据各论文方法节；不是性能排名。

图示对照直接隐状态、词分布到加权嵌入、共享核心内部状态三类接口。中间通路仅展示一次残差反馈接口，不是 PonderLM 全部累积计算；加权嵌入也不是并行运行多个完整模型。

7.4 计算可组合，监督却不能自动迁移

从计算图看，连续位置的每次前向完全可以由一个深度循环模型执行；反过来，循环核心也可以接收由其他模型提出的潜在表示。因此两条路线并非互斥。但这是组合的可行性分析，不是已经取得性能收益的实证结论。若有 J 个连续位置，每次运行 K 轮核心，在其他条件近似固定时，主干工作量通常同时依赖 J 和 K ；把两种“思考”放到一起，不会让它们的开销相互抵消。

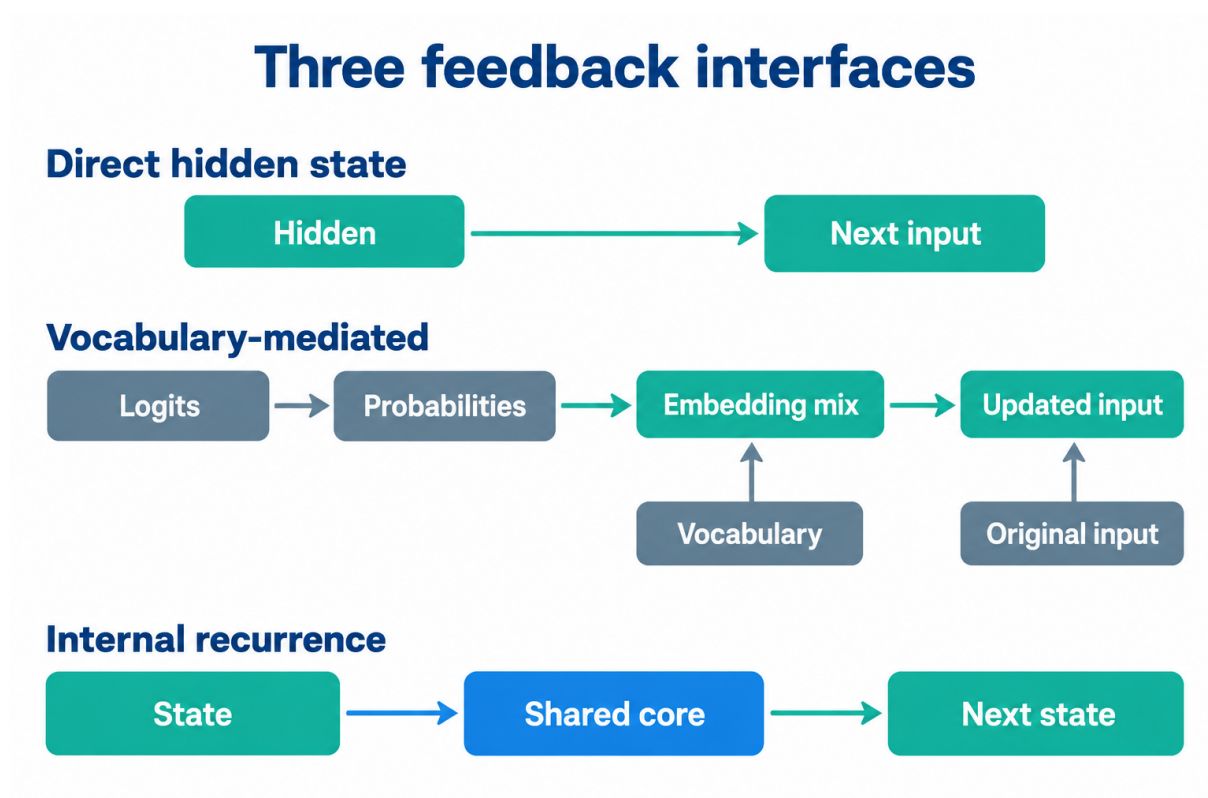


图 7.2: 三类连续接口：直接隐状态反馈、经词概率和词嵌入混合的反馈、共享核心内部的状态更新。中间通路示意一次残差更新；多轮累计依正文方法定义，图中不表示并行离散搜索。

来源与边界：概念性教学图；依据 [9, 12, 27] 及本章证据整理，不构成新增实验结果。

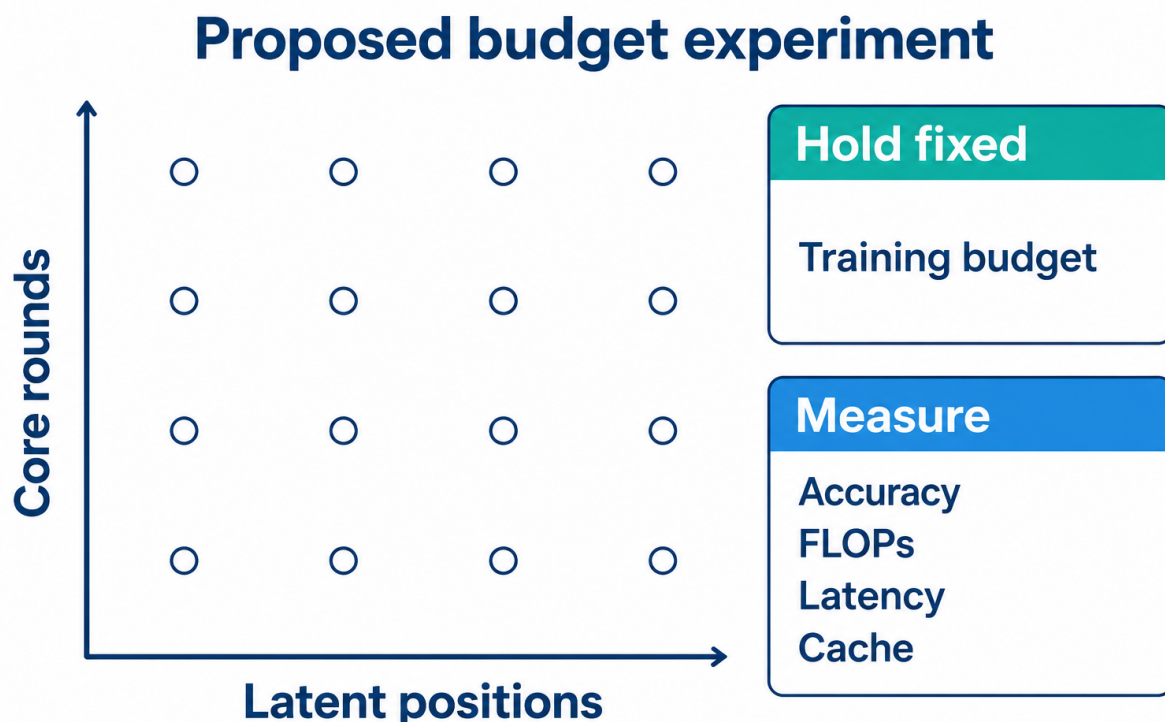


图 7.3: 待执行的二维预算实验: 分别改变潜在位置数与核心轮数, 在固定训练预算下记录准确率、FLOPs、延迟和缓存。空心格点代表待测配置, 不是实验结果或推荐最优点。

来源与边界: 概念性教学图; 依据 [9, 12] 及本章证据整理, 不构成新增实验结果。

更难的是训练职责。Coconut 的课程逐步替换显式推理步骤, 损失仍施加在剩余语言 token 上; 论文特别说明, 目标并不强迫每个连续向量压缩被移除的那一句话 [12, 第 3 节]。Huginn 则把不同循环深度作为预训练的一部分。前者提供了围绕问题求解的课程结构, 后者提供了更广泛的下一词任务和随机执行预算。把前者的结论当作后者会自然出现语义步骤的证据, 或把后者的预训练可行性当作前者无需课程的证据, 都是跨越了监督条件。

一个严谨的混合实验应先固定输入、训练 token 数和总推理预算, 再分别改变连续位置数与核心轮数。可以从一个二维网格开始: 横轴 J 、纵轴 K , 记录每个配置的准确率、前向 FLOPs、关键路径和缓存峰值。必须同时保留只增加 J 、只增加 K 以及固定二者的基线。否则, 一个混合模型优于浅模型, 只能说明用了更多计算, 不能说明两条路线存在互补机制。

图给出不含准确率颜色或最优点的二维实验设计。格点表示待测试配置, 旁边列出必须控制的训练预算和必须测量的延迟、缓存。它是研究方案, 不是已发表的实证热图。

7.5 中间状态可读, 不等于中间状态因果充分

两条路线都可以在某一轮接上词表头, 观察概率或解码结果。这个操作有教学价值: 它让我们看见输出如何随计算改变。但若探针能读出一个中间答案, 不能立刻断言该答案就是后续计算使用的因果变量。Coconut 将词表投影用作潜在状态分析, PonderLM 的反馈本身经过词分布, 两者的“可读性”在模型中的地位也不相同 [12, 第 3–4 节][27, 第 2 节]。

本书建议把观察、干预和算法解释分开。观察问“能否恢复中间变量”; 干预问“替换或破

坏这个表示是否按预期改变答案”；算法解释还要问“这种关系能否在不同长度、干扰项和运行深度上保持”。这是证据要求的综合，而不是已被所有模型满足的清单。尤其不能因两类模型都在连续空间中工作，就断言它们都会同时探索许多离散分支。

7.6 一个让两条路线公平相遇的任务

可以设计一道由两个阶段组成的任务：先从带干扰的文档中收集若干事实，再将这些事实按规则组合成答案。我们希望知道，增加潜在位置是否主要帮助保留多项证据，增加循环轮次是否主要帮助修订和组合。不能直接把这个直觉写成结论；应先独立改变事实数量、规则链长度以及无关文本长度，再观察不同预算分配是否产生不同失效模式。

例如，把事实数量固定，只增加组合链长度。如果循环轮数的收益随链长度上升，这与“增加计算帮助组合”的假说相容，但仍可能由更大总 FLOPs 解释。接着固定 FLOPs，在较多潜在位置和较多内部轮次之间重新分配预算，才开始接近机制比较。还可以保留文本长度但置换关键事实，使模型必须真正使用输入条件；否则训练中常见的答案模式可能让两条路线都显得有效。这些是实验提案，不是本书引用论文已经完成的测试。

记忆干预需要对称设计。对于追加位置模型，可以屏蔽某个潜在位置的缓存；对于深度循环模型，可以替换某一轮中的一组位置状态。两种操作破坏的数量和分布未必相同，因此应报告影响范围，而不是直接比较一个百分比的准确率下降。若恢复关键中间变量能够挽救错误轨迹，证据会比单纯解码更强；若恢复无效，则需要考虑该变量只是伴随信号，或者干预破坏其他必要信息。

最后还应检查两种预算是否真正互补。如果混合配置始终被纯粹增加其中一轴的配置支配，那么在当前任务、训练配方和硬件下，混合设计并未证明价值。如果混合配置形成新的成本-准确率折中，再进一步分析是哪一类题目受益。这样，模型名称与“连续思考”的共同叙事会退到背景，状态接口、记忆和更新次数成为可检验的研究对象。比较的依据仍是两篇方法中不同的地址更新方式，而性能预期保持开放 [9, 12]。

沿着本章的比较，我们可以更精确地提出互补假说：追加历史位置也许有助于保存阶段性证据，固定位置循环也许有助于对当前表示做反复修订。这里两个“也许”都应保留；它们需要在相同计算与记忆预算下通过任务拆分和因果干预检验。无论采用哪种组合，最后仍要决定何时读出。状态发生变化不保证答案更好，状态不再变化也不保证问题已经解决。下一章就从这个停机难题继续。

第八章 何时停止：自适应计算与过度迭代

8.1 多做一步，可能是在离开正确答案

给定一条循环轨迹，读者很容易想象准确率随着轮数上升，直到趋于平稳。这个图形很适合讲故事，却不是共享更新器自动满足的性质。训练只约束被监督的状态及其邻域；在更深位置，模型可能继续改善、保持不变、振荡，也可能破坏已经足够好的表示。所谓 *overthinking* 在这里是一个可测量的现象：某些输入在较早深度已经回答正确，继续更新之后却变错，而不是对模型心理状态的描述。

早期 Deep Thinking 工作就研究了这种外推退化，并用重复输入 recall 与 progressive training 改善其研究任务上的稳定性 [3, 第 3 节]。但这些实验的循环网络与开放域大语言模型不同，不能把解决特定算法任务中的过度迭代，升级为所有循环 Transformer 的保障。更重要的是，停机不仅是省算力的附加功能；它决定我们最终选择轨迹上的哪个状态。

8.2 先把轨迹质量与读出策略分开

设第 r 轮输出为 $\hat{y}_r$ ，正确标签为 y 。在最大预算 K 内，可以定义一个只用于分析的 oracle：

$$A_{\text{oracle}}(K) = \Pr[\exists r \leq K : \hat{y}_r = y]. \quad (8.1)$$

实际策略选择 $\tau(x) \leq K$ ，其准确率是

$$A_{\text{policy}}(K) = \Pr[\hat{y}_{\tau(x)} = y]. \quad (8.2)$$

同一条轨迹上，oracle 至少不会比真实策略差，但 oracle 使用了测试标签，只能作为诊断上界，不能当部署结果。若上界很低，问题可能在轨迹本身；若上界高而策略低，才值得优先研究选择器。这一分解与随机停止研究对 oracle-policy gap 的诊断相呼应 [15, 附录 C.3]。

教学图用“错、对、对、错”的人为序列，分别标出固定末轮、一个停机策略和事后 oracle 的选择。图中没有实验概率刻度。这个例子只说明选择问题，不声称所有模型都呈现该轨迹。

同样需要区分 best depth 和 final depth。前者在一组候选深度中选出较好结果，后者报告预先确定的最后预算。若直接用测试集挑 best depth，再把它当成已知最佳部署配置，就把选择信息泄漏进了评价。合理做法是在验证集上定阈值或深度，再在测试集上同时报告固定策略、最大预算及不同预算下的结果。

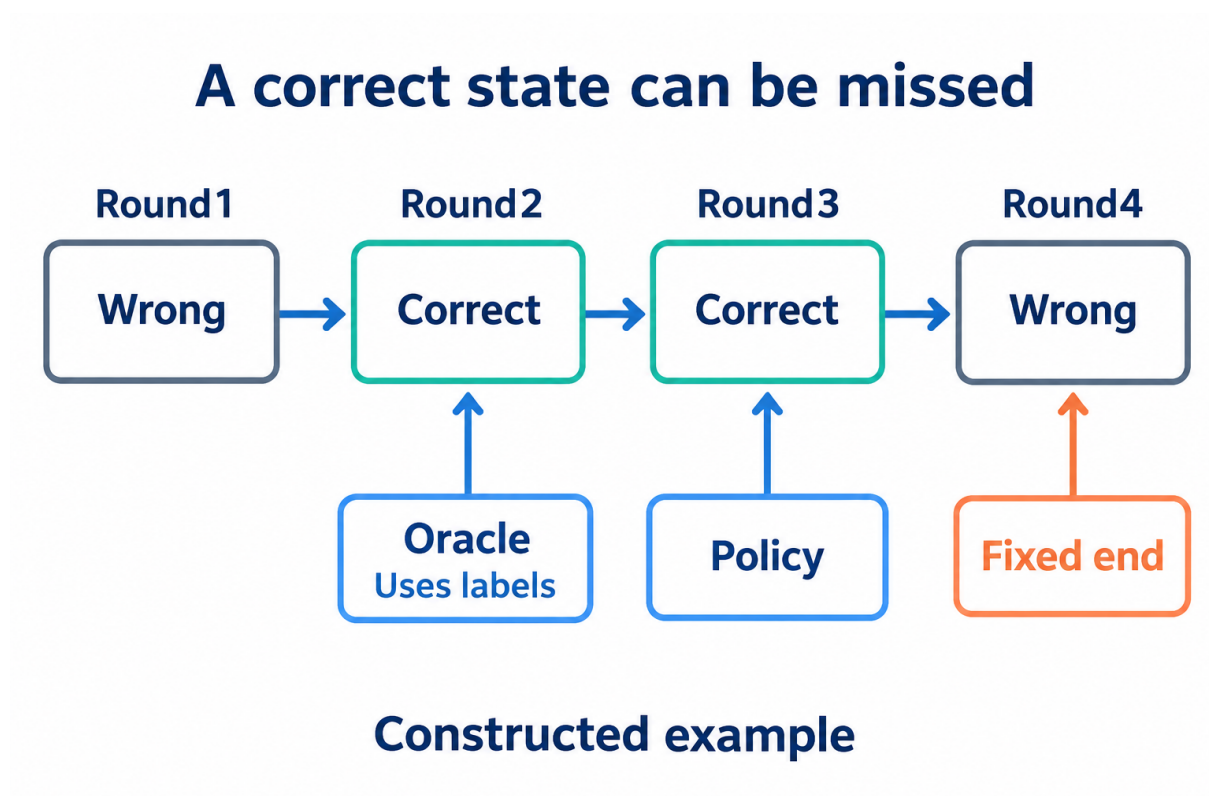


图 8.1: 人为构造的轨迹说明读出选择的重要性：正确状态可能早于最终轮次出现。Oracle 使用标签，是诊断上界；图中 Policy 的成功仅属于该例，不能推出实际策略总能选对。

来源与边界：概念性教学图；依据 [15] 及本章证据整理，不构成新增实验结果。

8.3 三类停止信号解决不同问题

ACT 累加停机质量，并通过剩余权重构造状态和输出的混合；它的训练还显式惩罚 ponder cost。Ouro 使用逐轮条件退出概率构造退出分布：若 a_r 是到达第 r 轮后停止的概率，则

$$q(r | x) = a_r \prod_{j < r} (1 - a_j), \quad \tau = \min \left\{ r : \sum_{j \leq r} q(j | x) \geq q_0 \right\}. \quad (8.3)$$

乘积表示之前都没有停止， q_0 为部署阈值；有限最大深度还需要末轮兜底。若要把这组权重当作有限支持上归一化的退出分布，就需把未退出的剩余质量归入末轮（等价于将最后一个条件退出概率置为 1）；这与部署时阈值未达到便返回最大深度的兜底规则应分别记录。它与 ACT 直接累加 sigmoid 输出并分配 remainder 的构造不同，尽管二者都可以用“累积到阈值”为直觉说明 [11][28, 第 3.2 节]。

第二类信号来自预测置信度，例如词分布熵或相邻轮次分布变化。它们容易计算，却把“输出确定”当作“继续计算价值低”的替代指标。模型可能非常确定地答错，也可能在两个等价表述之间摇摆。第三类信号来自状态变化，如 $\|s_r - s_{r-1}\|$ 。状态趋于稳定并不等于答案正确；不同坐标尺度和归一化还会改变这个距离的含义。Ouro 第 5.4.1 节确实比较了隐状态差阈值、固定退出和学习门控，但比较范围与任务必须保留 [28]。

其专门的 gate 训练使用相邻轮次任务损失的改善作为监督，损失信号对语言模型表征停止梯度，让门控学习哪些更新仍有价值 [28, 第 3.4 节]。这里训练时可以利用标签，推理时只能使用门控预测。原文 v1 同节公式与解释出现 t 和 $t-1$ 的索引不一致，因此本书仅引用其监督原则，不将有歧义的具体索引抄成可执行算法。即使索引明确，贪心的一步收益也不保证全局最优：某次更新可能暂时无收益，却为后续步骤准备条件。

8.4 训练时在新一轮停止，也会改变学到的轨迹

如果模型只在第 K 轮被监督，它可以允许前面各轮处于不适合读出的状态。若多个深度都要读出，训练就要求模型较早形成有用结果，或至少保持一个可被共享头解释的表示。由此，训练停止分布不能当成无关实现细节；它改变目标函数

$$\mathcal{L}_\pi = \mathbb{E}_{(x,y)} \mathbb{E}_{K \sim \pi(\cdot|x)} \ell(\hat{y}_K, y). \quad (8.4)$$

Huginn 对深度采样，Ouro 在多轮损失上构造权重；随机停止研究则明确比较固定、长度匹配、手工随机和学习随机停止日程 [9, 15, 28]。同样叫自适应计算，可能是在训练时选择监督终点，也可能是在部署时决定何时退出，二者需要分别说明。

Kuo et al. [15] 的主要诊断训练长度小于 20，在二进制加法上测试 20 至 60 等更长长度，并扩展到 Dyck-1、Unique Set 和 Copy。其循环模型复用三层块；RL-Halting 每次采样一个停止深度，以该深度的负损失作为奖励训练停止头 [15, 第 2-3 节]。论文发现随机日程可减轻一些运行间与深度间的不稳定，但稳定的策略也可能停留在次优计算上，Copy 上的学习停止并非最优 [15, 第 4 节、附录 C.3]。因此“更稳”与“更会外推”必须分开测量。

图中三条独立展开链表示固定终点监督、随机终点监督及多终点加权监督。步数仅作示例，损失路径表示训练监督而非推理状态更新；图比较目标结构，不暗示哪种策略获胜。

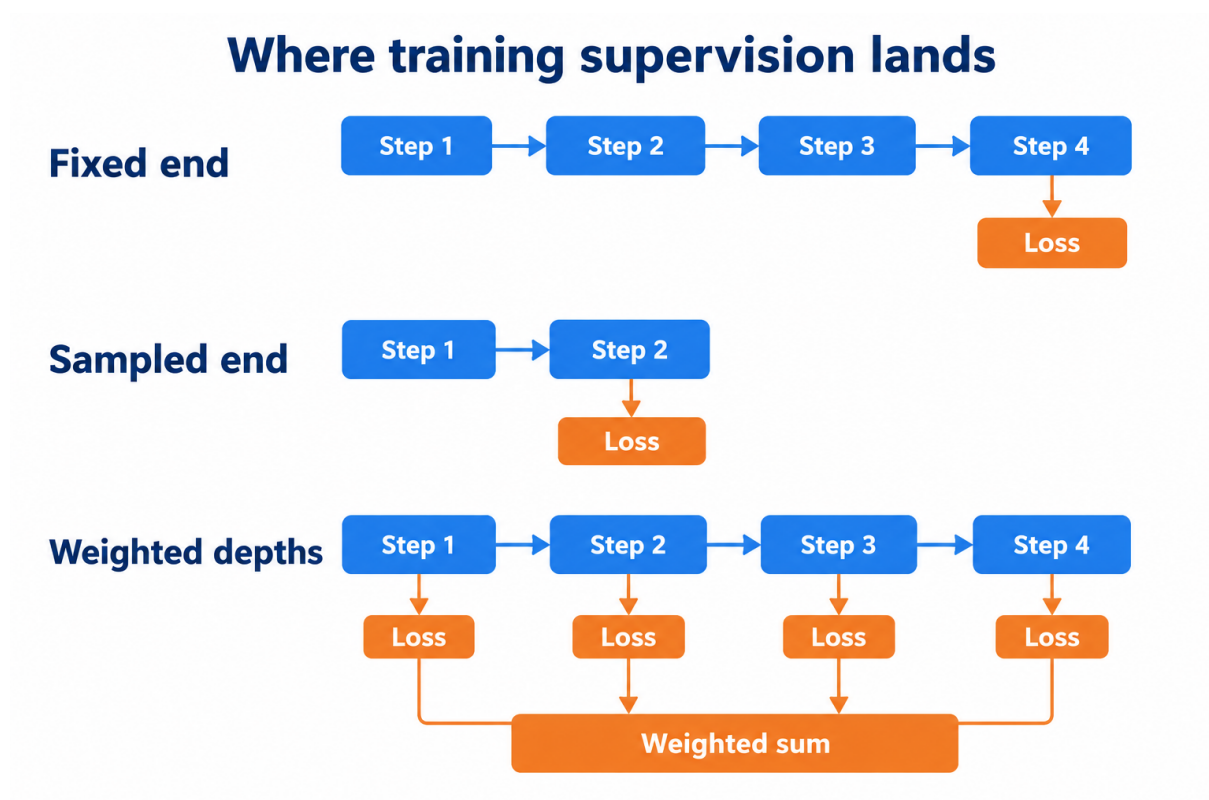


图 8.2: 三种监督终点：固定终点、抽样终点和多深度加权损失。图中的步骤数量为示意；下行橙色边表示损失计算和汇总，不是推理状态更新，也不意味着三种方法具有相同训练成本。

来源与边界：概念性教学图；依据 [9, 15, 28] 及本章证据整理，不构成新增实验结果。

Keep depth budgets separate

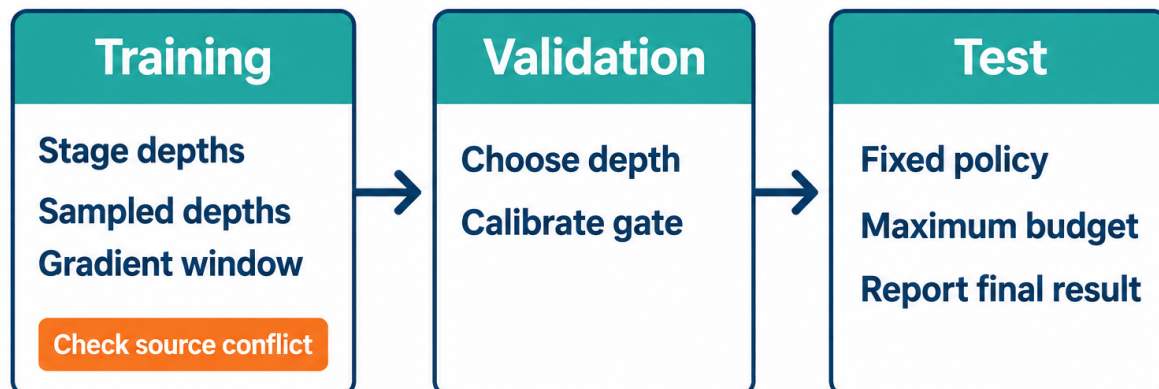


图 8.3: 深度评估应分开记录训练阶段与梯度窗口、验证集上的深度或门控选择、测试时的固定策略及最大预算。训练配置存在来源冲突时，应先核对再解释“未见深度”泛化。

来源与边界：概念性教学图；依据 [28] 及本章证据整理，不构成新增实验结果。

progressive training 提供另一个视角：先运行随机次数得到部分状态，截断早段梯度，再训练后续随机次数把该状态继续推进。原方法把这种损失与完整展开终点损失结合，并保留输入 recall[3, 第 3.1–3.2 节、算法 1]。它试图减轻对某个固定轮次计数的依赖。由此得到的研究问题并不是“循环多少次最好”，而是“模型是否学到了从多种可达状态继续工作的规则”。

8.5 真实模型的深度曲线应当怎样读

Ouro v1 的第 5.3 节给出基础模型和 Thinking 模型的逐深度结果。以该节表 10 的 1.4B 基础模型 MMLU 五样本平均为例，报告的 4 轮结果为 67.45，8 轮为 64.49；同一表中若干其他任务也在更深位置下降 [28, 第 5.3 节、表 10]。这两个数字仅是同一模型、同一表内的深度对照，不是跨模型排名。它足以反驳“更多轮一定更好”，却不足以解释退化来自数值失稳、状态漂移还是读出失配。

如第六章所述，论文将 5–8 轮称作外推，但其训练配方又包含前期 8 轮的条目。本书保留这项冲突，因此不把该曲线作为严格“从未见过的 8 轮”证据。读者还应注意，某项任务的最佳深度不必等于另一项任务，Thinking 后训练也可能改变曲线。报告一个平均 best 值会遮蔽这种结构。

本章第三幅图使用概念性评估账本：同一模型的训练阶段深度、验证选择深度、测试最大深度分别占一栏，并标出冲突需要核对。图中没有数值性能曲线；前文的表 10 数字只用于说明同一模型增加轮数仍可能退化。

最后，平均少运行两轮不自动意味着服务延迟减少一半。一个 batch 可能被最慢样本拖住，token 退出可能产生稀疏访问和调度开销，还可能改变历史 KV 的可用性。MoR 把路由、缓存与跨深度批处理放在一起研究，正说明停止策略需要系统设计配合 [1, 第 2–3 节]。公平评价应同时给出正确率、平均轮数、尾部轮数及真实延迟，说明计算粒度是样本还是 token，并让固定深度基线拥有同样的调优机会。

8.6 从停止阈值到可以部署的决策规则

设一个模型必须在固定服务预算内处理简单题和难题。平均轮数相同的两种策略可能表现很不同：策略甲对所有题都运行四轮，策略乙对一半题运行两轮、另一半运行六轮。如果乙把六轮分给真正能从额外更新获益的题，它可能更好；如果它把六轮分给始终答不对的题，预算便被浪费。题目困难程度和追加计算的边际收益并不是同一变量。

这意味着 gate 的监督最好明确回答“继续是否值得”，而不只是“当前是否不确定”。训练标签能够告诉我们某次更新是否降低损失，但部署时无法看到这个标签。校准实验可以按 gate 给出的继续概率分组，再统计追加更新的实际收益，检查高概率组是否真的更值得继续。同时保留无收益和负收益的比例；仅报告平均改善，会掩盖少数被过度迭代破坏的样本。Ouro 的损失改善监督提供了这种问题意识，但本段的分组校准是本书建议的进一步审计 [28, 第 3.4 节]。

分布变化会让这一审计更重要。在训练集上，低置信度可能主要来自尚可修正的中间状态；在新领域中，它也可能来自模型根本缺少知识。若 gate 把二者都解释成需要更多循环，就会增加成本而不提高正确率。相反，模型在错误知识上十分自信时，置信度停机可能很快退出。因此停止策略需要新的任务类别和输入长度上单独验证，而不能只沿用训练分布上的阈值。

报告时可以给出三种互补视图：按预算的总体准确率，按样本的首次正确与最终正确转移，以及实际策略相对 oracle 的差距。第一种用于部署取舍，第二种揭示过度迭代是否集中于某些题型，第三种定位停止选择器是否还有提升空间。随机停止论文的 oracle-policy 分析展示了最后一种诊断的价值；本书将它与预算及样本转移结合，是评估设计建议 [15, 附录 C.3]。三种视图缺少任何一个，都可能把“轨迹没有能力”和“有能力但没选对”混为同一个失败。

我们现在可以提出一个更严格的目标：好的自适应系统不仅应拥有有用的深度轨迹，还应不看标签的情况下选择合适读出，并在真实执行系统中兑现预算收益。稳定性、正确性、选择质量和延迟是四个相关但不同的量。第九章将讨论理论能够约束其中哪些量，第十章则把它们落到可复现的评估与成本账本上。

第九章 理论的边界：能表示、能学会与能泛化

9.1 先问是哪一种保证

前几章把额外计算放进了不同容器：可见文本、连续向量、共享模块的重复调用。理论研究的价值，是迫使我们说明容器的容量与调用规则；它不能替我们决定实际模型是否用好了这些资源。阅读一句“循环 Transformer 能执行算法”时，应把它拆成三个问题：是否存在一组参数实现算法，给定训练程序能否找到这组参数，找到后能否在新的长度和分布上继续正确。三个问题分别涉及表达能力、可学习性和泛化，不能相互代替。下文对几类定理的并列，是本书的综合框架，不是这些论文共同证明的一条定理。[13, 17, 23]

一个具体例子是奇偶校验：输入若干二进制位，输出其中一的个数是否为奇数。我们可以按顺序维护一位累加器，也可以把两两异或组织成树。两种算法计算同一函数，却有不同的串行深度、并行宽度和中间存储。因此“这个任务需要多少步”在未限定计算模型前没有唯一答案。连续状态、离散 token 和注意力头能处理的信息不同，换容器可能正是在改变问题中的资源约束。

把表达、学习和泛化画成三道独立关口，比把它们画成必然上升的阶梯更准确。通过第一关只意味着解在模型类中；通过第二关还需要说明数据分布、损失、初始化和优化规则；第三关必须声明测试变化的是长度、组合方式还是输入分布。一个有限长度上的高准确率实验，也不是对任意长度的全称命题。

9.2 计算模型的资源不能藏在形容词里

为方便审查，记输入长度为 n ，每次调用的层数为 L ，隐宽为 d ，数值精度为 b ，重复调用次数为 r ，额外生成长度为 t 。资源描述至少包含

$$\mathcal{R}(n) = (L(n), d(n), b(n), r(n), t(n), \text{注意力规则}). \quad (9.1)$$

这是本书的记账记号。它提醒我们：常数层数并不等于常数时间，固定参数数量也不等于固定数值信息量。允许随输入增长的精度，或者允许每个输入长度使用一套参数，都会改变结论。理论中的任意精度实数也不能自动替换成工程中的 BF16。

《The Expressive Power of Transformers with Chain of Thought》明确研究对数精度：输入长度 n 、中间解码长度 $t(n)$ 时，允许的精度随 $\log(n + t(n))$ 增长。其构造性能力结果使用饱和和注意力，即在达到最大分数的位置之间平均分配质量；图灵机模拟还需要严格因果掩码，当前位置不能看自身，并使用允许投影后归一化的 projected pre-norm。这些不是脚注中的装饰，而是实现信息检索与状态更新的工具。[17]

该文证明线性数量的中间解码步骤足以识别任意正则语言，并构造图灵机模拟。这里的“足以”是存在性结果；参数按目标算法设计，论文并未由此证明自然语言预训练一定学到该算法。

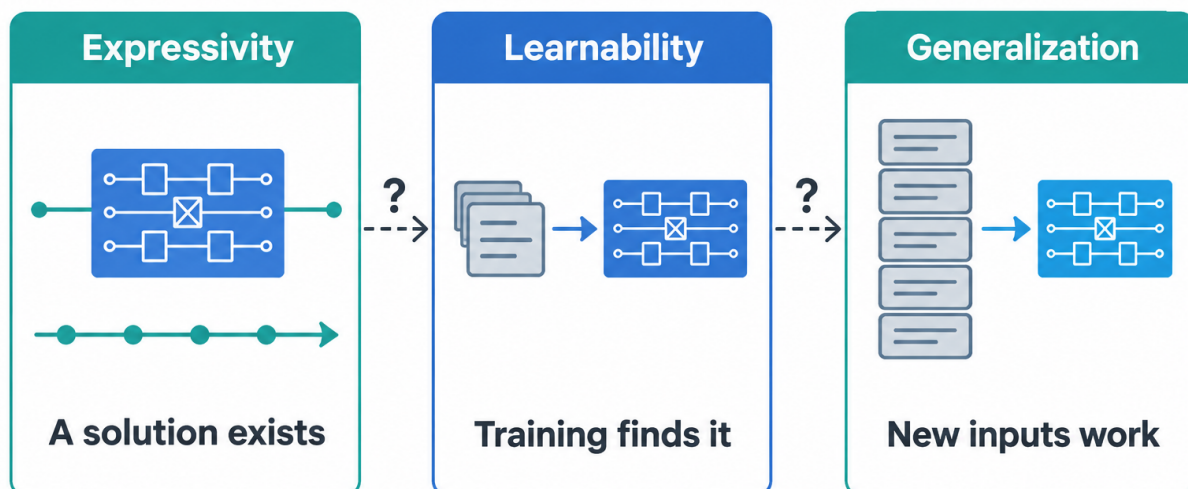


图 9.1: 概念示意：存在解、训练找到解与新输入上保持正确需要不同证据；问号处是尚需补充的条件。
来源与边界：概念性教学图；依据 [13, 17, 23] 及本章证据整理，不构成新增实验结果。

涉及无中间解码模型与正则语言能力差异的论述，还要保留文中的复杂度类分离假设。把它简写成“普通 Transformer 不会推理，CoT 会推理”，既抹去注意力与归一化条件，也把一个形式语言结论偷换成了人类任务分类。[17]

另一个方向问的是必须付出多少中间长度。《Lower Bounds for Chain-of-Thought Reasoning in Hard-Attention Transformers》在唯一硬注意力 UHAT 下得到奇偶校验所需 CoT 长度为 $\Omega(n)$ 的下界。其注意力每头选择一个位置，平局按固定规则处理；跨长度允许一族模型，但层数与头数一致有界，宽度不必一致有界。这个比单一长度泛化模型更宽松的设定仍受到下界约束。[4]

需要特别小心：这个证明不是“一个头每次只看一个位，所以必须看 n 次”的口头计数。被选中的位置本身由整个输入决定，选择行为已经可能携带信息。原文通过输入限制与敏感性分析控制这种间接通道。因此其结论既比朴素数格子严谨，也比“所有 softmax 模型都至少需要线性步骤”狭窄。这里的唯一硬选择，不能与允许并列最大值平均的饱和注意力混为一谈。两个模型类不同，定理看似冲突时应先检查这一点。[4]

9.3 循环可以逼近函数，但逼近什么函数

循环模型把同一块参数反复应用。这种共享约束带来一个自然问题：它是否仍能像非共享深层网络那样逼近复杂函数？《On Expressive Power of Looped Transformers》研究固定长度连续序列到序列函数，主要定理的目标类是在紧致区域上连续且置换等变的函数，误差用 L^p 范数度量。其分析中的 Transformer 不含层归一化，用 softmax 近似 hardmax，并把循环次数与目标函数的连续性模量联系起来。[23]

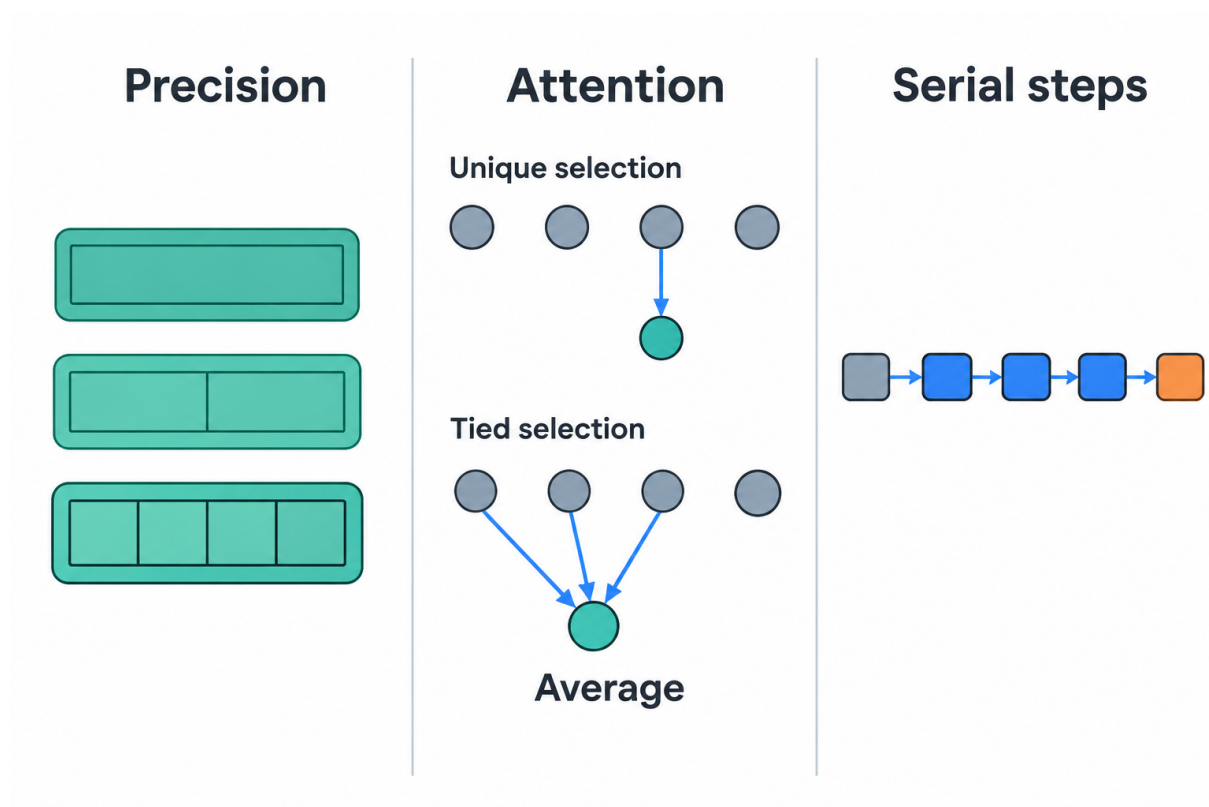


图 9.2: 概念示意：精度、注意力选择规则与串行步骤共同限定理论模型；并列最大值平均为一个注意力结果，分格不代表具体位数。

来源与边界：概念性教学图；依据 [4, 17] 及本章证据整理，不构成新增实验结果。

其定理还要求循环次数超过序列长度；由逼近界得到稠密性的推论允许无限精度权重。换言之，这里不能把“固定大小模块”理解为有限精度、固定时间的万能计算器。参数数目、权重所需位数和运行次数被分别计量，正是这项理论比笼统的通用逼近口号更有用的地方。[23]

连续性模量可以直观理解成“输入挪动一点，输出最多挪动多少”。序列模型还要区分整个序列变化、单个 token 变化、token 所处上下文变化。该文由此定义不同模量，揭示共享映射逼近不同上下文输出时的约束；随后引入时间步编码调制参数的 TMLT。这里的理论价值是指出哪些函数特征会进入逼近界，以及怎样给共享计算补充阶段信息，而不是给出一个对所有任务都成立的最佳循环次数。[23]

尤其不能把“固定长度上的通用逼近”读成“任意长度的泛化保证”。前者的定义域已固定，构造可以依赖该定义域与精度要求；后者要求在未见长度上继续遵守同一规则。该文同时讨论参数量与权重表示的位复杂度，也提醒我们不要把少量可学习标量误认为廉价的完整算法。若数值表示或循环次数随精度急剧增长，参数上的节省不等于计算预算上的节省。这是从其资源分析得到的解释，实际延迟仍要测量。[23]

9.4 从构造走向训练，需要补上哪些条件

近期《Transformers Provably Learn to Internalize Chain-of-Thought》把问题推进到训练动力学：它研究 k -parity 的树形中间监督，提出按树层几何移除思维位置的 Log-ICoT 课程，在指定设定下用 $L = \log_2 k$ 个训练阶段学习多层模型。这个结果的意义在于给出“中间监督如何逐层内化”的一条可分析路径，而不只是展示内部能够容纳答案。[13]

但它分析的是专门设计的简化模型。可训练部分主要是由位置编码决定的注意力矩阵；值矩阵固定，非线性连接函数满足特定条件，层间使用门控与分块移位，因果掩码受树层级限制。训练每阶段取新样本，做规定的梯度更新与整数化，并使用阶段相关学习率。理论嵌入维度也依赖输入、批量及层数。若删掉这些条件，只留下“Transformer 已被证明能学会隐式推理”，就会把证明真正解决的困难全部隐藏起来。[13]

更具体地，理论任务取 $k = 2^L$ 且 $k = \Theta(n)$ ，每阶段批量至少按规定的多项式规模增长，并对梯度 oracle 精度提出要求。其推理输入仍保留被置零的原 CoT 位置，而非把所有中间位置从计算图中删除。因此“一次前向”意味着不再逐个自回归输出中间 token，不意味着序列位置、注意力工作量和中间存储全部为零。阶段减少也不等于训练总成本等比例下降，因为每阶段样本量和计算图同样进入成本。[13]

这类结果仍然给实验提供了有价值的问题。比如中间监督是否必须遵循依赖图的层级？取消阶段门控后，先前学会的中间量是否会被覆盖？把手工树换成有共享子问题的图，几何移除是否仍然成立？这些是本书据理论结构提出的研究假设，不能算作原文已经证明的推广。合理做法是逐项放宽条件，记录性能何时下降，并定位下降来自表示丢失还是优化失败。

9.5 把定理变成可以使用的阅读工具

评价一项理论工作时，可以先写出它的量词：是对每个函数存在一个模型，还是同一个学习程序以高概率找到模型？接着列出随 n 增长的资源，再列出训练是否在结论内。最后问目标任务与实际任务之间还缺什么桥梁。这样并不会削弱理论；相反，它使理论能够排除错误解释，并帮助设计更有分辨力的实验。

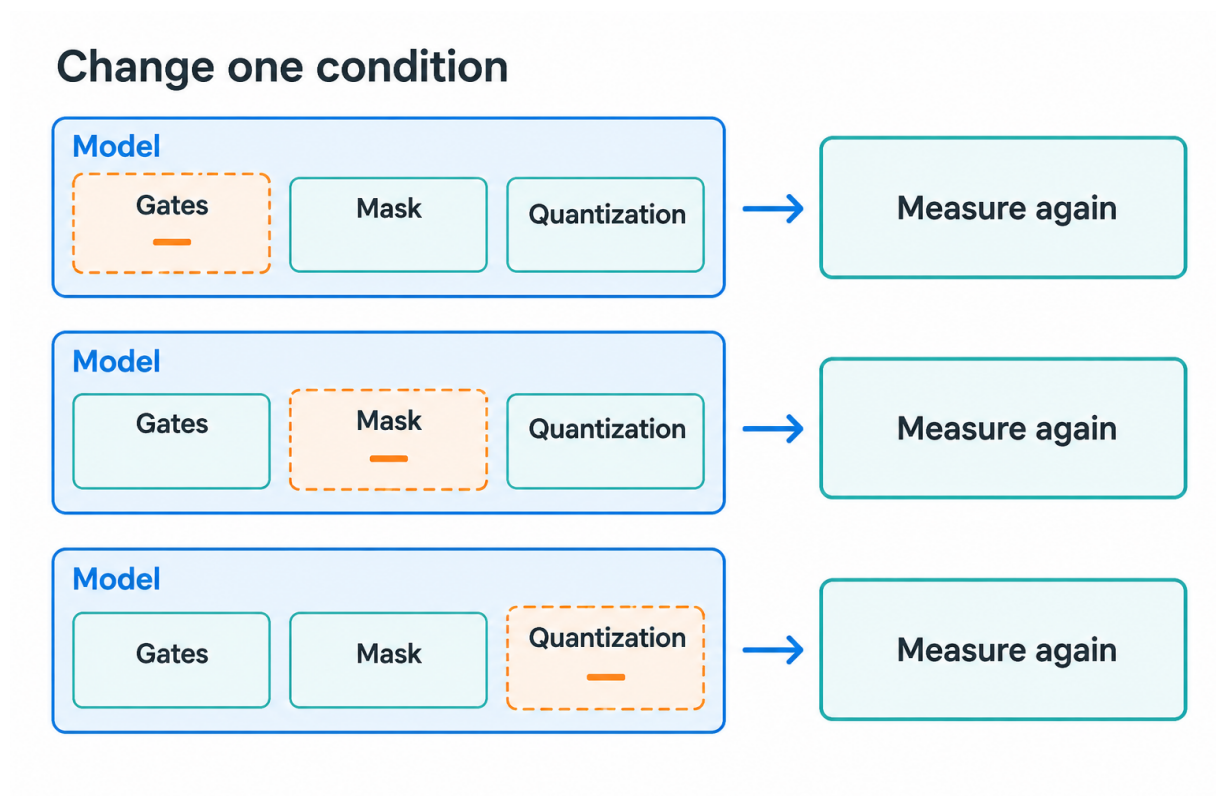


图 9.3: 概念示意：逐项放宽门控、掩码与量化条件，再分别验证学习行为；减号表示移除该条件，未表示实验已成功。

来源与边界：概念性教学图；依据 [13] 及本章证据整理，不构成新增实验结果。

例如，若一个训练好的固定深度模型在很长输入上突然失败，表达下界可能解释失败为何不可避免，也可能完全不适用，因为实际注意力属于另一模型类。若循环模型随测试步数增加而变好，存在性构造不能说明它学到了所构造的算法；还需要中间状态和干预证据。若内化模型在训练分布上保持准确率，学习定理也不能直接支持组合外推。第十章把这些逻辑区分落实为预算与对照，第十一章继续追问内部状态是否真的参与了所声称的计算。

第十章 成本与可信评估：让每一份额外计算有账可查

10.1 为什么一个更短的答案仍可能更贵

隐状态方法最容易展示的是可见推理 token 变少，循环模型最容易展示的是参数复用。两者都是真实的结构变化，却都不能单独证明部署成本降低：少输出文字的模型可能做了更多内部前向，参数共享的网络可能把同一块权重读写和计算很多次。CODI 把监督轨迹转入连续状态，TRM 把小网络递归用于状态与答案更新，正好说明“输出长度”和“参数数量”各自只覆盖了计算图的一部分。[14, 20]

本章提出的评估协议是本书的综合建议，不声称所有论文已经按此执行。基本原则是先明确要买什么：更高准确率、更低单请求延迟、更高吞吐，还是更少部署显存。它们可以同时改善，也可能相互交换。以参数受限设备为目标的循环模型，不必在延迟上全面胜出才有价值；但它应准确说明自己改善的轴。

10.2 把一条推理请求拆成计算图

设输入预处理的成本为 F_{pre} ，第 j 个潜在位置调用了 r_j 次循环核心，核心在当时上下文长度 n_j 下的成本为 $F_{\text{core}}(n_j)$ ，输出共 o 个 token。一个便于审计的分解是

$$F_{\text{request}} = F_{\text{pre}} + \sum_{j=1}^k r_j F_{\text{core}}(n_j) + \sum_{i=1}^o F_{\text{out}}(n_i) + F_{\text{select}}. \quad (10.1)$$

这是记账用的抽象，不是某篇论文的精确 FLOPs 公式。具体实现必须避免重复计费：若输出前向已经包含核心循环，就在输出项内部展开，不再加一次。若候选生成、验证器或重排序器参与最终决策，它们进入 F_{select} 。视觉编码、检索或教师调用也不能因为发生在“主模型”外就消失。

在同一请求中，至少区分总存储参数、一次调用所用的活跃参数，以及沿整条轨迹重复发生的参数使用。对于一个没有条件路由的共享核心，重复调用并不会创造新参数，却会创造新的运算。训练账本还需要前向次数、反向覆盖的步数、优化器状态与重计算；截断梯度让一部分前向没有对应反向，不意味着这些前向免费。TRM 的伪代码显式区分无梯度递归与带梯度递归，是阅读这种账本的好例子。[14]

延迟还取决于串行关键路径。两个独立候选可以在资源充足时并行，一条递归轨迹的后一步却依赖前一步。反过来，短序列上的多个小调用可能受内核启动和带宽限制，算术操作少也未必快。因而“步数减少不自动推出延迟降低”是一个测量纪律，而不是声称两者毫无关系。图中将存储、工作量与时间分列，正是为了避免从其中一项直接推断另外两项。

轴	应记录的量	容易混淆的替代量
模型容量	总参数、活跃参数、权重精度	只报核心参数
训练投入	数据身份、样本曝光、前反向预算	只报训练步数
推理工作量	潜在步、循环步、输出步、总 FLOPs	只报可见 token
时间	指定硬件下延迟分布与吞吐	理论操作数
内存	权重、激活、KV、峰值分配	checkpoint 文件大小
选择开销	采样次数、验证与重排序	最佳候选的单次成本

表 10.1: 本书建议的成本账本。各列是测量口径，不是任何模型的实测结果。

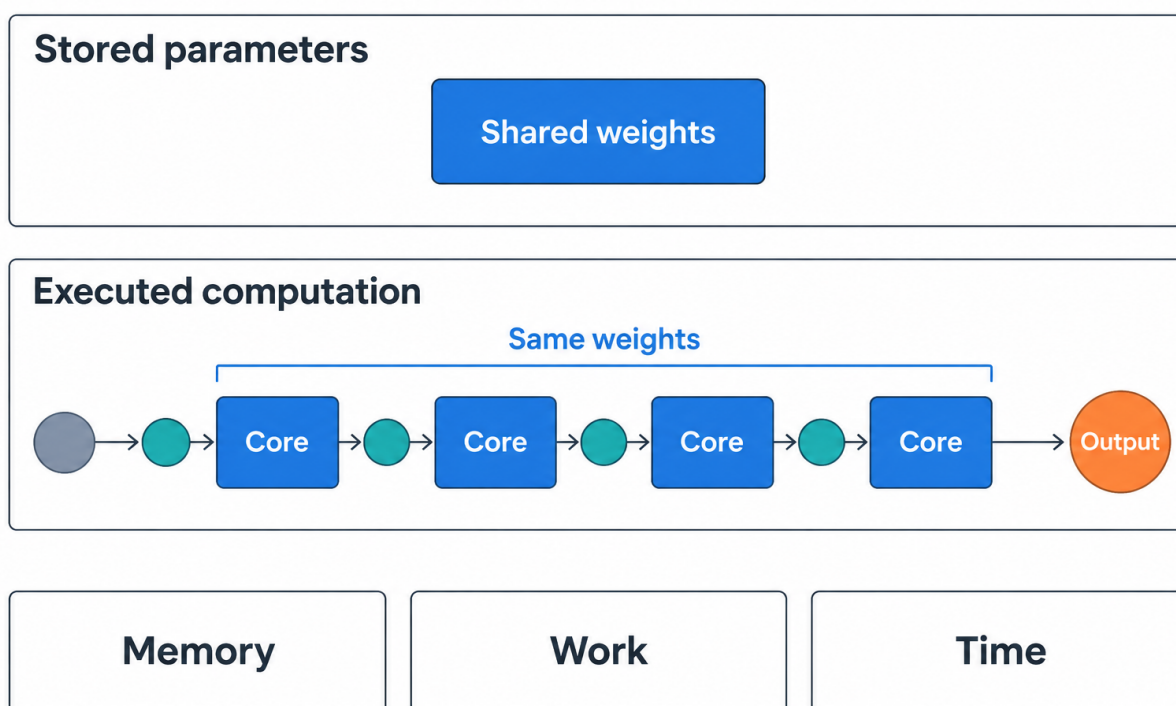


图 10.1: 概念示意：共享一份权重仍可对应多次前向计算；存储、工作量和实际时间需要分别核算，四次调用仅为示意。

来源与边界：概念性教学图；依据 [14, 20] 及本章证据整理，不构成新增实验结果。

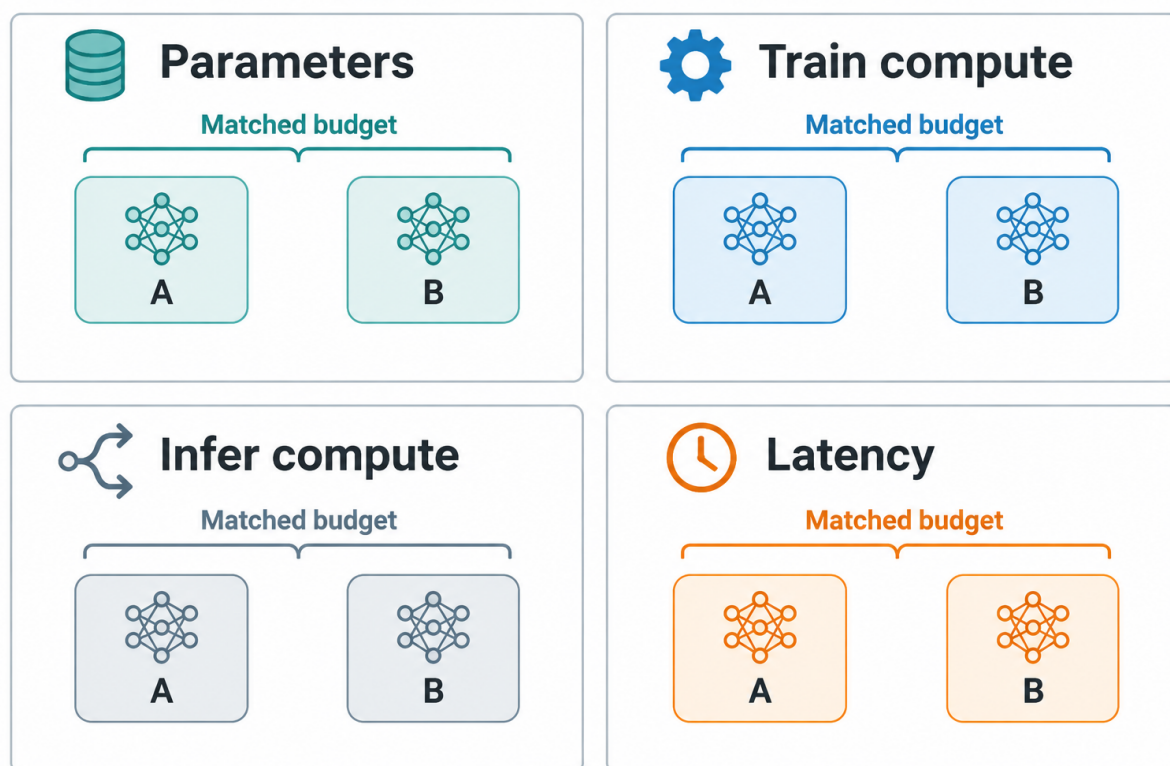


图 10.2: 概念示意：固定参数、训练计算、推理计算与延迟是四类不同的比较实验，图中不表示这些条件能同时配平。

来源与边界：概念性教学图；依据 [14, 23] 及本章证据整理，不构成新增实验结果。

10.3 公平比较需要不止一个切面

固定总参数回答的是：存储相同规模权重时，哪种计算结构更有效？固定训练 FLOPs 回答的是：投入同样训练计算后，哪种模型更好？固定推理 FLOPs 回答的是：同样请求预算下，哪种使用方式更好？固定延迟则加入了硬件和实现效率。除非专门构造实验，这些约束不会自动同时成立。把某个切面的胜利说成所有维度的优势，会制造无法解释的结论。

对一个共享核心的候选方案，建议至少建立三个基线。第一是结构相近、训练预算相同的非循环模型，检测额外循环是否值得。第二是同样可增加推理预算的显式 CoT 或重复采样模型，检测潜在计算是否比其他花钱方式更有效。第三是移除核心机制但保留辅助监督和训练时长的消融，检测收益究竟来自循环、教师信号还是新增数据。若条件无法完全配平，应公开差额并降低结论强度，而不是把不一致藏进模型名称。

一个假想例子能说明问题：方案甲使用较少潜在步，方案乙使用较多文本步，但甲在每步重算整个前缀、乙复用 KV 缓存。只数步骤会偏向甲，只数每步复杂度又可能漏掉乙随序列增长的缓存。正确实验应从实际调用轨迹累计操作量，并分别报告原生实现和经过等量优化后的实现。这是教学例子，没有对应的实测胜负。

10.4 训练成本不能从推理故事中退出

内化方法经常先学显式过程，再移除过程。训练期间的教师轨迹、蒸馏损失和课程阶段都是得到最终学生的原因，也都是成本。CODI 使用教师和学生模式并通过隐状态对齐传递信息；Log-ICoT 理论课程则按阶段采样和更新。两者的具体协议不同，但都说明最终推理图不是完整生产账单。[13, 20]

建议把已有基础模型视为共享起点，并另外报告本方法的增量训练成本；若比较方基础模型不同，再补充完整训练历史可获得的部分，不能把未知成本记成零。蒸馏还需要区分教师生成一次缓存轨迹与每个批次在线运行。对于会被调用大量次数的产品，训练成本可以按预期请求量摊销，但摊销数量必须明确。总生命周期成本可以写成 $C_{\text{train}} + NC_{\text{infer}}$ ，其中 N 是使用情景，而不是普遍常数。

反向图也会改变同预算比较。若方法截断了大部分循环的梯度，其每个训练样本可能比全展开反传便宜；节约的预算可以用于更多样本或更多步。若基线没有获得相同总预算，性能差异就同时包含架构和训练曝光差异。HRM 与 TRM 都让深监督和梯度处理成为设计的一部分，正因此不能只用一次前向的层数来总结训练公平性。[14, 21]

10.5 可信效果来自怎样的测试

首先固定数据身份与划分，防止同一题的表面变体跨集合泄漏。算法任务还应把长度、依赖深度和运算分布分开控制：输入更长不必然需要更多串行推理。对自然语言题，保留解析失败、无效输出、拒答和超时；若只统计成功解析者，短输出模型与长输出模型可能受到不同选择偏差。

其次，把单次作答、多个候选中的最好结果与自动选择后的结果分开。只有在部署时真的完美裁判，oracle 最佳候选才是可交付能力。自适应停机尤其需要逐样本保存步数和正确性，因为平均步数可能掩盖少数困难样本的预算爆炸。需要同时画准确率对平均成本与尾部成本的关系，报告预算上限内未完成的比例。

再次，区分固定 checkpoint 的推理预算扫描与每个预算重新训练。前者测量同一个已训练系统是否能利用更多测试计算；后者测量一个模型族的预算适配能力。若测试集被反复用于选步数或挑 checkpoint，即使没有直接反向传播，也已经参与了模型选择。应在验证集冻结配置，在保留测试集上做一次报告，并把探索性结果单列。

最后，需要对不确定性负责。多随机种子应保留逐种子结果，困难度分桶应报告样本数，配对方法比较应使用相同题目。与只给一个最高分相比，报告预算范围内的效果曲线和失败类型更能判断额外计算在解决什么。这里的曲线必须由真实日志绘制；概念插图只能展示坐标含义，不能凭空安排某条方法曲线领先。

10.6 一份足以复核的最小报告

读者应能从报告还原三个对象：哪一个模型、在哪一批输入上、执行了怎样的计算。例如，若内化课程把中间 token 换成零填充，输出上已经看不到轨迹，但这些位置可能仍占用注意力与激活空间。Log-ICoT 的理论推理设置就是保留置零思维位置的明确例子。报告“零生成思维 token”时，应同时写出这些位置是否还被计算；否则读者无法知道省去的是自回归串行过程，还是连并行状态空间也一并省去了。[13]

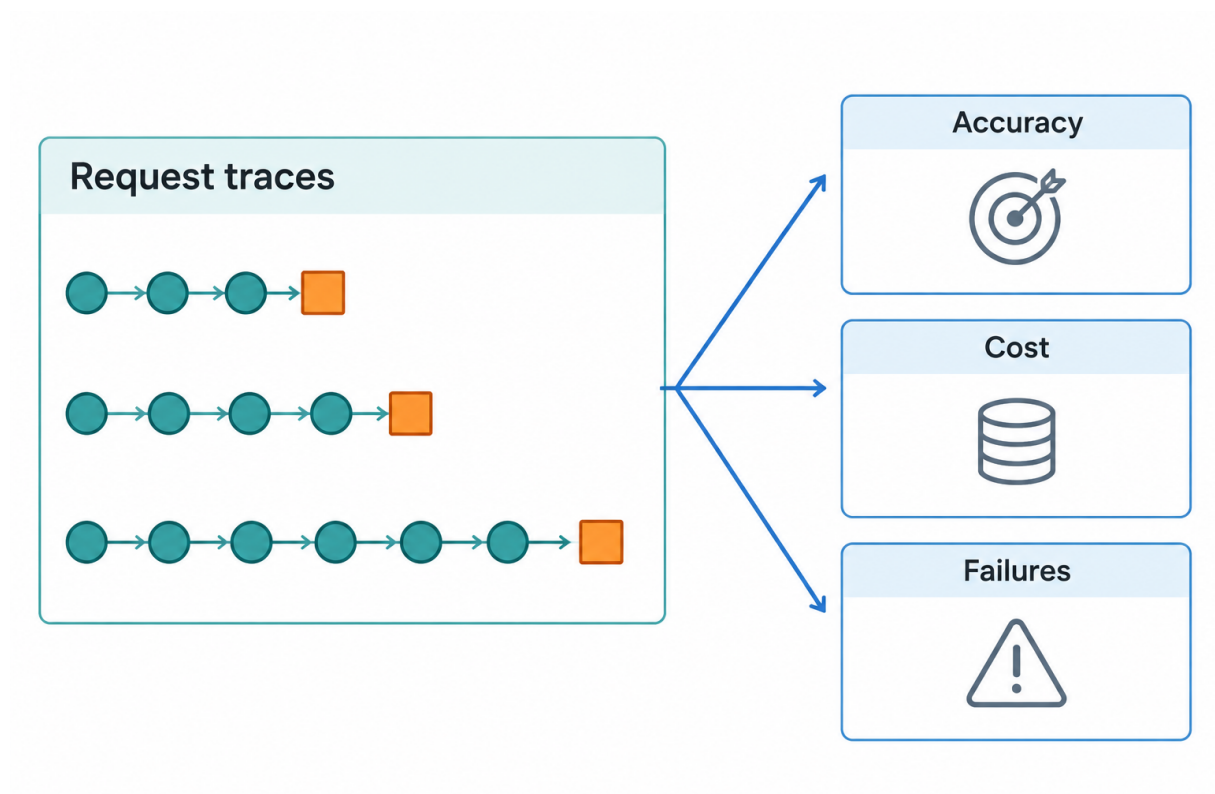


图 10.3: 概念示意：同一份逐请求轨迹支撑效果、成本和失败模式的联合审计；路径长短不对应实测结果。

来源与边界：概念性教学图；依据 [14, 20] 及本章证据整理，不构成新增实验结果。

最小记录包含 checkpoint 标识、代码版本、数据版本与划分、精度、批量、硬件、缓存规则、训练预算、推理预算、停止与选择规则，以及逐题输出和计时日志。冷启动与预热后的性能分开，测量 GPU 时间时处理异步执行，并说明预处理与结果后处理是否进入时间窗。

复核时还可以抽取少量请求，对照模型日志、设备计时和解析后的答案逐项检查。特别要核验终止条件：计时结束点是最后一个潜在更新、首个答案 token，还是完整输出到达调用端？三者分别对应内部计算完成、首 token 延迟与端到端延迟。它们都可以报告，但不能用较早的结束点与另一方法的完整请求时间比较。

这不是要求每篇论文都完成所有部署工程，而是要求结论与证据相称。只有 token 计数时，可以说减少了可见生成；只有 FLOPs 估计时，可以说减少了估计算术工作量；有相同硬件测量时，才报告该环境下的加速。下一章进一步检查：即使账目和准确率都清楚，模型内部执行的机制是否与我们讲述的故事相符。

第十一章 扩展与机制审计：状态里到底发生了什么

11.1 从语言向多模态迁移，究竟增加了什么

连续向量不受单个词的词表限制，因此看起来适合承载图像、空间关系与文字之间的中间表示。但“模型使用向量”本来就是视觉语言模型的常态；真正需要辨认的是向量是否在答案前被额外更新，更新能访问什么信息，以及这些访问是否改变决策。多模态输入本身不能证明模型已经在进行多模态潜在推理。

MCOUT 提供了一个具体案例。其原文以视觉编码器与语言模型构成基础架构，先组织视觉与文本嵌入，再增加连续思维位置。MCOUT-Base 将最后的隐藏状态反馈为下一思维嵌入；MCOUT-Multi 则让该状态通过多模态潜在注意力与输入嵌入结合。新思维嵌入被追加到输入序列中，随后继续计算。[18]

这个机制属于沿序列追加潜在位置的反馈，不能仅因“迭代细化”四个字就将其归为固定序列上的共享深度循环。也不能把访问既有视觉嵌入说成每步重新看原始图像：若视觉编码器只运行一次，之后重复读取的仍然是那份表示。这样的区分会改变成本估计，也会改变能力预期。初始视觉编码已经遗漏的细节，后续状态更新未必能够恢复。

MCOUT 在视觉语言任务上的报告提供了探索证据，但本文只把其架构作为可分析实例，不把某个最优增益推广到所有多模态任务。特别是“跨模态对齐改善”的机制解释，需要模态消融或干预来支持；总分提高也可能来自额外计算、辅助训练信号，或者利用更强的语言先验。原文中关于人类反思的动机不构成相同认知机制的证据。[18]

一个具有分辨力的测试是制作视觉最小对：只改变图中一个决定答案的关系，保持问题文字不变；再制作文本最小对，保持图像不变。分别在不同潜在步骤替换视觉嵌入，检查答案何时随证据切换。若只在初始步骤替换才有效，模型可能早已把信息压进状态；若后期替换仍按任务语义改变答案，才支持后续步骤持续读取相关模态。这个实验是本书建议，不是 MCOUT 已完成的结论。

11.2 层级递归是计算调度，不是架构阵营

HRM 的命名强调高低层级与不同更新频率，容易让人将它放到“非 Transformer 推理”一栏。然而原文架构说明明确：高层与低层循环模块都采用 encoder-only Transformer 块，输入通过逐元素相加组合，并使用旋转位置编码、门控线性单元与 RMSNorm 等组件。因此更准确的说法是，它在 Transformer 计算模块之上组织层级递归和监督。[21]

可用下面的概念式理解其调度：低层状态较频繁地结合输入与高层状态更新，高层状态较

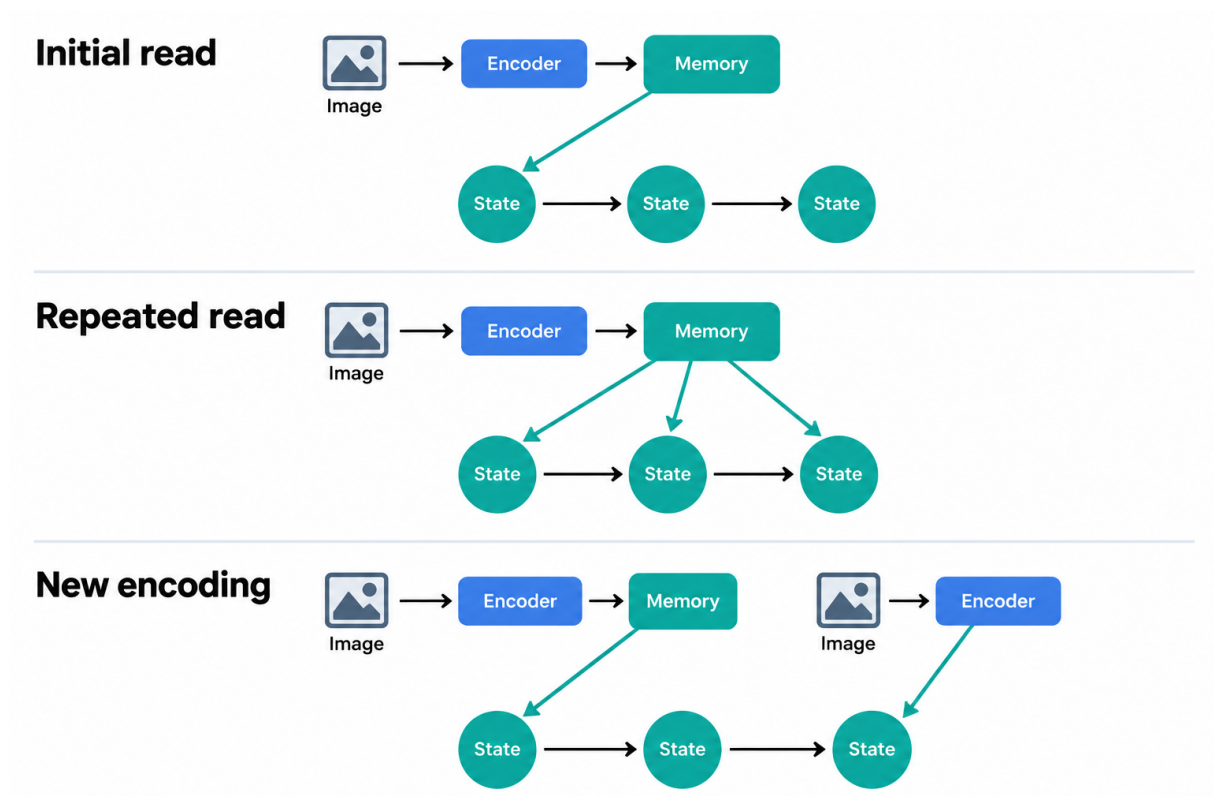


图 11.1: 概念示意：初始读取、重复读取固定视觉表示与重新编码图像是不同操作；新视觉证据进入持续更新的状态链。三行是机制对照，不表示 MCOUT 实现了全部三种方式。

来源与边界：概念性教学图；依据 [18] 及本章证据整理，不构成新增实验结果。

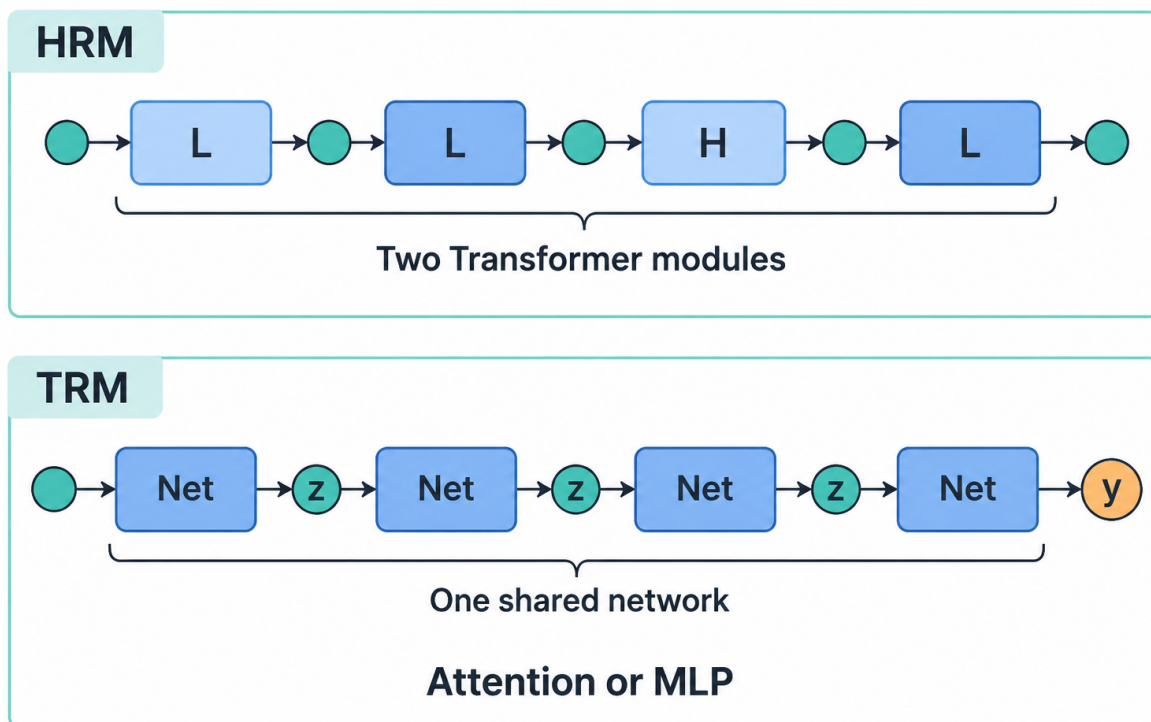


图 11.2: 概念示意：HRM 调用高低两个 Transformer 模块，TRM 复用一个网络更新潜在状态和答案表示。图仅示模块调用与复用关系，省略多状态融合、输入注入、梯度与精确循环次数；颜色深浅不编码模块身份。

来源与边界：概念性教学图；依据 [14, 21] 及本章证据整理，不构成新增实验结果。

低频地吸收低层结果，输出从高层状态读出。

$$z_L^{(i+1)} = f_L(z_L^{(i)}, z_H^{(j)}, x), \quad (11.1)$$

$$z_H^{(j+1)} = f_H(z_H^{(j)}, z_L^{(\text{end})}). \quad (11.2)$$

这里省略了具体的加法融合、初始化和循环索引，只表达信息依赖。不能从高低层命名推出高层一定负责抽象计划、低层一定负责执行；那需要任务分解或因果证据。不同频率先是一种计算图事实，功能分工是另一个待检验命题。

HRM 使用一阶梯度近似和分段深监督以处理很深的展开。固定点语言帮助解释该近似的出发点，但有限次前向是否到达足够接近的固定点，是实证问题；收敛也不等于答案正确。TRM 论文正面讨论了这个缺口，报告所检查轨迹在实际计算范围下不满足理想的固定点图景，并提出改用对完整递归单元反传的设计。这里应把它视为后续论文对特定设置的分析与替代，不能升级成所有 HRM 训练都无效的判决。[14, 21]

TRM 用一个小网络反复更新潜在状态 z ，再更新答案表示 y ，两者在监督步骤之间传递。原文同时讨论带自注意力与将自注意力替换为沿序列维度 MLP 的变体，且不同任务对两者的偏好不同。这使“Transformer 与非 Transformer 谁胜出”的笼统问法失去作用；真正可控制的变量是跨位置混合方式、状态数、递归次数、梯度覆盖和监督安排。[14]

这些模型在数独、迷宫、ARC 类任务上的结果值得研究，因为输入结构、输出约束与重复计算比较清楚。但小型专用任务上的监督学习成绩不能直接等同于开放领域语言推理。更有价值的迁移问题是：哪些结构约束让反复修正答案有效？对自然语言开放输出，错误能否像格子约

束冲突一样被内部状态发现？如果不能，单纯增加递归次数可能只是让既有判断更加稳定。

11.3 可解码的中间量，尚不是被使用的中间量

假设一个探针能从第 r 步状态 h_r 预测中间变量 s ，即 $q(h_r) \approx s$ 。这说明在该探针类和数据分布下，状态含有可恢复的信息。它没有证明主模型的答案读出使用了 q ，也没有证明该信息是经由预期步骤计算而来。状态可能保留了全部输入，探针自己完成了部分计算；同一答案也可能通过另一条捷径获得。

Logit lens、线性探针与注意力图各自回答不同问题。Logit lens 借用输出映射查看可读内容，线性探针检查一种简单读出能恢复的信息，注意力图显示某个信息路由的权重。三者均不能仅凭一张热图宣布因果机制。尤其是注意力权重大，并不意味着该位置的值向量对最终答案贡献最大；后续非线性与其他路径可能抵消或替代它。

《Do Latent-CoT Models Think Step-by-Step?》在 CODI 的多项式迭代任务上把这些诊断与 activation patching 结合起来。任务有明确状态更新 $s_t = s_{t-1}x_t + b \pmod{m}$ ，因此可以检查每个中间量。论文在浅层少跳任务中发现桥接状态，同时发现最后输入可直接进入答案读出；随着跳数增加，这些诊断没有显示稳定、完整的可解码逐步轨迹；这不排除现有读出未捕获的其他编码。[16]

这项工作的重要性不是宣布“所有潜在推理都是假推理”，而是提供了反例构造方式：正确答案和部分可解码状态可以与非预期路径共存。其主实验使用特定 CODI 训练协议与小型 GPT-2 风格模型；模数与更新函数也影响历史信息是否可压缩。作者观察到素数与合数设置的差异，并用多对一收缩解释其中部分机制。因此结论应保留任务代数结构，不能扩大成自然语言推理的一般规律。[16]

11.4 怎样做有分辨力的因果实验

Activation patching 的基本操作是，把干净运行的某处激活移入被扰动的运行，观察输出是否恢复。它比相关探针多走了一步：实验主动改变中间变量。但仍要问替换的是什么。若复制整个残差流，恢复可能来自携带答案或其他信息，而不只来自想研究的中间状态。若破坏制造了分布外激活，性能下降也可能只是网络受到异常冲击。

因此建议先在自然输入对上定义明确的反事实。例如两道题只有第二个操作数不同，正确中间值和最终答案都可计算。先确认破坏确实改变目标答案，再检查在指定步骤置换状态是否按反事实方向改变输出；增加同任务无关位置替换、随机匹配替换和后续步骤替换作为对照。若只发生“任意替换都坏”，证据支持脆弱性，尚不足以支持语义清晰的因果变量。

探针实验本身也需要保留训练和测试边界。探针应在独立样本上训练，容量与正则化选择不能直接使用最终机制测试集；不同步骤的探针要有相同容量预算。再加入只读输入的探针作为对照：若不看潜在状态也能同样恢复中间量，状态探针的高分可能主要反映任务本身易预测，而非计算已经完成。若换一种编码或非线性探针才读得到，也只能说明所选读出类的限制，不能立刻宣布模型没有该信息。

还应区分必要性和充分性。删除一路使性能下降，可能说明该路必要，也可能说明删除破坏了分布；补入一路可以修复，说明在该干预条件下它足够携带有用信息，却不说明原模型只靠这一路。更强的机制解释要让多种局部干预形成一致图景，并承认模型可能有冗余通道。本书据此

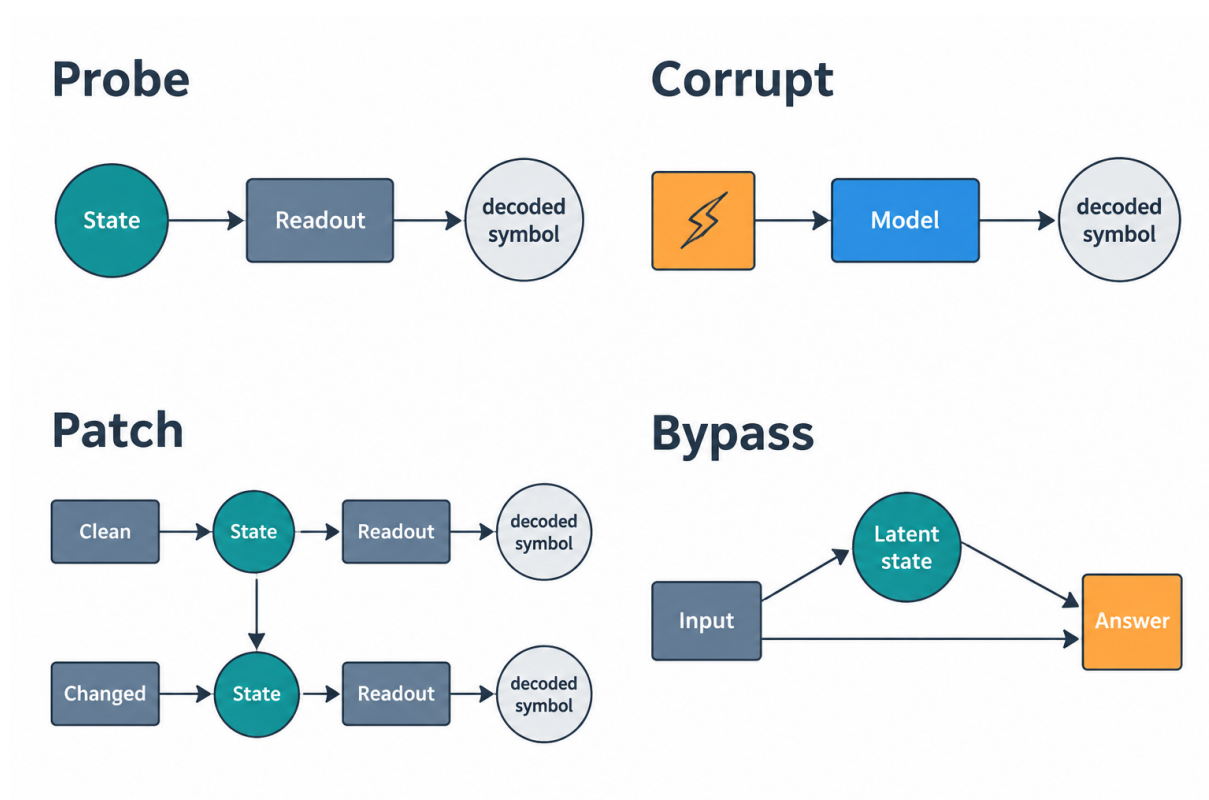


图 11.3: 概念示意：探针只读取状态，激活替换则把干净运行的状态移入扰动运行并检查模型输出；旁路可能同时影响答案。Readout 在左上指探针，在左下指主模型后续读出，符号节点不表示已恢复正确答案。

来源与边界：概念性教学图；依据 [16] 及本章证据整理，不构成新增实验结果。

把探针、干预和跨分布验证视为互补工具，而非一个万能证明等级表。

11.5 不把不可见误写成神秘

隐去文本只改变了观察窗口。隐藏的文字推理仍是离散生成，连续状态可以只是压缩摘要，循环轨迹也可以包含大量近似恒等更新。反之，没有逐步对应自然语言解释的内部算法也可能是有效算法。研究目标应是辨认哪些输入信息经由哪些状态变化影响输出，以及这些变化是否值得其成本，而不是要求每个向量都翻译成一句人话。

一个值得保留的反直觉可能是：模型做对了题，却没有执行我们期待的步骤。这不必意味着结果无价值；若捷径在目标分布上可靠且便宜，它也可能有实际用途。问题发生在把这种成功当作更长组合任务的保证。机制审计的任务是界定迁移条件，而不是给算法贴上“真正思考”的身份标签。相反，即使干预支持某条逐步路径，也应继续检查它在错误样本中是否仍然存在，以及它是否比更简单基线划算。

这一立场把多模态、层级递归与机制分析连接起来：先画真实计算图，再说明监督和梯度如何进入，最后用可失败的实验检验功能解释。第十二章据此提出研究程序，让“压缩”“更多计算”“算法泛化”这些常被混用的词分别面对可测量的预测。

第十二章 从综述到研究程序：让新问题能够被证伪

12.1 可以带走的结论与仍然缺少的桥梁

全书的两条路线都在重新安排计算。潜在思维把一部分中间过程移出可见词表；循环深度把一部分额外计算交给重复使用的模块。它们可以组合，但并不天然相同：追加状态位置会改变序列和缓存，固定位置循环会改变深度和状态轨迹。参数共享、可见步骤减少、预算增加和准确率提高，是需要分别证明的命题。前面的形式模型、成本分析和机制反例共同支持这种拆分，而不是支持一种架构普遍取代另一种的结论。[16, 17, 23]

现有证据留下三个相互关联的缺口。第一，潜在状态到底在压缩已完成的信息，还是创造新的任务相关计算？第二，共享更新是否学到了可重复使用的规则，还是仅在训练深度内拟合了轨迹？第三，自适应调度是否能把预算分配给真正可修复的错误，而不是只选择容易样本提前结束？本章的方案均是本书提出的研究假设与实验设计，不是已有结果。其目的不是增加方法名称，而是让失败也能提供信息。

12.2 研究一：压缩与新增计算可以怎样分离

这个问题来自 CODI 的机制分析：在部分任务上可以发现中间桥接状态，但更长链条的诊断未呈现完整的可解码逐步轨迹；任务更新的可压缩性又会改变结果。[16] 因此仅比较显式轨迹长度与潜在步数，无法确定学生是把多个步骤合并执行，还是利用了终点预测的捷径。

建议先使用可生成精确中间状态的任务族，分别控制依赖深度和信息收缩程度。第一阶段复用多项式迭代，并记录每个更新在有限状态空间中的多对一程度；随后扩展到有限状态机组合。素数或合数不能作为唯一解释变量，因为模数大小、可达状态频率和标签熵也会变化。实验应匹配这些统计量，另外加入相同序列长度但不同依赖深度的样本。若仍无法完全匹配，应报告残余差异，而不是把它算作机制证明。

在同一个基础模型、相同训练样本与训练 FLOPs 下，比较答案监督、显式中间监督、仅终点表征蒸馏和分阶段中间状态监督。潜在步数与状态维度分开扫描，并记录实际前向预算。主指标是保留组合上的准确率，以及在控制输入后，中间状态干预对输出产生的方向性影响。探针作为定位工具，不能单独决定成功。

关键干预是“续算测试”：选择处在相同计算阶段、但具有不同已知中间值的两条自然轨迹，把供体状态移入受体运行，同时保留受体的后缀操作。先计算由供体中间值和受体后缀共同决定的反事实答案，并优先选择该答案不同于受体原答案的配对，观察模型是否按置换后的语义状态续算。另用中间语义相同而前缀表面不同的配对检验表示兼容性；仅在这些配对上成功，不能证明状态的因果作用。还需要无关状态替换、等距离随机扰动和只替换输入缓存的对照，避免把状态搬运与缓存搬运混淆。若探针能解码，但续算不跟随置换的中间值，优先考虑被动保存或

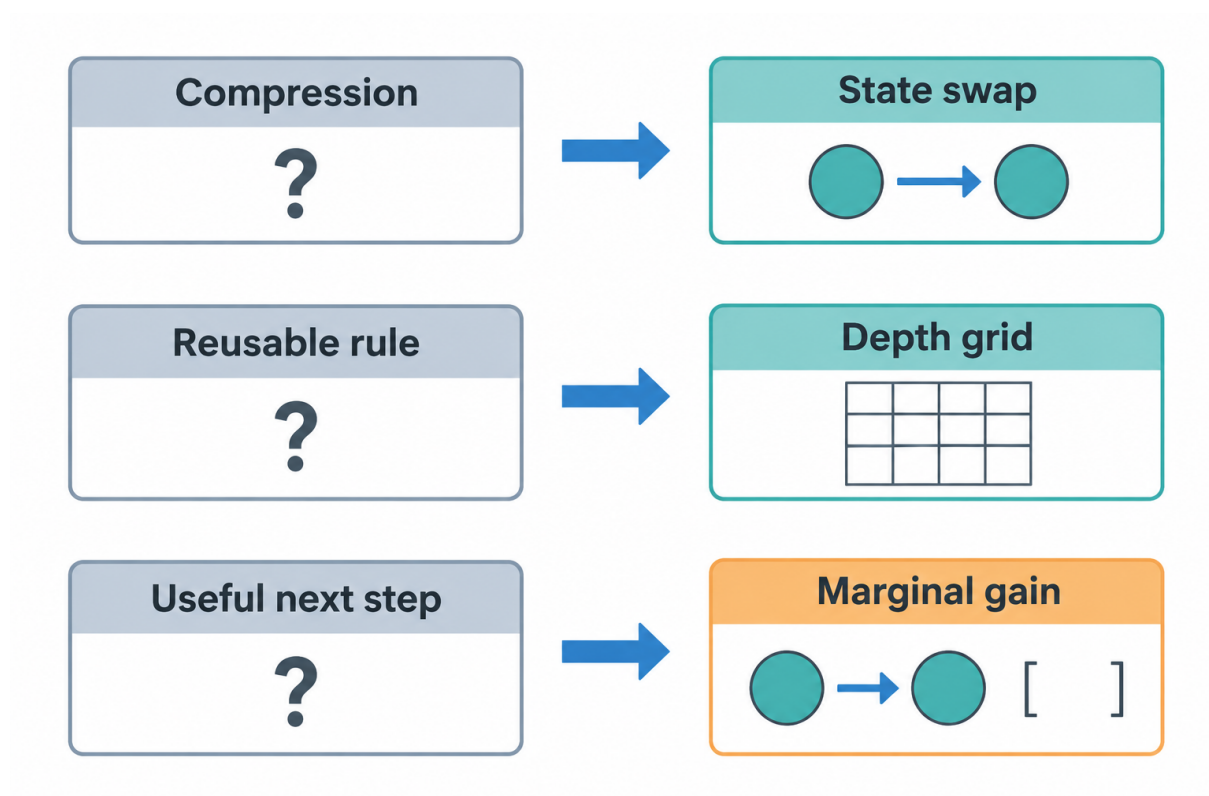


图 12.1: 概念示意：压缩、规则复用与下一步是否有益，分别需要状态置换、深度外推和边际收益等不同检验；图中未给出实验结果。

来源与边界：概念性教学图；依据 [14, 16, 23] 及本章证据整理，不构成新增实验结果。

旁路解释。若续算有效却随依赖深度迅速退化，支持局部可组合更新，但长程稳定性仍未解决。

相反，若短潜在轨迹能在低收缩任务中稳定实现多步反事实续算，且在移除原输入旁路后仍成立，这会增强“内部执行了可组合计算”的解释。即便如此，也不要求每一步与教师文本一一对应：学生可能学到更并行的算法。真正需要保留的是任务状态转移的可组合性，而非教师叙述的表面顺序。

12.3 研究二：共享规则、时间编码与深度外推

循环逼近理论指出共享权重对上下文输出映射的限制，并提出时间步调制；学习内化理论则依赖分阶段结构化训练。[13, 23] 二者共同提出一个尚未由这些结果直接回答的问题：显式告诉模型“现在是第几步”，究竟帮助它学会算法阶段，还是帮助它记住训练范围内的步骤表？

设计三种核心：不含步号的严格共享核心、使用有界或周期性步号编码的共享核心、使用可外推连续步号特征的共享核心。另设参数量配平的非共享模型。所有组保持输入表示、监督、宽度选择规则与训练计算预算一致；时间编码带来的新增参数必须计入。训练时分别采用固定深度和预先指定的深度分布，禁止根据测试曲线回头改变分布。

测试需要一个二维网格：横轴是任务所需依赖深度，纵轴是给模型的循环次数。输入长度另作分层，避免更长输入同时带来更多工作量和更难注意力检索。不能只沿“任务变难同时多给几步”的对角线测量，因为那样无法区分容量不足与调度不足。保留超出训练步数的区域，并将输入长度外推和循环次数外推分别报告。

若模型随循环增加只能改善训练范围内的任务，支持的是预算适配而非一般算法外推。若带步号模型在训练上更好、超范围却突然失效，说明步骤定位可能成为外推瓶颈；但还要检查编码数值范围与归一化是否失配。若无步号模型泛化更好但收敛更慢，则存在学习便利与规则复用之间的权衡。不能仅凭最终准确率差异断言哪种“更像算法”。

进一步做跨步状态交换：把第 r 步的状态放进第 $r + \Delta$ 步调用，同时分别保留原步号或更换步号。若输出主要随步号改变，模型依赖阶段调度；若主要随状态语义推进，模型可能学到较自治的更新规则。这个干预仍可能离开训练状态流形，应优先使用自然轨迹上语义相近的配对状态，并检查替换前后的状态范数与读出质量。

12.4 研究三：停机应预测可修复性，而不只是信心

HRM 与 TRM 把监督、递归和停止决策放在同一个训练系统里，但它们的梯度与调度设计不同。[14, 21] 一个值得单独研究的问题是：停止器能否预测“再算一次是否有益”？这不同于预测“当前答案是否正确”，因为当前错误可能不可修复，当前正确也可能被后续计算破坏。

先冻结一个已训练核心，固定最大预算 R ，在停止器训练与验证划分上分别保存每一步答案、正确性和成本，避免让停止器与核心同时变化。定义离线反事实收益

$$\Delta_r(x) = \mathbf{1}[\hat{y}_{r+1}(x) = y] - \mathbf{1}[\hat{y}_r(x) = y] - \lambda c_{r+1}(x). \quad (12.1)$$

其中 c_{r+1} 是归一化的下一步实测成本， λ 由应用预算事先确定。该量使用正确答案，只能用于训练标签、分析和离线 oracle；部署停止器不能看到它。对有随机采样的系统，还需估计条件期望及方差，单条轨迹不代表真实期望收益。这里的 Δ_r 只衡量下一步的局部收益，不能判定继续多步是否值得：连续几步都答错后才修正的轨迹，会被纯贪心规则过早截断。因此还应预先选定

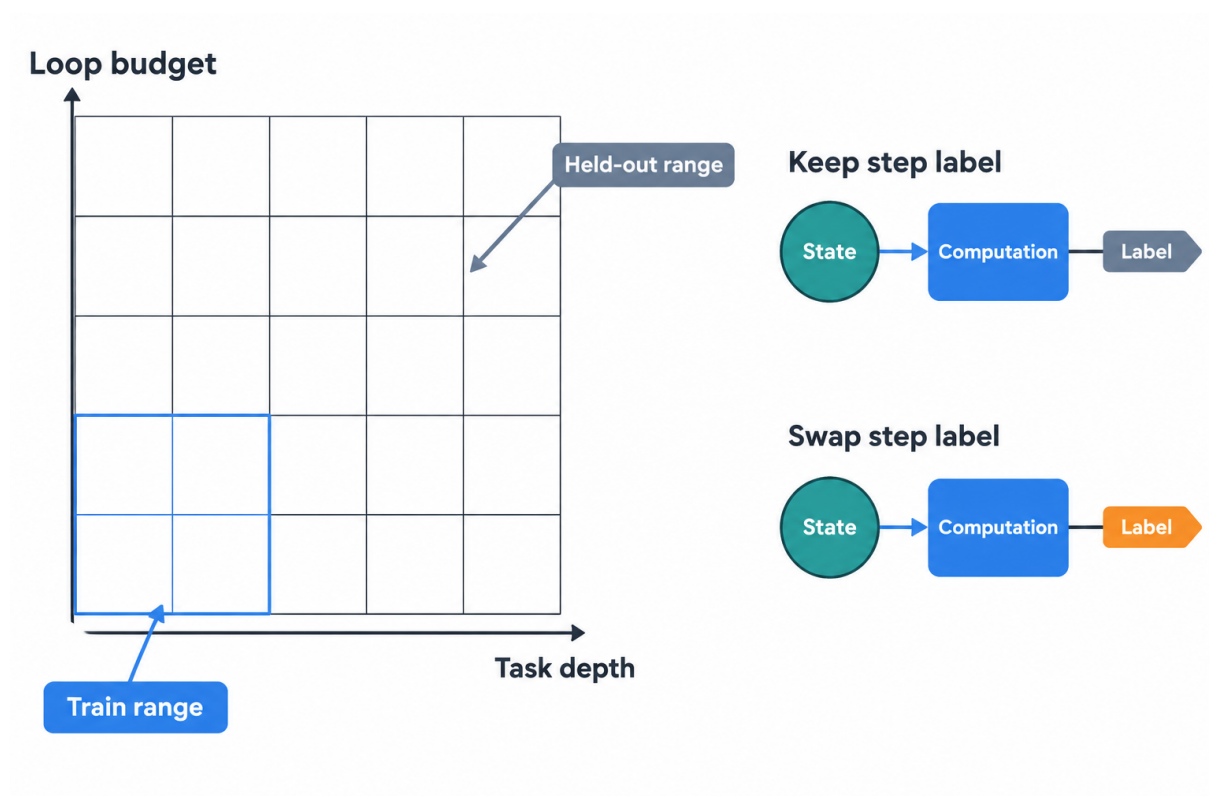


图 12.2: 概念示意：独立扫描任务依赖深度和循环预算，并在跨步状态置换时保持或更换步号条件。网格为空白实验设计；右侧 Label 为计算块附带的步号条件，不是生成输出。

来源与边界：概念性教学图；依据 [13, 23] 及本章证据整理，不构成新增实验结果。

若干展望长度，计算对应终点的正确性变化减去途中累计成本，并加入多步继续价值预测器；所有展望范围与阈值都在训练或验证阶段冻结。

比较固定步数、当前置信度阈值、状态变化阈值和收益预测器。用独立的停止器训练划分拟合收益预测器，在验证划分按同一预算选择各策略阈值，并在保留测试集上报告准确率、平均延迟、尾部延迟和错误类型。再与“事后知道哪一步最好”的 oracle 比较，但明确它只描述可改善空间。没有这个上界，停止器差可能是预测失败，也可能是核心轨迹本来就没有足够的可利用差异。

如果收益预测器在同分布中有效、在难度迁移后失效，下一步应研究校准与状态分布迁移，而不是先增加停机头容量。若所有可部署策略都接近固定步数，而 oracle 显著更好，说明轨迹有潜力但现有观测不足。若 oracle 也没有收益，则应返回核心训练研究，不应继续优化无收益的调度。这个分流使项目可以在无效方向上及时停止。

12.5 研究四：多模态循环是否真的重新利用证据

MCOUT 的潜在位置更新提供了多模态入口，HRM/TRM 则展示了结构化状态反复修正的价值。^[14, 18, 21] 它们启发一个更窄、也更容易检验的问题：在视觉证据已编码后，重复计算主要改善跨模态检索，还是改善已提取符号之间的组合？

使用可控图形场景生成器，使答案依赖少量可定位的物体关系，并提供同一场景的无损符号描述。设置仅初始注入视觉、每步读取固定视觉嵌入、按需重新编码局部图像三组；第三组的额外视觉编码计算必须计费。分别改变感知难度和关系链深度，形成两个独立轴。三组保持语言核心、监督和数据划分一致，分别报告匹配总推理预算与原生配置的结果。符号描述组用于定位感知表示接口相关的错误，而非作为更容易的排行榜替代；应按任务难度分层，并排除符号组准确率已达上限导致增益消失的天花板效应。

若固定视觉读取只在关系链变长时改善，支持组合计算解释；若重新编码主要修复细粒度视觉变化，支持感知瓶颈解释。若所有收益在用无损符号替换视觉后消失，应优先研究感知接口，而不是宣称新增了通用推理能力。用局部遮挡与特征替换验证相关区域的因果影响，并保留语言问题不变的视觉最小对，检验模型是否只是强化语言先验。

12.6 执行顺序与可积累的失败

上述研究不应一次性全部推进一个新架构。建议先完成第一项的小型诊断任务，建立逐步轨迹、成本与干预的共同日志格式；再在第二项中固定监督方案，只改变共享和步号。只有核心轨迹存在可修复错误时，才进入第三项学习停止；最后把已通过诊断的机制搬到多模态设置，并允许迁移失败。这样的顺序让每次新增复杂度都对应一个明确证据缺口。

每个阶段都应预先写出主要指标、最低可辨认效应、样本生成规则、随机种子和预算上限，用试验性方差估计决定正式样本量，而非事后挑出最好种子。负结果也要保存原始轨迹：是状态未形成、形成但未使用、使用但无法外推，还是有效却成本过高？这些失败指向不同下一步，不能统一标成“模型规模不够”。

最终值得积累的不是某个模型拥有多少“思维步”，而是一组带条件的可复核认识：在哪些任务结构、监督信号、状态容量和预算下，重复内部计算能带来稳定收益；这种收益如何被干预支持，又在哪些迁移中失效。当这些条件能够写清楚，隐状态推理与循环 Transformer 才从引人联想的名称，变成可以持续改进的研究对象。

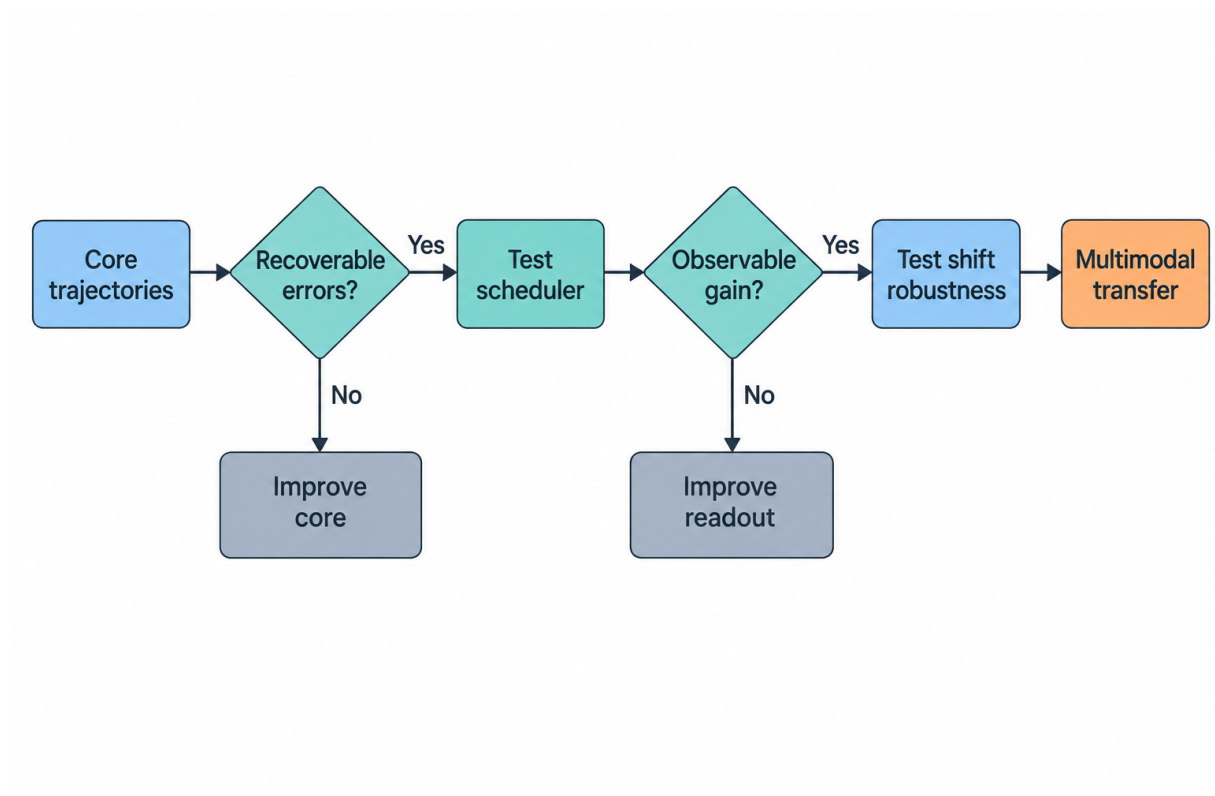


图 12.3: 概念示意：先判断核心轨迹是否存在可修复错误，再测试调度器与分布迁移；没有收益时返回相应机制研究。箭头表示研究顺序，非已获得的成功结果。

来源与边界：概念性教学图；依据 [14, 16, 18] 及本章证据整理，不构成新增实验结果。

术语速查

显式思维链 (CoT) 以可读词元生成的中间过程；可读性不自动保证忠实性。

连续思维位置 将连续向量作为序列中的后续输入位置；须说明反馈接口与可见缓存。

循环深度 对同一组位置反复应用共享核心；深度步与新增序列位置不同。

权重共享 多次计算使用同一组参数，不代表计算、激活或 KV 缓存免费。

前奏 / 核心 / 终曲 循环语言模型中输入编码、迭代更新、输出解码的三个功能部分。

停机策略 根据可获得信息决定实际停止步骤；与事后使用正确答案选择最佳步骤的 oracle 不同。

截断反向传播 执行的前向路径可以长于保留梯度的路径；截断不删除已经执行的计算。

机制证据 观察、解码与干预回答不同问题；解码成功本身不证明模型使用该信息。

预算匹配 明确匹配的资源与测量边界，例如训练计算、推理 FLOPs、串行调用、内存和实际延迟。

参考文献

- [1] Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyouon Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-Recursions: Learning Dynamic Recursive Depths for Adaptive Token-Level Computation. arXiv preprint, 2025. URL <https://arxiv.org/abs/2507.10524>. arXiv:2507.10524.
- [2] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep Equilibrium Models. arXiv preprint, 2019. URL <https://arxiv.org/abs/1909.01377>. arXiv:1909.01377.
- [3] Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end Algorithm Synthesis with Recurrent Networks: Logical Extrapolation Without Overthinking. arXiv preprint, 2022. URL <https://arxiv.org/abs/2202.05826>. arXiv:2202.05826.
- [4] Alireza Amiri Bavandpour, Xinting Huang, Mark Rofin, and Michael Hahn. Lower Bounds for Chain-of-Thought Reasoning in Hard-Attention Transformers. Primary source, n.d. URL <https://proceedings.mlr.press/v267/bavandpour25a.html>.
- [5] Jeffrey Cheng and Benjamin Van Durme. Compressed Chain of Thought: Efficient Reasoning Through Dense Representations. arXiv preprint, 2024. URL <https://arxiv.org/abs/2412.13171>. arXiv:2412.13171.
- [6] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal Transformers. arXiv preprint, 2018. URL <https://arxiv.org/abs/1807.03819>. arXiv:1807.03819.
- [7] Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. Implicit Chain of Thought Reasoning via Knowledge Distillation. arXiv preprint, 2023. URL <https://arxiv.org/abs/2311.01460>. arXiv:2311.01460.
- [8] Yuntian Deng, Yejin Choi, and Stuart Shieber. From Explicit CoT to Implicit CoT: Learning to Internalize CoT Step by Step. arXiv preprint, 2024. URL <https://arxiv.org/abs/2405.14838>. arXiv:2405.14838.
- [9] 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, 2025. URL <https://arxiv.org/abs/2502.05171>. arXiv:2502.05171.

- [10] Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training Language Models With Pause Tokens. arXiv preprint, 2023. URL <https://arxiv.org/abs/2310.02226>. arXiv:2310.02226.
- [11] Alex Graves. Adaptive Computation Time for Recurrent Neural Networks. arXiv preprint, 2016. URL <https://arxiv.org/abs/1603.08983>. arXiv:1603.08983.
- [12] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training Large Language Models to Reason in a Continuous Latent Space. arXiv preprint, 2024. URL <https://arxiv.org/abs/2412.06769>. arXiv:2412.06769.
- [13] Yixiao Huang, Hanlin Zhu, Zixuan Wang, Jiantao Jiao, Stuart Russell, Somayeh Sojoudi, and Song Mei. Transformers Provably Learn to Internalize Chain-of-Thought. arXiv preprint, 2026. URL <https://arxiv.org/abs/2605.28600>. arXiv:2605.28600.
- [14] Alexia Jolicoeur-Martineau. Less is More: Recursive Reasoning with Tiny Networks. arXiv preprint, 2025. URL <https://arxiv.org/abs/2510.04871>. arXiv:2510.04871.
- [15] Hsun-Yu Kuo, El Mahdi Chayti, Patrik Reizinger, Wieland Brendel, and Martin Jaggi. Stabilizing Extrapolation in Looped Transformers via Learned Stochastic Stopping. arXiv preprint, 2026. URL <https://arxiv.org/abs/2606.29983>. arXiv:2606.29983.
- [16] Jia Liang and Liangming Pan. Do Latent-CoT Models Think Step-by-Step? A Mechanistic Study on Sequential Reasoning Tasks. arXiv preprint, 2026. URL <https://arxiv.org/abs/2602.00449>. arXiv:2602.00449.
- [17] William Merrill and Ashish Sabharwal. The Expressive Power of Transformers with Chain of Thought. arXiv preprint, 2023. URL <https://arxiv.org/abs/2310.07923>. arXiv:2310.07923.
- [18] Tan-Hanh Pham and Chris Ngo. Multimodal Chain of Continuous Thought for Latent-Space Reasoning in Vision-Language Models. arXiv preprint, 2025. URL <https://arxiv.org/abs/2508.12587>. arXiv:2508.12587.
- [19] Machel Reid, Edison Marrese-Taylor, and Yutaka Matsuo. Subformer: Exploring Weight Sharing for Parameter Efficiency in Generative Transformers. arXiv preprint, 2021. URL <https://arxiv.org/abs/2101.00234>. arXiv:2101.00234.
- [20] Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. CODI: Compressing Chain-of-Thought into Continuous Space via Self-Distillation. arXiv preprint, 2025. URL <https://arxiv.org/abs/2502.21074>. arXiv:2502.21074.
- [21] Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical Reasoning Model. arXiv preprint, 2025. URL <https://arxiv.org/abs/2506.21734>. arXiv:2506.21734.
- [22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv preprint, 2022. URL <https://arxiv.org/abs/2201.11903>. arXiv:2201.11903.

- [23] Kevin Xu and Issei Sato. On Expressive Power of Looped Transformers: Theoretical Analysis and Enhancement via Timestep Encoding. arXiv preprint, 2024. URL <https://proceedings.mlr.press/v267/xu25x.html>. arXiv:2410.01405.
- [24] Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. SoftCoT++: Test-Time Scaling with Soft Chain-of-Thought Reasoning. arXiv preprint, 2025. URL <https://arxiv.org/abs/2505.11484>. arXiv:2505.11484.
- [25] Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped Transformers are Better at Learning Learning Algorithms. arXiv preprint, 2023. URL <https://arxiv.org/abs/2311.12424>. arXiv:2311.12424.
- [26] Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D. Goodman. Quiet-STaR: Language Models Can Teach Themselves to Think Before Speaking. arXiv preprint, 2024. URL <https://arxiv.org/abs/2403.09629>. arXiv:2403.09629.
- [27] Boyi Zeng, Shixiang Song, Siyuan Huang, Yixuan Wang, He Li, Ziwei He, Xinbing Wang, Zhiyu Li, and Zhouhan Lin. Pretraining Language Models to Ponder in Continuous Space. arXiv preprint, 2025. URL <https://arxiv.org/abs/2505.20674>. arXiv:2505.20674.
- [28] Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling Latent Reasoning via Looped Language Models. arXiv preprint, 2025. URL <https://arxiv.org/abs/2510.25741>. arXiv:2510.25741.

术语索引

Adaptive computation, 41

CCoT, 17

Chain-of-Thought, 6

Coconut, 17

CODI, 11

Continuous feedback, 35

curriculum learning, 11

Halting, 41

knowledge distillation, 11

KV cache, 29

latent reasoning, 1

Overthinking, 41

pause tokens, 6

Quiet-STaR, 6

Recurrent depth, 23

Recurrent pretraining, 29

SoftCoT++, 17

State interface, 35

Universal Transformer, 23

隐状态, 1