Self-AC: Self-Actor-Critic 后训练方法 用于 LLM Agent 多回合强化学习

相比于 GRPO Post-Training:

1. 只加少量参数 (训完可以扔掉)
2. 推理成本完全相同、训练成本几乎一致
3. 回合级别 Credit-Assign
4. 免费得到 Value Function

郭一岑 @ 上海交通大学数学科学学院

引言：为什么要对 LLM Agent 做多回合 RL

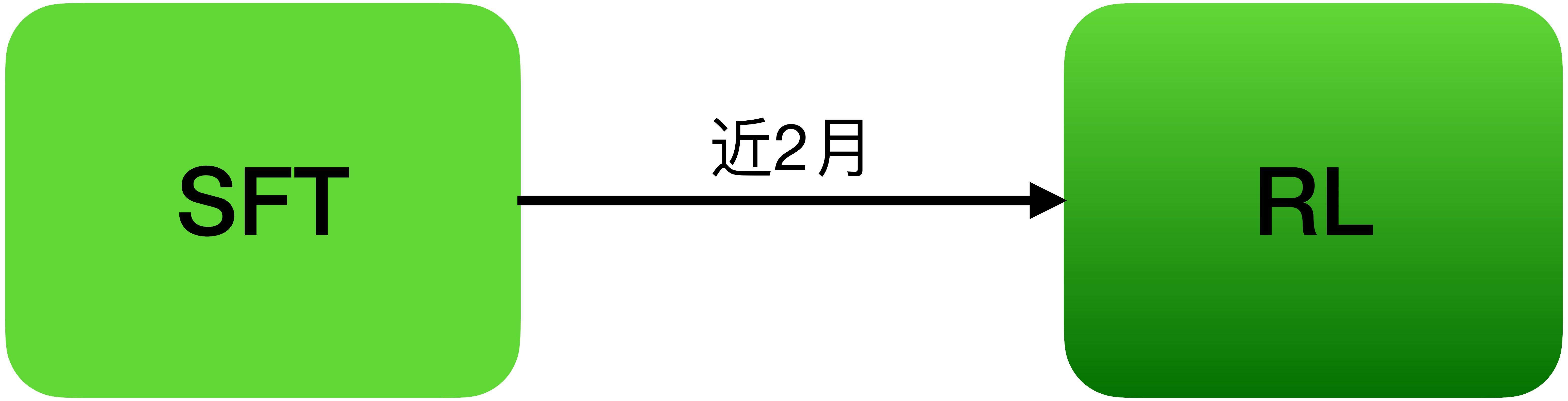
很多 LLM Agent 都是 RL Agent

特点: 程序性奖励函数、多步Credit-Assign

特点: 多模态、世界知识、自带推理能力

Post-Training 现状：范式正在转变

Post-Training：把 GPT 这样的模型变为 Application-Specific 模型的过程



手搓交互数据

有Env和Reward就能无限训

Post-Training 现状：GRPO

所有动作都要**塞到一回合里面去**

$$\mathcal{J}(\theta) = \mathbb{E}_{x\sim\mathcal{D},\{y_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|x)}$$
$$\frac{1}{G}\sum_{i=1}^{G}\left[\min\left(\frac{\pi_\theta(y_i|x)}{\pi_{\theta_{\text{old}}}(y_i|x)}A_i, \text{clip}\left(\frac{\pi_\theta(y_i|x)}{\pi_{\theta_{\text{old}}}(y_i|x)}, 1-\epsilon, 1+\epsilon\right)A_i\right) - \beta\mathbb{D}_{\text{KL}}\left(\pi_\theta||\pi_{\theta_{\text{ref}}}\right)\right]$$

现状可能的矛盾点:

1. **现实需要多回合:** 更细的 Credit-Assign
2. **GRPO不原生支持多回合:** 效果未知
3. **Actor-Critic则很昂贵:** Critic Model 大量参数

为什么需要多回合

O1中RL的可能行为空间：“思考因子（Thought-Factor）”离散行为空间

It seems that the ciphertext words are exactly twice as long as the plaintext words.

(10 vs 5, 8 vs 4, 4 vs 2, 8 vs 4)

Idea: Maybe we need to take every other letter or rebuild the plaintext from the ciphertext accordingly.

Let's test this theory.

提出猜测

Sum: 15 +25 = 40

But 'T' is 20.

Alternatively, perhaps subtract: 25 -15 = 10.

No.

否定猜测

Alternatively, perhaps combine the numbers in some way.

Alternatively, think about their positions in the alphabet.

Alternatively, perhaps the letters are encrypted via a code.

提出候选方案

Let's list them properly.

Wait, earlier I missed some letters there.

Let's re-express the sixth word letters:

m y n z n v a a t z a c d f o u l x x z

自我发现&修正错误

So the user is requesting a bash script that can take a string representing a matrix, such as '[1,2],[3,4],[5,6]' and output its transpose, in the same format.

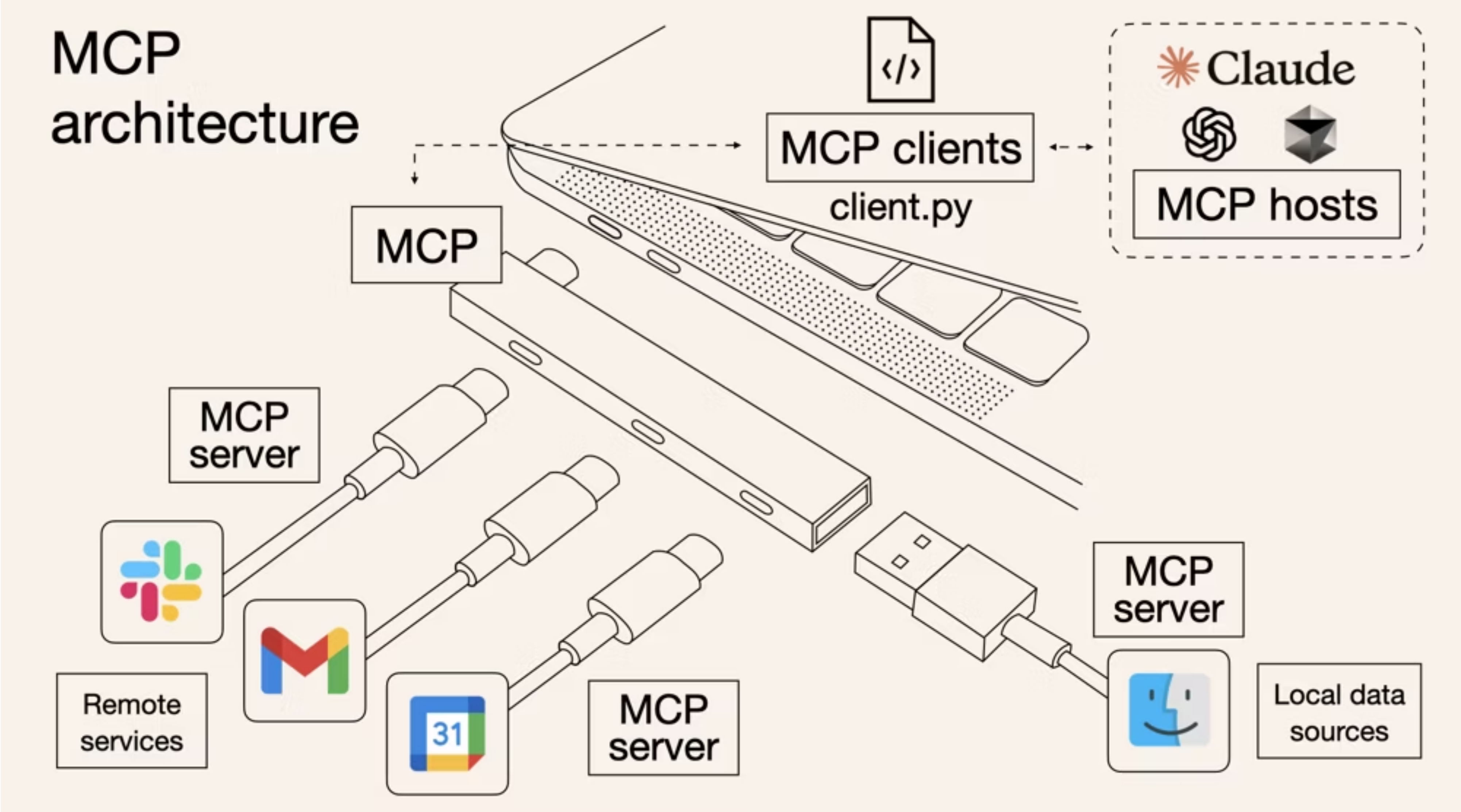
澄清目标

Approach:

- Parse the input string to extract the matrix elements.
- Build the matrix as an array of arrays.
- Transpose the matrix.
- Output the transposed matrix in the same format.

拆解子任务（Flan

o1：它…是不是按回合在思考?



MCP工具: 每次工具调用, 都是自然的回合

活动 25 个源

郊区的差异。

Searched for 2023年东京各区1R公寓租金

还在考虑GaijinPot博客的内容是否包含相关信息。

读取 blog.gaijinpot.com

正在点击[3]，发现东京都23区与市中心的房租起伏很大。

读取 tokyoportfolio.com

正在分析东京中心部与其他行政区的共享房租金差异，并询问初始费用和紧急备用金的建议。

我在查看如JASSO和东京专门学校协会等资源，寻求东京单人居住的平均水电费标准。

读取 tsk.or.jp

个人租房每月花费通常在50k-80k日元之间，另加约10k的水电费用。合租公寓的水电价格可能固定在20k左右。

一人间公寓在东京市中心的月租金范围为60,000-120,000日元。合租房的单人房/月租金为40,000-70,000日元。郊区/近郊的租金可能在50,000-80,000日元之间。

查看GaijinPot的图片搜索或截图，探索广告价格波动。最底价可能为Adachi的1R 65k，高价则为Minato或Shibuya的1R 100k或更高。

读取 ejable.com

我正在跨校区课程政策和费用方面做一些研究。**

Rakuten和Docomo性价比高的套餐，平均费用大约为3,000日元。分享屋和公寓的互联网费用也有所不同。

活动 25 个源

到¥150,000，多用于食物购物、交通和健康保险。

读取来自 romancing-japan.com 的更多内容

我正在查看博客中的食品和通讯费用数据，参照JASSO官方数据或其他可靠来源了解更详细的信息。

读取来自 romancing-japan.com 的更多内容

正在查看MEXT或其他官方对日本学生通话费用的平均支出情况。

搜索了留学生通信费和手机网络平均费用

洞悉台湾博客关于日本学习费用的细节。

读取 djbcard.com

比较多个网络选项的费用可能因计划不同而异。5,000至8,000日元对于手机套餐、口袋Wi-Fi和家庭网路来说可能是平均费用。

读取 romancing-japan.com

娱乐和购物成本没有具体数字，可能将其整合到“其他”类别中。参考提到的120-150k月度预算涵盖了一部分。

读取 studyinjapan.go.jp

探讨东京或办公室常见开销

读取来自 studyinjapan.go.jp 的更多内容

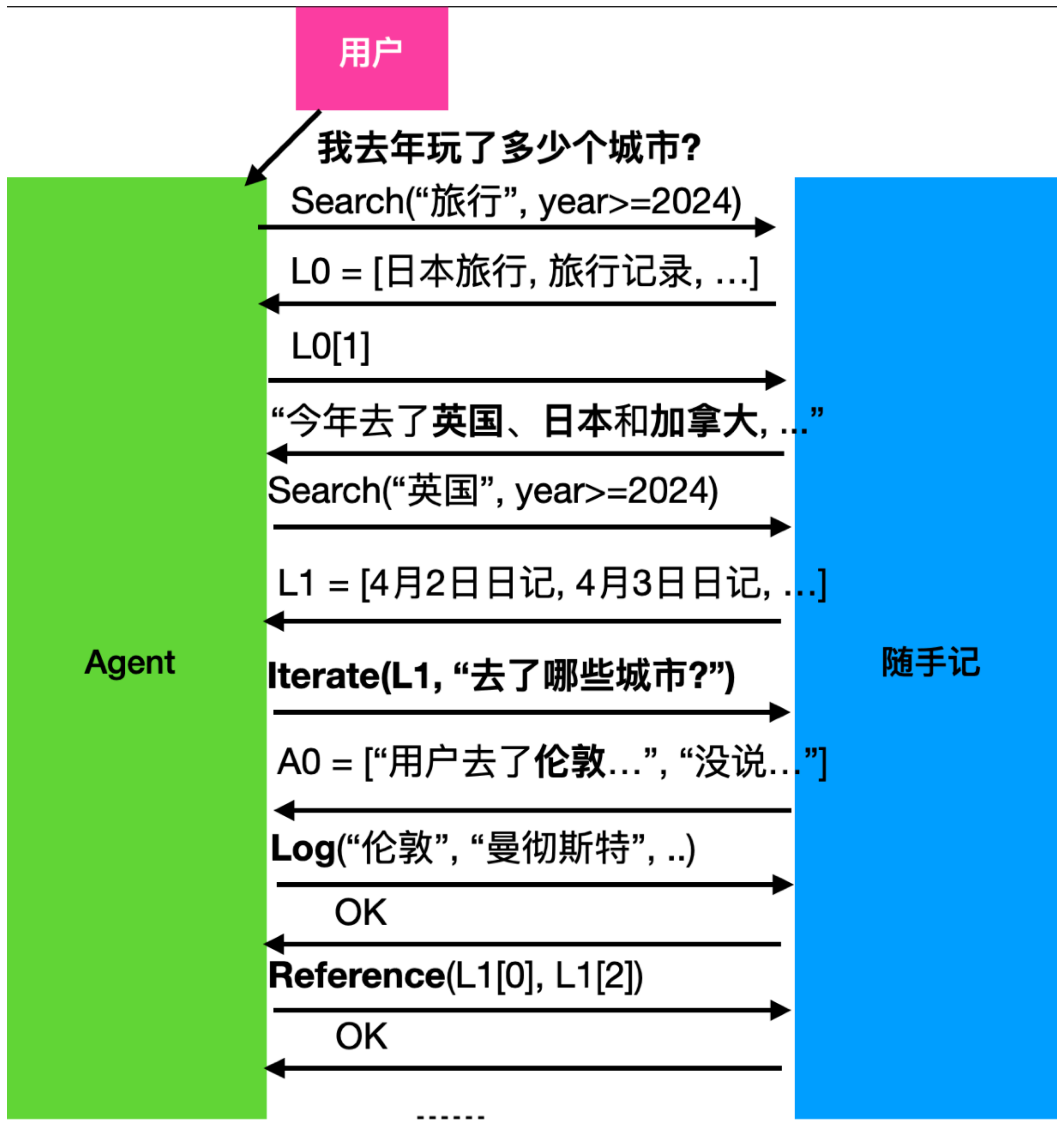
东京的房租成本为57万日元，全国均价为41万日元。东京的食物、通学、学习成本分别为37万、6万、56万日元。全国均价分别为32万、4万、52万日元。

东京生活的留学生每月生活费在约120万至180万日元之间，具体取决于个人消费习惯和住宿选择。

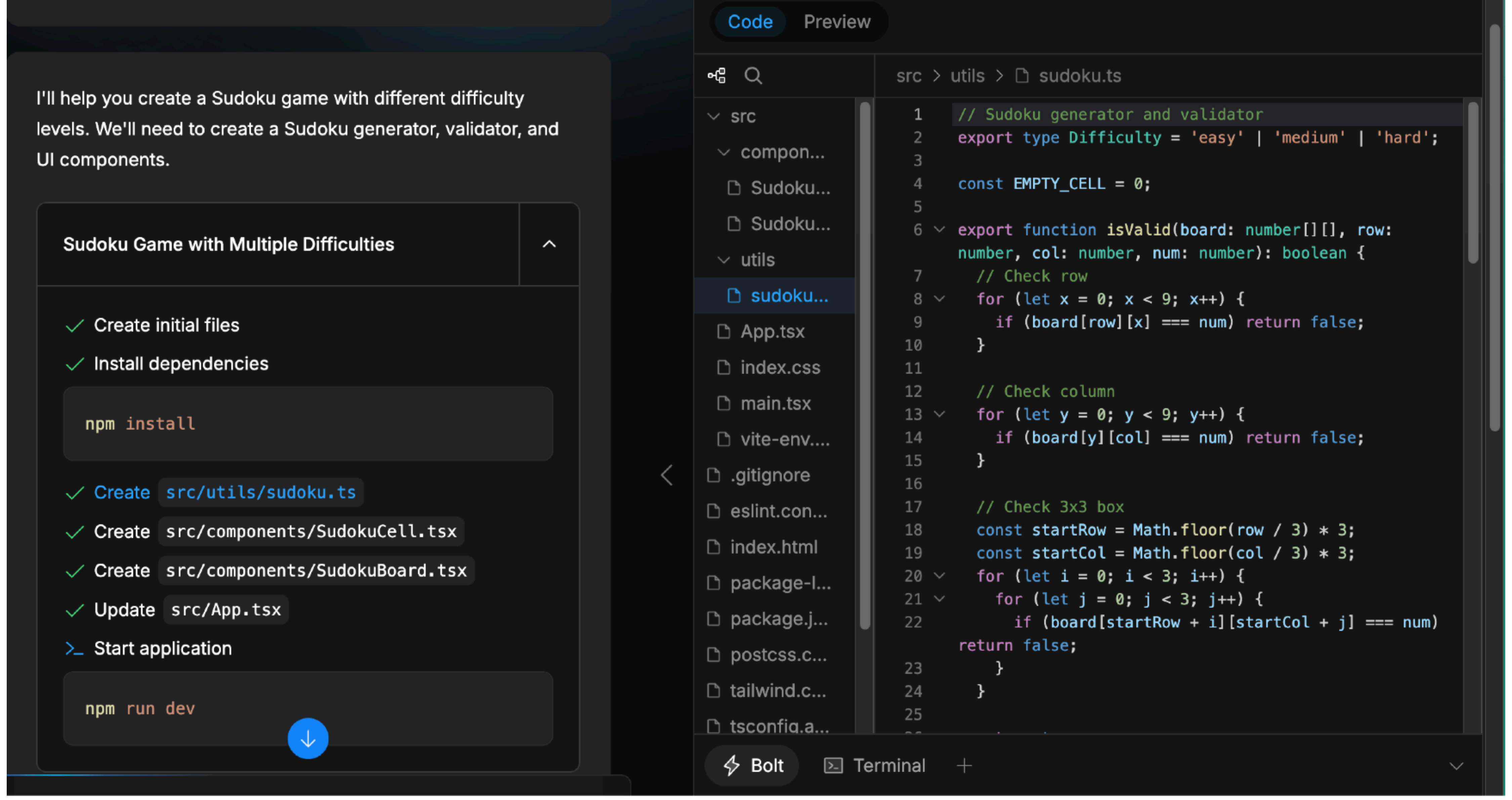
Deep Research:

每个页面交互都是一回合

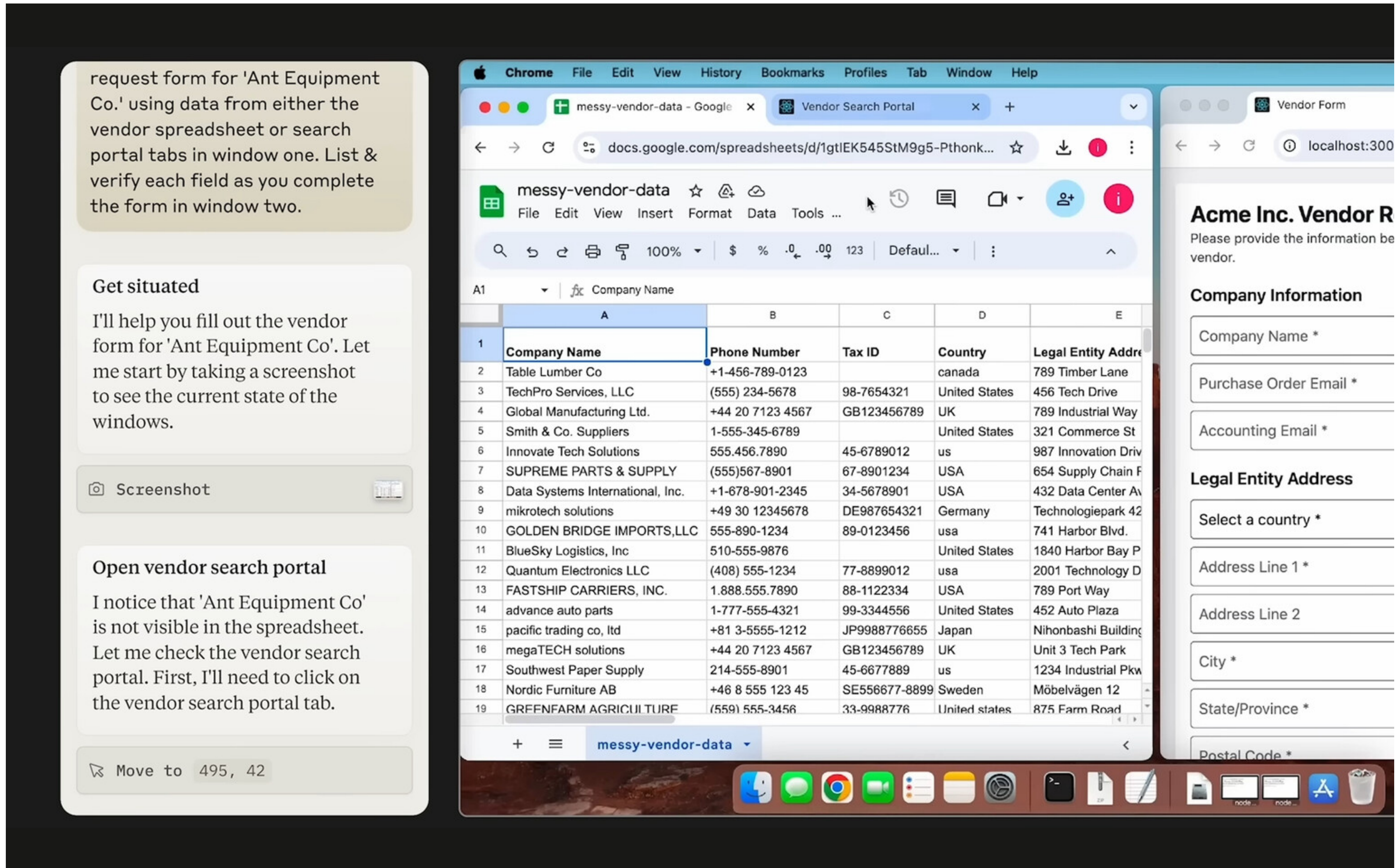
为什么需要多回合



检索Agent: 每个工具交互都是一回合



Coding Agent: 每次代码编辑都是一回合



Computer Use:

分回合才能
提高规划能力

正文：Self-AC 的架构和训练

Self-AC 的核心思考

1. **Critic Model 可以和 Actor Model 共享模型:**

   我们的 Actor Model 本来就需要承担很多任务. 对于一个 SearchAgent 来说,
   它需要搜索、点击、页内查找、保存、回退.
   如果它本身就需要做好这些任务, 那么再增加一个 Critic 任务也不过分

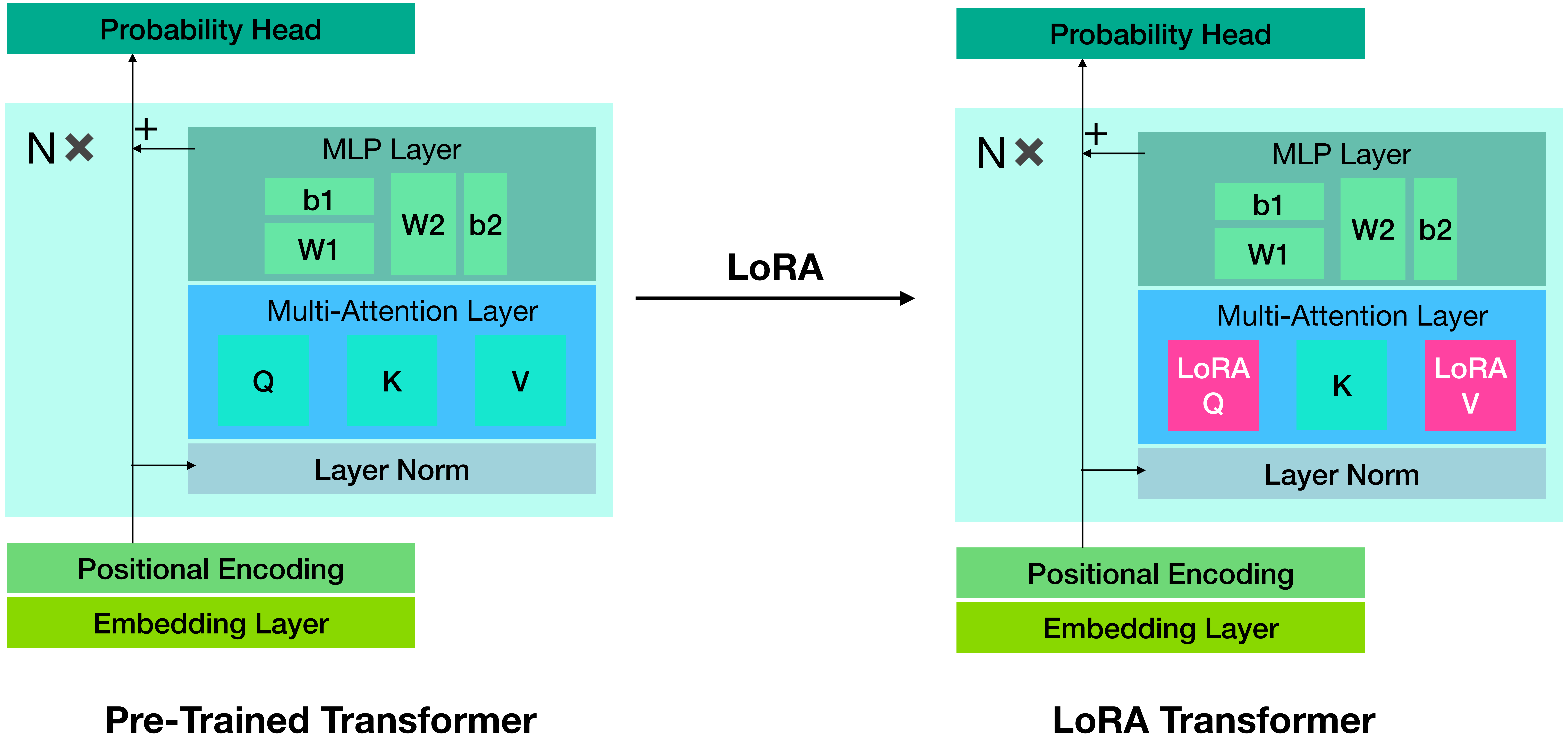
2. **需要用 Prompt 引出 Critic 能力:**

   模型需要意识到: 它正在评价某件事
   否则它更可能输出 Actor 的下一个行动 (我们不会真的让它输出内容, 但它需要一个隔离的语意空间)
   我们要用 Prompt 来区分 Actor/Critic 角色

3. **集向量技术 + ShadowPrompt 技术提 1 阶训练速度: (?)**

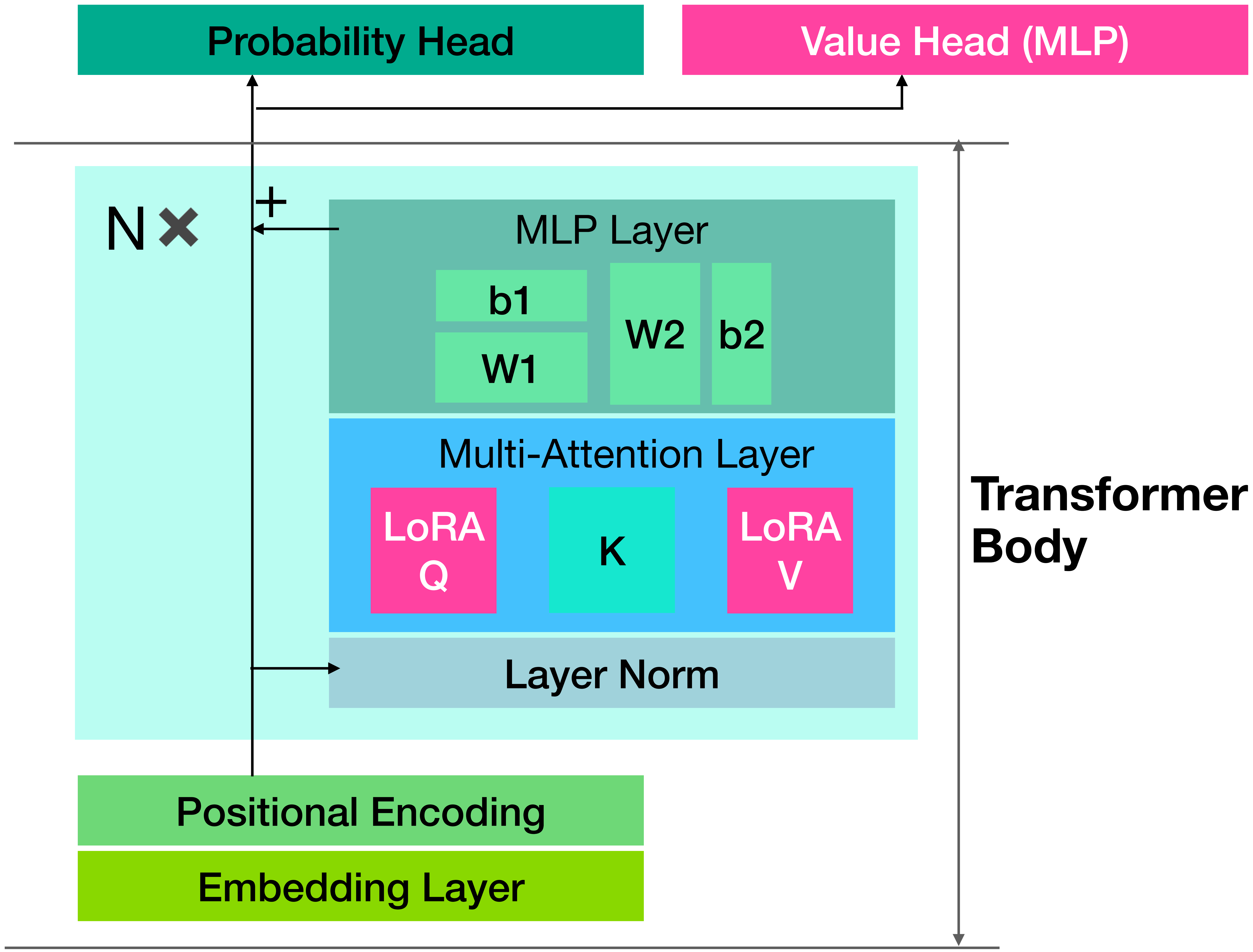
   一般的 Actor-Critic 一次训练一个回合, 而 GRPO 一次训练一集
   因此 GRPO 比一般的 Critic-Actor 快 avg_s 倍，avg_s 是每集的平均回合数
   基于集向量 + ShadowPrompt 技术, Self-AC 一次训练一集, 一个 N-Batch 训练 N 集

Self-AC 的模型架构 Part 1



Self-AC 的模型架构 Part 2

Self-AC 的模型架构 Part 3: Actor & Critic



Self-AC Transformer

Actor(a|s)=Transformer(a|s)
=(ProbabilityHead∘TransformerBody)(a|s)

Critic(s)=ValueHead(TransformerBody(s+**cp**)[-1])

cp= <eos>

System: Critic Mode! Evaluate the current state with a single expressive word:<eos>

Assistant:

Self-AC 的模型架构 Part 4: 例子

Actor ✅

State

在东京、大阪、京都各旅行3天的总预算?

<think>让我们一步步思考…</think>
<action>Search(东京旅行预算)</action>

1. Wikipedia: …
2. 微信公众号: …
3. 知乎: ...

Action

<think>让我们一步步思考…</think>
<action>Click(1)</action>

朴素 Critic ❌

State

在东京、大阪、京都各旅行3天的总预算?

<think>让我们一步步思考…</think>
<action>Search(东京旅行预算)</action>

1. Wikipedia: …
2. 微信公众号: …
3. 知乎: ...

Action

<think>让我们一步步思考…</think>
<action>Click(1)</action>

MLP

Value

<t….<think?

Self-AC Critic ✅

State

在东京、大阪、京都各旅行3天的总预算?

<think>让我们一步步思考…</think>
<action>Search(东京旅行预算)</action>

1. Wikipedia: …
2. 微信公众号: …
3. 知乎: ...

Shadow Prompt (对后文不可见)

SYSTEM Critic Mode! Evaluate the current state with a single expressive word.

Promising

MLP

Value

0.9

Action

<think>让我们一步步思考…</think>
<action>Click(1)</action>

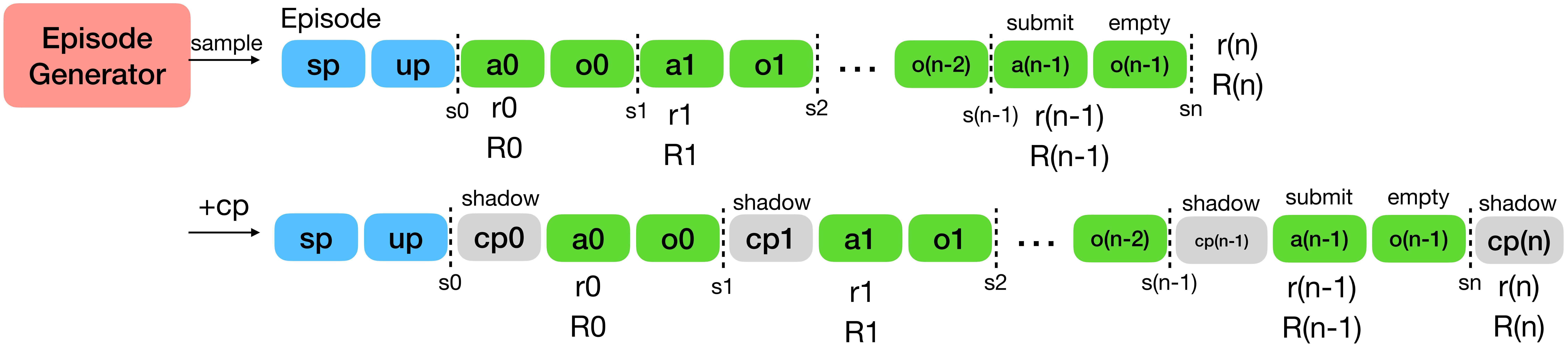
Self-AC 的训练说明 Part 0: ReAct场景 — 训练数据长啥样

sp	system: 你是一个查维基百科的高手, 现在请帮用户在维基百科上查找资料并保存, 你的回复格式是…<eos>
up	user: 二战死亡的说英语的总人数?<eos>assistant:
cp0	<eos>system:Critic Mode! Evaluate the current state with a single expressive word:<eos>assistant:
a0	<think>让我们一步步思考…</think><action>NAVIGATE(二战死亡的说英语人数)</action><eos>
o0	tool:不存在这个页面, 相似页面:…<eos>assistant:
cp1	<eos>system:Critic Mode! Evaluate the current state with a single expressive word:<eos>assistant:
a1	<think>让我们一步步思考…</think><action>NAVIGATE(二战死亡人数)</action><eos>
o1	tool:二战死亡人数—Wikipedia:自由的百科全书 ……<eos>assistant:
cp1	<eos>system:Critic Mode! Evaluate the current state with a single expressive word:<eos>assistant:
a2	<think>让我们一步步思考…</think><action>SAVE_LINE_IDS(17-19, 108-145)</action><eos>
o2	tool:17-19: 保存成功, 共计3行\n108-145:保存成功, 共计38行<eos>assistant:
cp2	<eos>system:Critic Mode! Evaluate the current state with a single expressive word:<eos>assistant:
…	
a(n-1)	<think>让我们一步步思考…</think><action>SUBMIT(二战死亡的说英语总人数为…, 其中……)</action><eos>
o(n-1)	
cp(n)	<eos>system:Critic Mode! Evaluate the current state with a single expressive word:<eos>assistant:

Self-AC 的训练说明 Part 0: CoT场景 — 训练数据长啥样

sp	system: 你是一个数学天才, 帮用户解决数学问题. 你的思考由很短的“思考因子”组成, 用\n\n分割思考因子<eos>
up	user: 计算1+1+1+1<eos>assistant:让我们一步步思考:
cp0	<eos>system:Critic Mode! Evaluate the current state with a single expressive word:<eos>assistant:
a0	首先, 根据加法交换律, 1+1=1+1 \n\n
o0	
cp1	<eos>system:Critic Mode! Evaluate the current state with a single expressive word:<eos>assistant:
a1	其次, 根据0元素的性质, 1+1=0+1+1 \n\n
o1	
cp1	<eos>system:Critic Mode! Evaluate the current state with a single expressive word:<eos>assistant:
a2	等等! 既然0+0=0, 那么1+1是不是等于… \n\n
o2	
cp2	<eos>system:Critic Mode! Evaluate the current state with a single expressive word:<eos>assistant:
…	
a(n-1)	1+1+1+1=4 <eos>
o(n-1)	
cp(n)	<eos>system:Critic Mode! Evaluate the current state with a single expressive word:<eos>assistant:

Self-AC 的训练说明 Part 1: 生成Episode



Tokens Record

seq =[“Sys”, “tem”, “: ”, ……, “, I”, <eos>, “Sys”, “tem”, “: ”, ……, “ think”, ….]

seq_next=[“tem”, “: ”, …… , “ *think*”, “Sys”, “tem”, “: ”, ……, “ *think*”, “that”, ….]

pos =[0 , 1 , 2 , …….., 9 , 10 , 11 , 12, 13 , … , 28 , *10* , ….]

Episode-Vectors: **(vectors of length n or n+1)**

a_start = [a0_start, a1_start, …, a(n-1)_start]

a_end = [a0_end, a1_end, …, a(n-1)_end]

cp_start = [cp0_start, cp1_start, …, cp(n)_start]

cp_end = [cp0_end, cp1_end, …, cp(n)_end]

pi_theta_old = [pi_theta_old_0, pi_theta_old_1, …, pi_theta_old(n-1)]

r = [r0, r1, …, r(n)]

R = [R0, R1, …, R(n)]

Self-AC 的训练说明 Part 1 (Annotated)

Tokens Record

seq =[“Sys”, “tem”, “: ”, ……, “, I”, <eos>, “Sys”, “tem”, “: ”, ……, “ think”, ….] Token序列

seq_next=[“tem”, “: ”, …… , “ *think*”, “Sys”, “tem”, “: ”, ……, “ *think*”, “that”, ….] 应预测Token序列

跨ShadowPrompt传递

pos =[0 , 1 , 2 , …….., 9 , <u>10</u> , 11 , 12, 13 , … , 28 , <u>*10*</u> , ….] 位置序列

注意ShadowPrompt位置编码被复用

Episode-Vectors: **(vectors of length n or n+1)** p-集向量: 每个分量记录对应回合的p性质

a_start = [a0_start, a1_start, …, a(n-1)_start] 动作起始位置(包含)

a_end = [a0_end, a1_end, …, a(n-1)_end] 动作结束位置(包含)

cp_start = [cp0_start, cp1_start, …, cp(n-1)_start] Critic Prompt起始位置(包含)

cp_end = [cp0_end, cp1_end, …, cp(n-1)_end] Critic Prompt终止位置(包含)

pi_theta_old = [pi_theta_old_0, pi_theta_old_1, …, pi_theta_old(n-1)] Rollout时动作概率

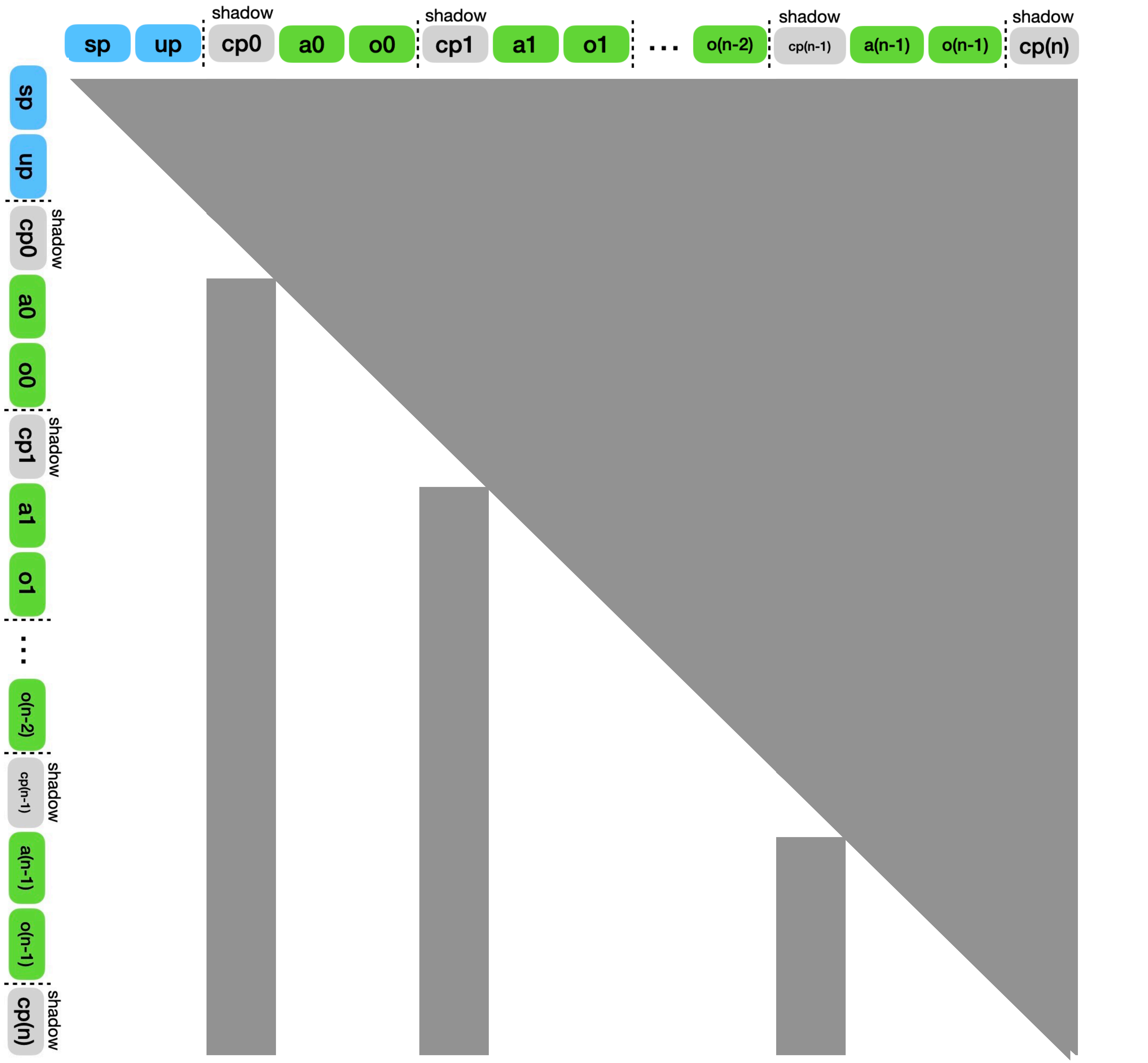
r = [r0, r1, …, r(n)] r_k: 从s_k到s_{k+1}的奖励量

R = [R0, R1, …, R(n)] R_k: 从s_k开始Rollout的奖励量(带衰减因子)

Self-AC 的训练说明 Part 2: Transformer Pass

Self-AC 的训练说明 Part 2.5: 注意力掩码

Attention Mask



Self-AC 的训练说明 Part 3: Critic Loss

(从现在开始考虑 BatchSize)

$$V=\begin{bmatrix}\text{Value Episode-Vector}\\ \text{Value Episode-Vector}\\ \text{Value Episode-Vector}\\ \cdots\\ \text{Value Episode-Vector}\end{bmatrix}\overset{\text{eg.}}{=}\begin{bmatrix} v0 & v1 & v2 & 0 & 0 & 0\\ v0 & v1 & 0 & 0 & 0 & 0\\ v0 & v1 & v2 & v3 & v4 & 0\\ & & \cdots & & & \\ v0 & v1 & v2 & v3 & 0 & 0\end{bmatrix}$$

$$r=\begin{bmatrix}\text{Reward Episode-Vector}\\ \text{Reward Episode-Vector}\\ \text{Reward Episode-Vector}\\ \cdots\\ \text{Reward Episode-Vector}\end{bmatrix}\overset{\text{eg.}}{=}\begin{bmatrix} r0 & r1 & r2 & 0 & 0 & 0\\ r0 & r1 & 0 & 0 & 0 & 0\\ r0 & r1 & r2 & r3 & r4 & 0\\ & & \cdots & & & \\ r0 & r1 & r2 & r3 & 0 & 0\end{bmatrix}$$

TD(1): $V = r + (\lambda\, V \overset{\text{左移算子}}{<<} 1)$

TD(2): $V = r + (\lambda\, r<<1) + (\lambda^2\, V<<2)$

TD(n): $V = r + \ldots + (\lambda^{n-1}\, r <<(n-1)) + (\lambda^n\, V<<n)$

$$\text{Loss\_TD(n)} = \left\| V - [r + \ldots + (\lambda^{n-1}\, r <<(n-1)) + (\lambda^n\, V<<n)] \right\|_2^2$$

$$\text{Loss\_Critic} = (\text{Loss\_TD}(1) + \ldots + \text{Loss\_TD}(5))/5$$

Self-AC 的训练说明 Part 4: Actor Loss

$$V=\begin{bmatrix}\text{Value Episode-Vector}\\ \text{Value Episode-Vector}\\ \text{Value Episode-Vector}\\ \cdots\\ \text{Value Episode-Vector}\end{bmatrix} \overset{\text{eg.}}{=} \begin{bmatrix} v0 & v1 & v2 & 0 & 0 & 0\\ v0 & v1 & 0 & 0 & 0 & 0\\ v0 & v1 & v2 & v3 & v4 & 0\\ & & \cdots & & & \\ v0 & v1 & v2 & v3 & 0 & 0\end{bmatrix} \xrightarrow[\text{无梯度拷贝}]{\text{.detach().clone()}} \text{V\_detach}$$

$$R=\begin{bmatrix}\text{Return Episode-Vector}\\ \text{Return Episode-Vector}\\ \text{Return Episode-Vector}\\ \cdots\\ \text{Return Episode-Vector}\end{bmatrix} \overset{\text{eg.}}{=} \begin{bmatrix} R0 & R1 & R2 & 0 & 0 & 0\\ R0 & R1 & 0 & 0 & 0 & 0\\ R0 & R1 & R2 & R3 & R4 & 0\\ & & \cdots & & & \\ R0 & R1 & R2 & R3 & 0 & 0\end{bmatrix}$$

$$\pi_{\theta_{\text{old}}}=\begin{bmatrix}\text{OldPolicy Episode-Vector}\\ \text{OldPolicy Episode-Vector}\\ \text{OldPolicy Episode-Vector}\\ \cdots\\ \text{OldPolicy Episode-Vector}\end{bmatrix} \overset{\text{eg.}}{=} \begin{bmatrix} q0 & q1 & 1 & 1 & 1 & 1\\ q0 & 1 & 1 & 1 & 1 & 1\\ q0 & q1 & q2 & q3 & 1 & 1\\ & & \cdots & & & \\ q0 & q1 & q2 & 1 & 1 & 1\end{bmatrix}$$

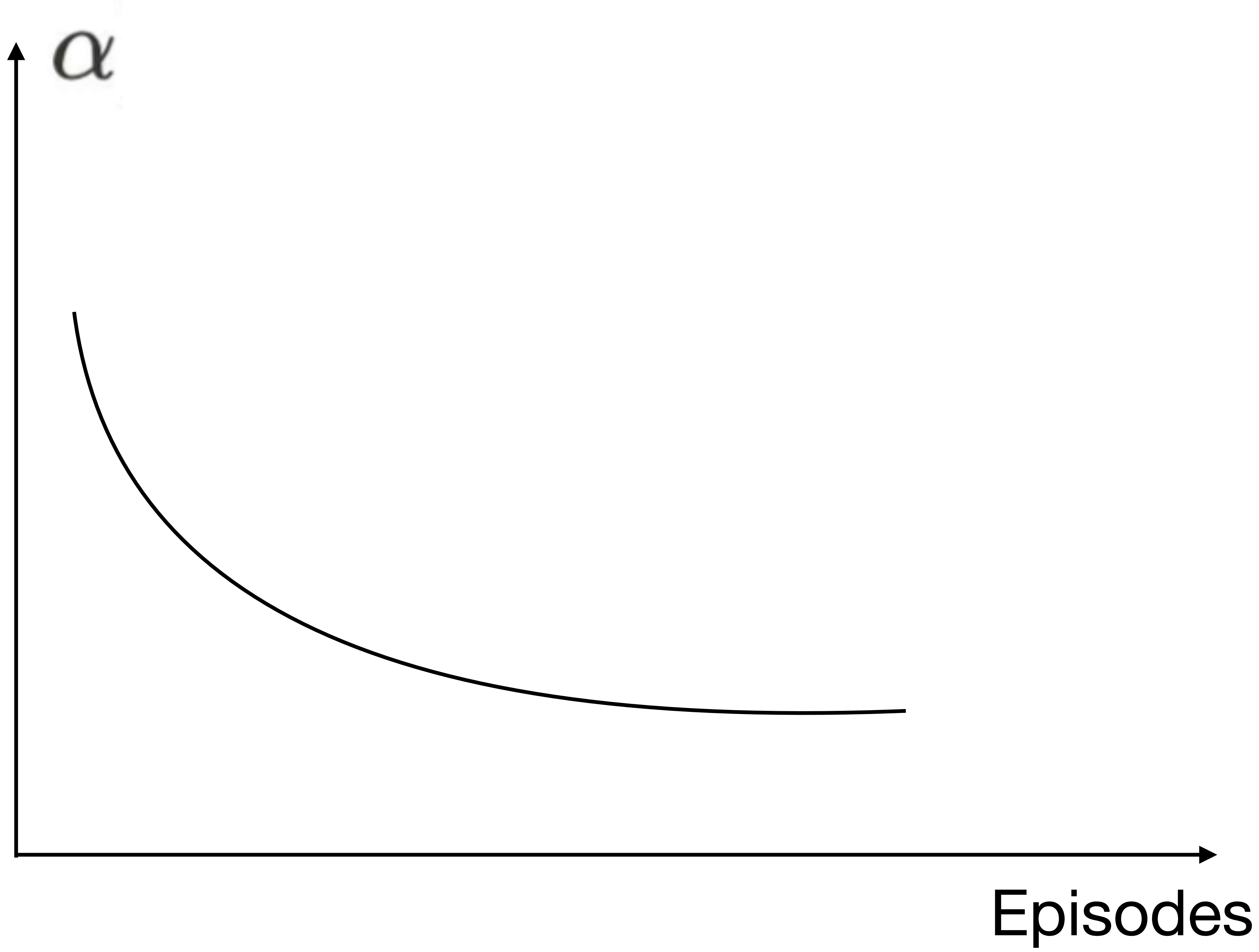
$$\pi_{\theta}=\begin{bmatrix}\text{Policy Episode-Vector}\\ \text{Policy Episode-Vector}\\ \text{Policy Episode-Vector}\\ \cdots\\ \text{Policy Episode-Vector}\end{bmatrix} \overset{\text{eg.}}{=} \begin{bmatrix} p0 & p1 & 1 & 1 & 1 & 1\\ p0 & 1 & 1 & 1 & 1 & 1\\ p0 & p1 & p2 & p3 & 1 & 1\\ & & \cdots & & & \\ p0 & p1 & p2 & 1 & 1 & 1\end{bmatrix}$$

A = R - V_detach

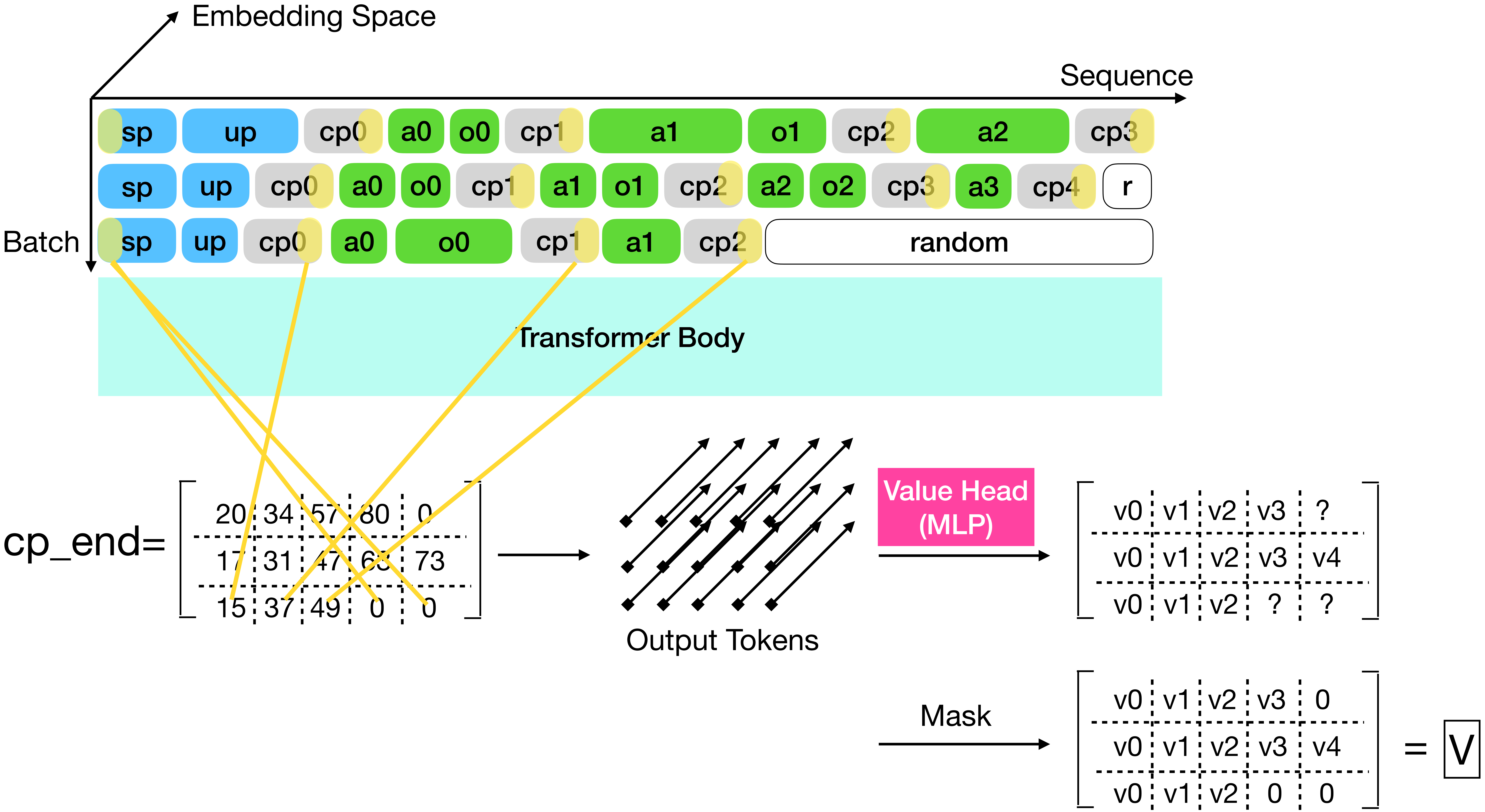
$$\text{Loss}_{\text{Actor}} = -\text{Mean}(\text{Min}(\frac{\pi_\theta}{\pi_{\theta_{\text{old}}}}A, \text{Clip}(\frac{\pi_\theta}{\pi_{\theta_{\text{old}}}}, 1-\epsilon, 1+\epsilon)A))$$

Self-AC 的训练说明 Part 5: Self-AC Train-Loss

$$\text{Loss}_{\text{Self-AC}} = \alpha \text{Loss}_{\text{Critic}} + (1-\alpha)\text{Loss}_{\text{Actor}}$$

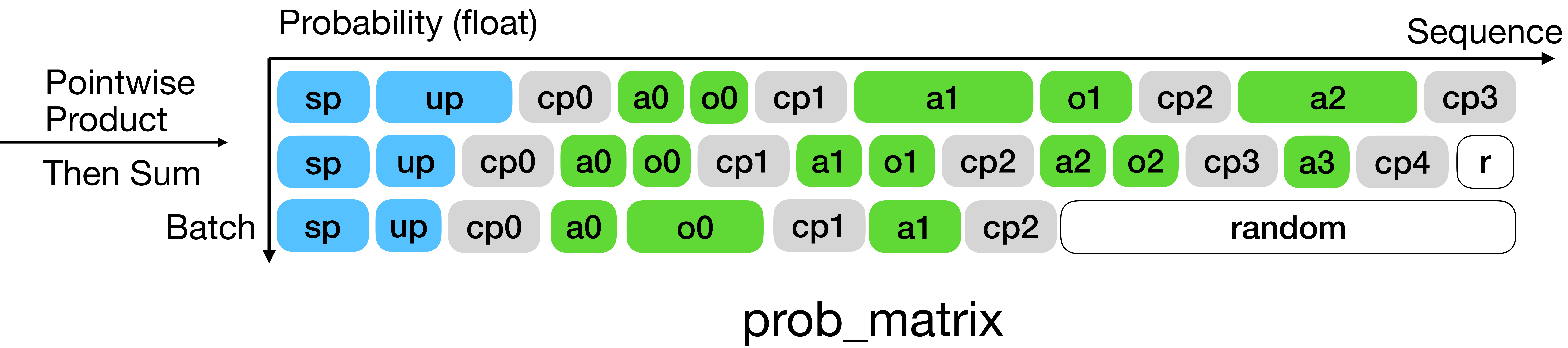


Self-AC 的训练说明 附录C: V矩阵计算

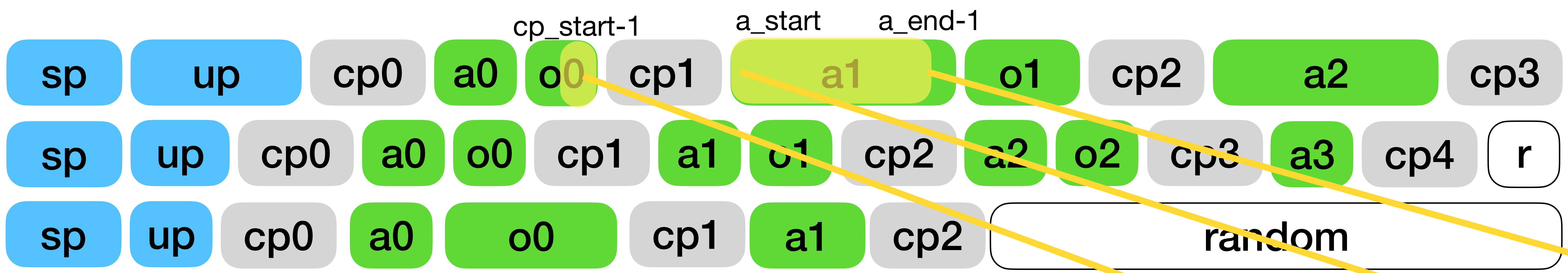


Self-AC 的训练说明 附录D1: Pi_theta矩阵计算

Self-AC 的训练说明 附录D2: Pi_theta矩阵计算



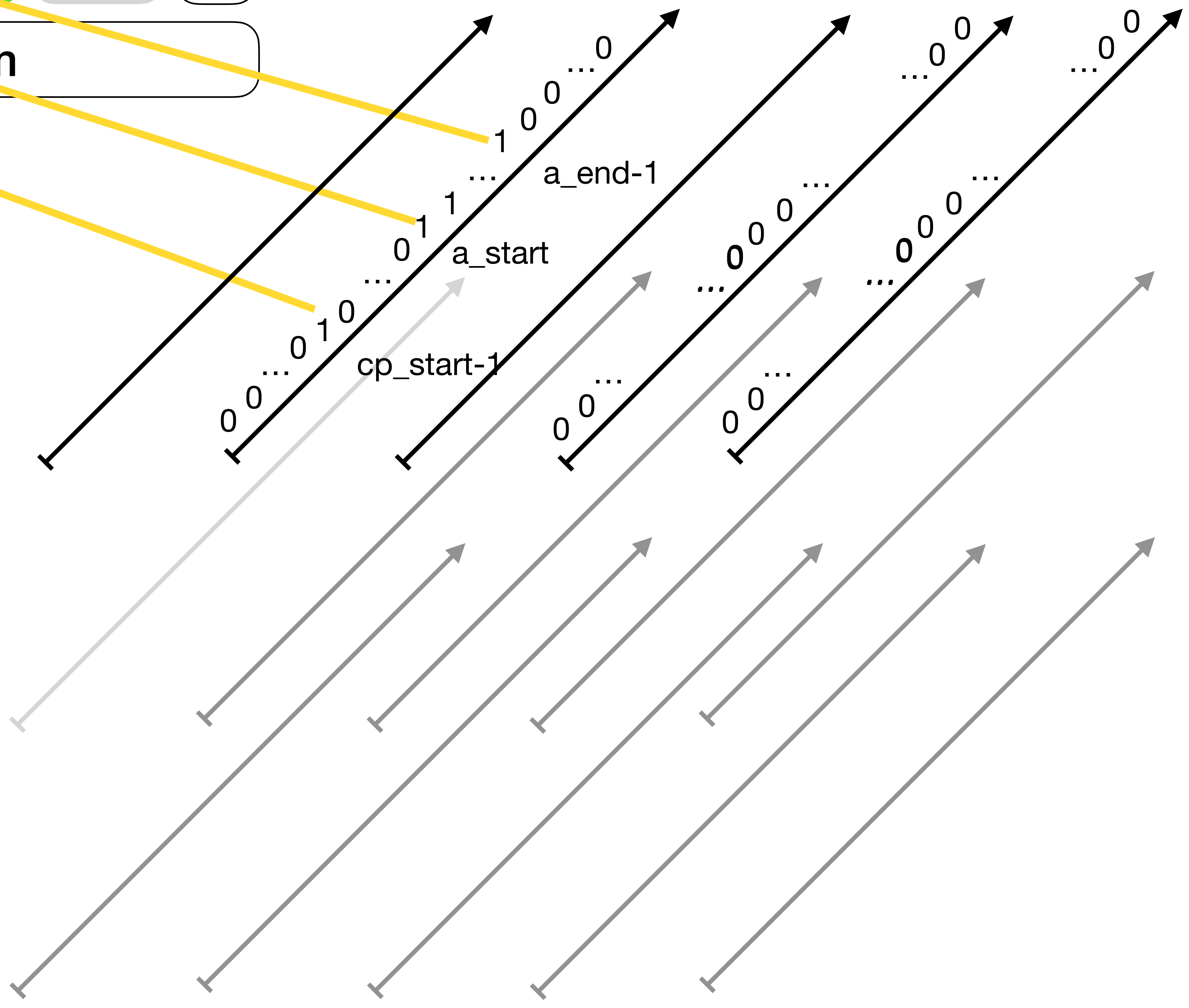
Self-AC 的训练说明 附录D3: Pi_theta矩阵计算



$$cp\_start=\begin{bmatrix} 15 & 29 & 52 & 75 & 0 \\ 12 & 26 & 42 & 58 & 68 \\ 10 & 32 & 44 & 0 & 0 \end{bmatrix}$$

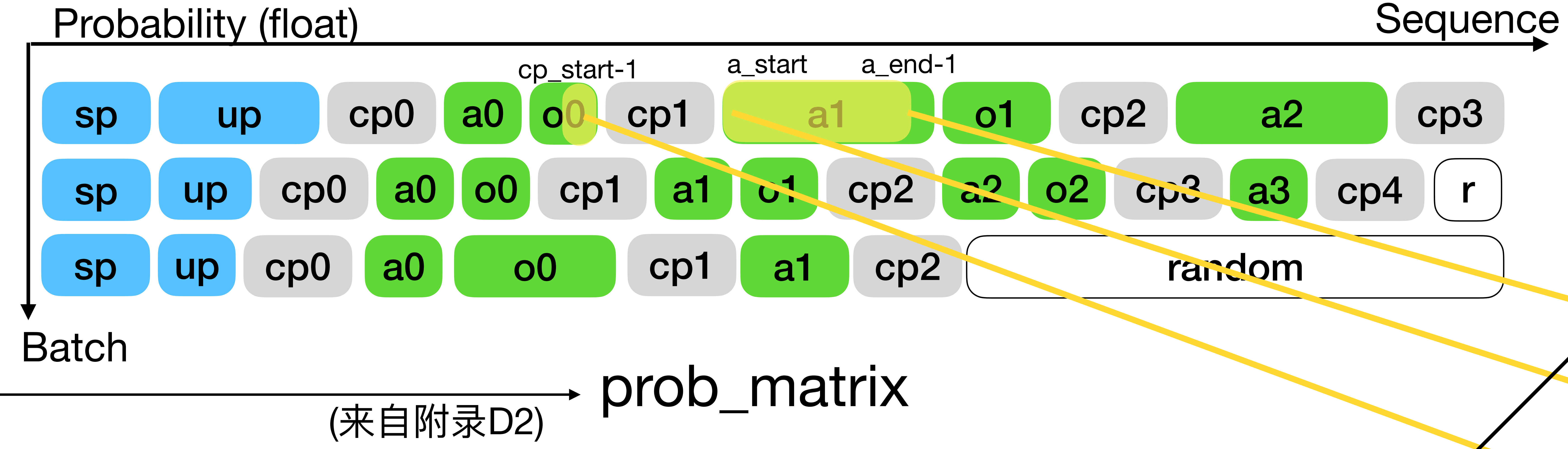
$$a\_start=\begin{bmatrix} 21 & 35 & 58 & 0 & 0 \\ 18 & 32 & 48 & 67 & 0 \\ 16 & 38 & 0 & 0 & 0 \end{bmatrix}$$

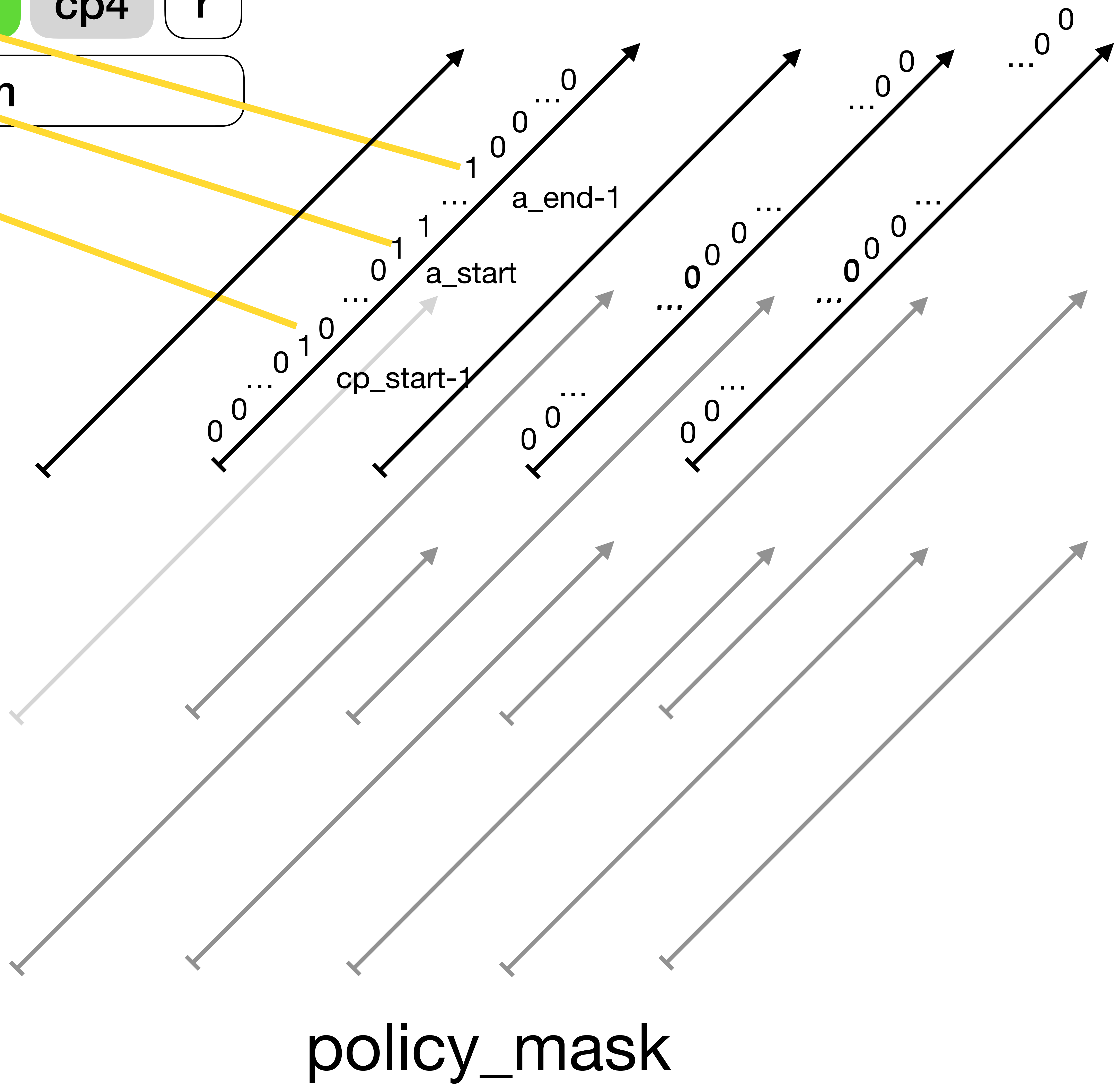
$$a\_end=\begin{bmatrix} 25 & 51 & 76 & 0 & 0 \\ 22 & 36 & 52 & 71 & 0 \\ 20 & 44 & 0 & 0 & 0 \end{bmatrix}$$



policy_mask

Self-AC 的训练说明 附录D4: Pi_theta矩阵计算





π_θ = torch.prod(
policy_mask * prob_matrix[None, :, :] +
(1 - policy_mask),
dim=-1)

例子

ReAct-SelfAC Agent

建模所有 Agent-Env 类问题

Agent-Env 类问题包括：DeepResearch(反复与浏览器交互) / CodingAgent(反复与代码编辑器、终端、浏览器交互) / 游戏Agent(反复与文字化的游戏交互) / PC Agent(反复与桌面交互)

ToT-SelfAC Agent

建模所有 Agent-树搜索 问题

Agent-树搜索 问题包括：数学解答寻找 / 智力游戏 / …