arXiv:2603.05694v1 [cs.LG] 5 Mar 2026

利用自动机学习热启动状态空间模型

William Fishell[1],Sam Nicholas Kouteili[2], 和 Mark Santolucito[1†]

[1] 哥伦比亚大学（† 巴纳德学院，哥伦比亚大学）[2] 耶鲁大学

摘要。我们证明，摩尔机可以被精确地实现为状态空间模型（状态空间模型），从而确立符号自动机与这些连续机器学习架构之间的形式对应关系。这些 Moore-状态空间模型既保留了原始摩尔机完整的符号结构和输入输出行为，又在欧几里得空间中运行。基于这种对应关系，我们比较了状态空间模型与被动和主动自动机学习的训练效果。在从 SYNTCOMP 基准测试中恢复自动机时，我们发现，状态空间模型所需的数据量比符号方法高出几个数量级，而且无法学到状态结构。这表明，符号结构为学习此类系统提供了强大的归纳偏置。我们利用这一发现，结合自动机学习和状态空间模型各自的优势，以高效学习复杂系统。我们在 SYNTCOMP 的一组仲裁器上学习自适应仲裁策略，并表明，使用符号学习得到的近似模型初始化状态空间模型，可以实现更快、更好的学习。与随机初始化的模型相比，我们观察到收敛速度提高了 2 至 5 倍，且测试数据上的模型准确率整体更高。我们的工作将自动机学习从纯离散空间拓展出来，使人们能够在连续域中有原则地利用符号结构，从而高效地学习复杂场景中的系统。

1 引言

诸如主动和被动自动机学习之类的符号学习方法 [6,33]对于允许有限可表示行为模型的系统非常有效。这些方法已被广泛用于推断网络和通信协议的模型 [3]，，它们通过将目标系统视为黑盒来推断行为模型。自动机学习恢复符号结构：对状态、转移和输出的显式离散表示，这些表示能够在有限配置集合上完整刻画系统行为。然而，这些方法面临关键局限：被动学习难以扩展到复杂问题，而主动学习则受限于成员查询和等价查询的成本 [4]。自动机的离散性质加剧了这些挑战，因为行为模型之间不存在邻近性概念；求解一个更简单的问题并不能为学习相关但更复杂的目标提供初始化。此外，跟踪累积量相关事物的系统，

例如版本控制历史或 API 请求计数，从根本上超出了经典自动机学习的能力范围。这些方法恢复有限状态表示，但行为依赖于整个无界历史的系统需要无限记忆。

我们通过形式化摩尔机与状态空间模型（SSM）之间的关系来解决这些问题 [17]。SSM 是连续状态的递归模型，通过学习到的线性动力学处理序列。诸如 Mamba [16] 这样的 SSM 日益受到关注，因为它们提供了一种计算效率高的替代方案，可用于取代能够计算线性注意力的 Transformer 架构。除了效率之外，我们还发现 SSM 能够精确实现摩尔机。将摩尔机表示为 SSM 有三项优势：我们继承了符号结构作为归纳偏置；获得了一个可利用邻近性的连续空间，从而能够通过基于梯度的优化适应更复杂的情形；并且可以利用递归，更轻松地对无限状态系统建模。这种对应关系提供了一条端到端流程：自动机学习提取结构并以此初始化连续模型，从而拓展符号方法的适用范围，并使深度学习场景中的学习更加高效。

我们在一组来自 SYNTCOMP 的仲裁器上评估了符号热启动方法。这些系统的底层控制策略由有限状态仲裁器描述。随后，我们扩展这一基础策略，使仲裁器必须跟踪每个通道相对于平均授权次数的偏差。计算这一偏差需要跟踪整个历史中的授权情况，因而理论上需要无限记忆。这类系统无法由有限状态机精确表示，因此非常适合作为以符号结构初始化的神经网络方法的应用对象。

为阐明这一混合方法的动机，我们首先确立符号结构在纯有限状态场景中的价值。我们在一组来自 SYNTCOMP 的正则语言上，评估符号方法和神经网络方法的样本效率。我们将主动学习（L*算法）和被动学习（RPNI）与通过梯度下降训练、随机初始化的状态空间模型进行比较。这项基线实验衡量了每种方法需要多少样本才能恢复目标自动机的行为。自动机学习的样本效率显著高于从头训练的神经网络方法。这促使我们将符号结构用作学习系统的初始化，使其能够超越有限状态表示[3]。总而言之，本研究的主要贡献如下：

1. 据我们所知，我们首次利用通过经典学习算法恢复的自动机来初始化状态空间模型，从而能够以更高的样本效率学习复杂场景。2. 我们形式化了一项证明，表明摩尔机可作为状态空间模型的精确实现，同时保留结构与行为。3. 我们在 SYNTCOMP 基准测试上开展了一项实证研究，表明符号方法的样本效率比梯度训练的状态空间模型高出数个数量级。

[3] 所有代码均可在 https://github.com/wfishell/Automata SSM Learning/tree/main 获取

日期	小时	请求	操作	C1/C2/C3/C4	总计	
星期一	00:00	C1	Grant	2/0/0/0		2
星期一	00:00	C1	Grant	2/0/0/0		2
星期一	00:00	C1	**Rej.**	2/0/0/0		2
星期一	01:00	C1	Grant	2/0/0/0		2
星期一	01:00	C1	Grant	2/0/0/0		2
星期一	01:00	C1	**Rej.**	2/0/0/0		2
星期一	02:00	C1	Grant	1/0/0/0		1
星期一	03:00	C1	Grant	2/0/0/0		2
星期一	03:00	C1	Grant	2/0/0/0		2
周一	04:00	C1	Grant	2/0/0/0		2
周一	04:00	C1	Grant	2/0/0/0		2
周一	04:00	C1	**Rej.**	2/0/0/0		2
周一	05:00	C1	Grant	2/0/0/0		2
周一	05:00	C1	Grant	2/0/0/0		2
周一	08:00	C2	Grant	0/2/0/1		3
星期一	08:00	C2	Grant	0/2/0/1		3
星期一	08:00	C4	Grant	0/2/0/1		3
周一	09:00	C2	Grant	0/1/0/2		3
周一	09:00	C4	Grant	0/1/0/2		3
周一	09:00	C4	Grant	0/1/0/2		3
周一	10:00	C3	Grant	0/0/2/0		2
周一	10:00	C3	Grant	0/0/2/0		2
星期一	11:00	C4	Grant	0/0/0/1		1

（a）循环仲裁器，硬性拒绝 25%

日期	小时	请求	操作	C1/C2/C3/C4	总计	
星期一	00:00	C1	Grant	2/0/0/0		2
星期一	00:00	C1	Grant	2/0/0/0		2
星期一	00:00	C1	**Rej.**	2/0/0/0		2
星期一	01:00	C1	Grant	2/0/0/0		2
星期一	01:00	C1	Grant	2/0/0/0		2
星期一	01:00	C1	Grant	3/0/0/0		3
星期一	02:00	C1	Grant	1/0/0/0		1
星期一	03:00	C1	Grant	2/0/0/0		2
星期一	03:00	C1	Grant	2/0/0/0		2
周一	04:00	C1	Grant	2/0/0/0		2
周一	04:00	C1	Grant	2/0/0/0		2
周一	04:00	C1	Grant	3/0/0/0		3
周一	05:00	C1	Grant	2/0/0/0		2
周一	05:00	C1	Grant	2/0/0/0		2
星期一	08:00	C2	Grant	0/2/0/1		3
星期一	08:00	C2	Grant	0/2/0/1		3
星期一	08:00	C4	Grant	0/2/0/1		3
周一	09:00	C2	Grant	0/1/0/2		3
周一	09:00	C4	Grant	0/1/0/2		3
周一	09:00	C4	Grant	0/1/0/2		3
周一	10:00	C3	Grant	0/0/2/0		2
周一	10:00	C3	Grant	0/0/2/0		2
星期一	11:00	C4	Grant	0/0/0/1		1

（b）新期望策略的合成数据

2 动机示例

云资源分配是一项重要任务，用于管理客户对亚马逊网络服务（AWS）等提供商所提供云资源的访问 [5]。对分配协议进行建模与验证一直是形式化方法研究的重点 [8,12,15]。此外，机器学习技术越来越多地被用于对更复杂的资源分配方案进行建模 [35,29]。再者，热启动——即先用从较简单场景中学到的参数初始化模型，再在更复杂实例上训练——也被证明对云资源分配有效。先前工作在用于云资源分配的神经网络模型中采用热启动，并依赖更简单的神经架构与无监督方法 [25]。我们的工作通过利用符号结构对神经网络模型进行热启动，从而弥合形式化方法与神经方法之间的鸿沟，以高效学习云资源分配中的自适应仲裁策略。

示例 *1*（*GPU* 分配系统）。A 公司是一家云 GPU 提供商，拥有四位客户和一个 8-GPU 集群。A 公司采用基于循环仲裁器的策略来授予客户访问其网络的权限，并附加安全约束：任何用户请求的资源不得超过网络容量的 25%。一旦分配了 GPU，其使用权将在整个小时内保持有效。之后，该用户将被踢出网络。

A 公司通过分析表 1a 得出结论：其静态公平性策略正在导致资金损失。从午夜到凌晨 5 点，C1 是唯一活跃的客户，但由于僵化的 25% 上限，即使有大量未使用的容量，请求仍会被拒绝。他们希望建立模型，以便根据底层仲裁结构动态调整安全约束。

混合控制器通过将任务划分到不同的架构组件上来解决此类问题。神经模块可以处理复杂的模式识别，而符号模块则强制执行公平性或安全性等硬约束。

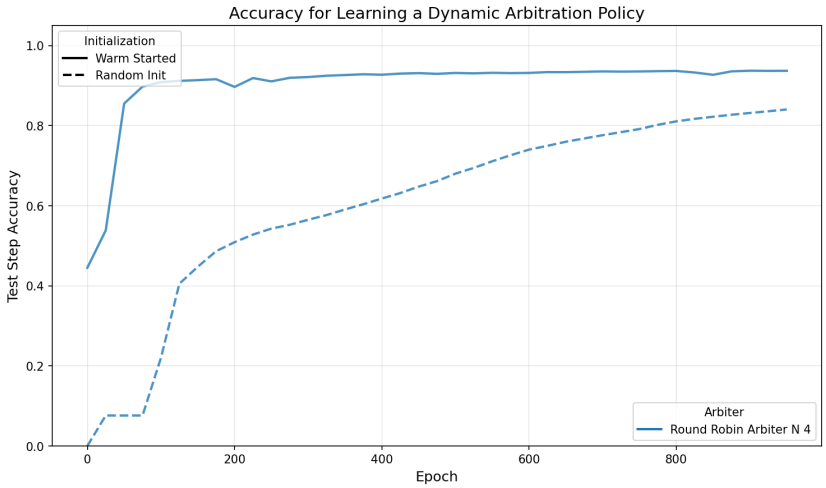

图 1：在具有 4 个通道的动态循环仲裁策略上，利用符号结构对状态空间模型进行热启动，与使用传统随机初始化进行学习的对比。

这种划分利用了学习方法与形式化方法各自互补的优势。例如，近期研究探索了将状态空间模型与传统控制器相结合的混合架构，利用队列反馈机制以及能够捕捉整个输入历史依赖关系的状态空间模型，为网状 Web 系统进行资源配置 [21]。这些方法产生了一个系统，其中各组件彼此独立地运行，并对问题的不同组件进行建模。由于不存在统一的策略，这可能造成灾难性边缘情况 [13]。

我们提出了一种替代方案。我们不再耦合相互独立的模块，而是训练单个状态空间模型，使其充当完整的分配策略。A 公司可以追溯分析其历史数据，并生成反映其期望行为的合成轨迹：只要容量充足就批准请求，不受静态上限限制；同时，在多个客户端竞争时仍确保公平性。

随后可使用表 1b 中的数据训练一个状态空间模型，使其同时捕捉仲裁逻辑以及更为复杂的依赖历史的动态特性。然而，随机初始化的状态空间模型难以高效学习此类系统；符号热启动可显著改善收敛与准确率。我们在图 1 中说明了这一点：我们学习一种动态仲裁策略，其行为类似于具有 4 个通道的循环仲裁器，但必须持续将授权保持在不断演变的安全界限之内。热启动的影响表现为更快收敛到稳定的准确率，并且即使经过 1000 个训练轮次后，总体准确率也更高。

3 预备知识

3.1 状态空间模型

经典的状态空间模型表示为

$$\begin{aligned} x(t+1) &= Ax(t) + B\mu(t) \\ y(t) &= Cx(t) + D\mu(t). \end{aligned} \tag{1}$$

其组件为

– $x(t+1)$ 是未来状态– A 是一个描述状态如何随时间演变的矩阵– $x(t)$ 是当前状态– B 是一个描述当前输入如何影响下一状态的矩阵– $\mu(t)$ 是当前输入– $y(t)$ 是系统当前的输出– C 是一个描述当前状态如何影响输出的矩阵– D 描述当前输入如何影响输出

通常，这些系统可以更简洁地表示为

$$\begin{aligned} x(t+1) &= Ax(t) + B\mu(t) \\ y(t) &= Cx(t) \end{aligned} \tag{2}$$

其中 $D = 0$ [9]。为便于建模，我们采用离散化时间概念。这些状态空间模型关于输入和状态都是线性的。尽管近期的状态空间模型变体（如 Mamba 和 S4 [16] ）引入了额外的复杂性，我们仍聚焦于公式 2 中的典范形式，因为它更清晰地揭示了该架构的基本属性。

3.2 摩尔机

摩尔机是一种描述离散符号状态空间的系统，其未来状态取决于当前状态和输入，而输出取决于当前状态。具体而言，摩尔机定义为一个系统

$A = (S,\ S_0,\ \Sigma,\ \Lambda,\ T,\ G)$，其中

– S 是有限状态的集合– S_0 是初始状态– Σ 是输入字母表– Λ 是输出字母表– T 是将输入状态对映射到未来状态的转移函数[4]– G 是将输出状态对映射为输出的输出映射

具体而言，摩尔机可以表示为两个不同的函数：

$$\begin{aligned} S_k &= F(S_i, \Sigma_j), \quad \text{where } S_0 \text{ is the initial state,} \\ \Lambda_j &= G(S_i). \end{aligned} \tag{3}$$

[4] 这也通常表示为 δ .

4 Moore-状态空间模型

尽管公式 (2) 和 (3) 在结构上相似，但公式 (2) 中状态空间模型生成新状态时所具有的加法性质意味着状态与输入彼此独立，而摩尔机中并不存在这种性质。

$$Ax(t) + B\mu(t)$$

式 (3) 中的摩尔机将当前输入和状态共同作为函数的自变量，返回下一个状态。

$$S_k = F(S_i, \Sigma_j)$$

因此，要在欧几里得空间中使用状态空间模型的动力学来表示我们的摩尔机，映射必须允许输入和状态独立运作。

引理 1（Moore-状态空间模型）。每个摩尔机 $\mathcal{A} = (S, S_0, \Sigma, \Lambda, T, G)$，其转移映射和输出映射由式 (3) 定义，均可等价地表示为如下形式的离散时间状态空间模型

$$\begin{aligned} x(t+1) &= Ax(t) + B\mu(t), \\ y(t) &= Cx(t), \end{aligned}$$

其中，状态向量 $x(t)$ 对符号状态 S, $\mu(t)$ 对输入进行编码，而矩阵 A、B 和 C 经过结构化设计，以保留摩尔机原有的输入—输出行为和状态转移。

证明（引理 *1*）。给定摩尔机 $\mathcal{A} = (S, S_0, \Sigma, \Lambda, T, G)$，我们首先关注如何以状态空间模型的形式表示该系统的状态更新

$$x(t+1) = Ax(t) + B\mu(t)$$

使用状态空间模型表示摩尔状态更新动力学
将每个 $S_i \in S$ 视为正交归一基中的一个基向量，我们可以将符号状态从离散空间映射到欧几里得空间 E 中的 $\mathbb{R}^N$，其中 $N = |S|$。因此，状态空间模型中记为 $\mathbf{e}_i$ 的每个状态向量都属于 E，并表示为

$$\mathbf{e}_i^\top = \begin{pmatrix} 0 & 0 & \cdots & 1 & \cdots & 0 \end{pmatrix}$$

矩阵 A 展示了 $x(t) = e_i$ 在没有任何输入的情况下如何随时间演化；然而，在摩尔机中，没有输入时状态不会发生演化。因此，我们可以将 A 设为单位矩阵，使得 $Ax(t) = Ix(t) = x(t)$。令 $A = I$ 可得到如下方程

$$x(t+1) - x(t) = e_j - e_i = B\mu(t)$$

该表述将状态变化表示为输入的函数。然而，这种表示并未消除状态与

输入之间的共依赖关系，而这种共依赖刻画了摩尔机。尽管差值 $x(t+1)-x(t)$ 关于 $\mu(t)$ 是线性的，其底层转移动态仍共同依赖于当前符号状态和当前输入。因此，仅凭该表述不足以解耦摩尔机的状态–输入依赖。

为了在状态空间模型框架内保留这种联合依赖，我们将输入重新解释为一个同时编码当前状态与当前输入的单一复合符号。具体而言，通过将输入空间定义为克罗内克积 $S \otimes \Sigma$，我们构造增广输入向量 $\mu(t)$，使得每个输入符号唯一表示一个状态-输入对。在此编码下，状态转移相对于 $\mu(t)$ 变为线性，同时仍保留原始摩尔机的动态特性。

$$\mu(t)^\top = \left[\mu_{(s_1,\sigma_1)}\ \mu_{(s_1,\sigma_2)}\ \mu_{(s_1,\sigma_3)}\ \mu_{(s_2,\sigma_1)}\ \mu_{(s_2,\sigma_2)}\ \mu_{(s_2,\sigma_3)}\ \cdots\ \mu_{(s_n,\sigma_3)}\right]$$

通过选取上述克罗内克积形式的 $\mu(t)$，我们定义矩阵 $B \in \mathbb{R}^{|S|\times(|S|\cdot|\Sigma|)}$，使得每一列对应一个唯一的状态-输入对 (s_i,σ_j)。具体而言，由 (i,j) 索引的列由下式给出

$$B_{:,(i,j)} = e_{T(i,j)} - e_i$$

其中 $x_i \in \mathbb{R}^{|S|}$ 表示与当前状态 s_i 对应的标准基向量，$e_{T(i,j)}$ 表示由转移函数 $T(i,j)$ 从 A 确定的后继状态所对应的标准基向量。在此构造下，状态更新

$$\begin{aligned} x(t+1) - x(t) &= B\mu(t), \\ x(t+1) &= x(t) + B\mu(t), \\ x(t+1) &= Ax(t) + B\mu(t) \end{aligned}$$

精确复现摩尔机的转移动态。

使用状态空间模型表示摩尔输出更新动态

将摩尔机 $\mathcal{A}$ 的输出动态编码到状态空间模型的输出方程 $y(t) = Cx(t)$中十分简单，因为摩尔机的输出仅取决于当前状态。将每个状态 $s_k \in S$ 表示为一个独热向量 $x_t = e_k \in \mathbb{R}^{|S|}$后，我们可以构造一个输出矩阵 $C \in \{0,1\}^{|\Lambda|*|S|}$，其各列编码了摩尔输出函数。具体而言，$C$ 的第 k 列是与输出符号 $\lambda = G(s_k)$对应的独热向量。在这种构造下，线性读出

$$y(t) = Cx(t)$$

能够精确复现摩尔机的输出，其中 $y(t)$ 表示输出字母表元素 $\lambda \in \Lambda$ 的独热编码表示。因此，我们已将 A 完美编码进状态空间模型的动态中。 □

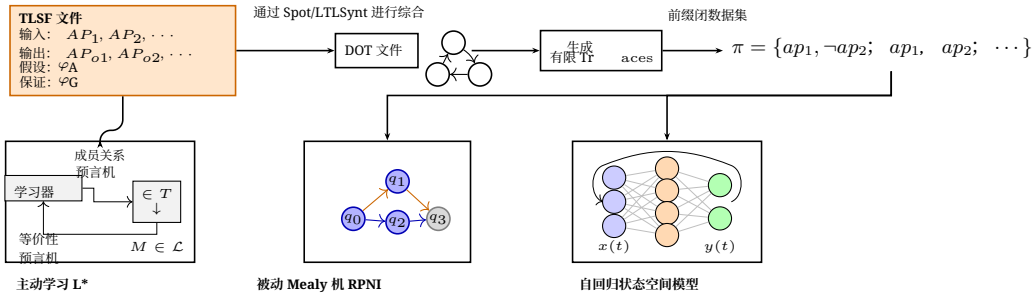

图 2：从 SYNTCOMP 学习正则语言的高级训练流水线可视化

5 从数据中学习正则语言

引理 1 确立了摩尔机与一类状态空间模型之间的精确等价性。然而，我们表明，在正则语言任务上训练状态空间模型时，梯度下降无法恢复符号结构，即使该结构能够完全刻画目标函数也是如此。相比之下，自动机学习方法直接利用这种结构。这一差异凸显了符号先验在加速连续架构学习方面的潜力。我们评估了随机初始化的状态空间模型，并将其与主动和被动自动机学习方法进行比较，考察它们学习由 SYNTCOMP [19] 合成的自动机结构的表现。尽管 AALpy 等领先的自动机学习工具使用随机自动机结构进行基准测试， [23],更具结构性的基准测试集能让我们更清楚地了解学习动态。

问题 *1*。（SYNTCOMP 仿真任务）具体而言，我们的任务是恢复一个系统 $\mathcal{A}^*$，该系统生成无限轨迹 $\omega \in \Sigma \cup \Lambda$，其中 Σ 和 Λ 是输入和输出字母表，且其行为被原始系统 $\mathcal{A}$接受。由于我们正在测试无法恢复自动机结构的状态空间模型，因此不使用 ω-测试，而是选择更简单的任务，即仿真语言 L。我们的评估指标是由 $\mathcal{A}^*$ 生成且被 $\mathcal{A}$接受的轨迹 ω_i 所占的百分比。

5.1 方法

图 2 对比了从自动机 $\mathcal{A}$ 学习正则语言 L 的三种方法。首先，我们使用 ltlsynt 工具 [27]通过 Spot2.14 [11] 从 TLSF 文件合成一个系统。该系统是我们主动学习设置中的预言机系统。主动学习通过查询在内部生成数据，而基于梯度的学习和被动学习则需要训练数据集。因此，我们必须从合成的系统中生成符合其假设的数据。

数据生成 从自动机系统生成数据是一项尚未得到充分探索的任务。从这些系统生成数据时存在两个问题：（1）生成的数据必须是有限的，但原始系统接受无限行为；（2）大多数数据生成算法不遵循底层系统的复杂假设。我们

通过简化生成训练数据的要求来解决问题 1。为此，我们利用 Spot 语义。生成的数据需要在生成的有限迹长度为 T 时遵循底层语言的规则。随后我们附加循环{1}。这表示之后的所有行为都为真，因此轨迹中的任何错误行为都只能出现在前 T 个时间步内。我们使用 Spot 语义生成数据来验证轨迹，然后使用 autfilt 检查该系统的 HOA [7] 版本，以验证这条有限迹。（样本迹见附录 A.1。）

HOA 文件格式表示一个与其摩尔结构等价的系统，该结构在输入和输出对的并集上具有确定性。虽然 HOAX [10] 等工具能够使用 HOA 文件格式生成符合这些系统行为的有限轨迹和无限轨迹，但这些工具速度较慢，并且在尝试对复杂假设建模时会失败。

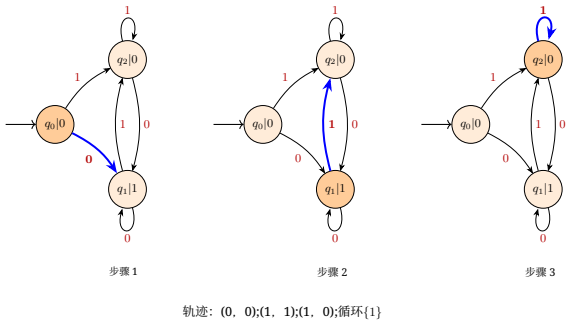

图 3：通过从状态中随机选择转移来生成符合假设的轨迹的方法

如图 3 所示，我们使用 GraphViz Dot 文件表示法 [14] 将系统编码为关于输入确定的系统。我们通过在该图上进行随机游走并记录生成的轨迹来生成数据，并在每条轨迹后附加循环{1} 。借助真实的 Moore 机结构，我们确保所有数据都满足规约假设，同时随机游走也使大规模数据生成能够高效进行。

主动自动机学习 我们使用 AALpy 进行主动学习，以恢复与目标摩尔机等价的 Mealy 机；AALpy 对 Mealy 机的支持更好，因此我们依赖摩尔机与 Mealy 机众所周知的等价性 [20]。我们采用标准的主动学习设置，使用 L^* 算法。尽管自动机学习领域已有许多进展（参见第 7 节），但 L^* 足以说明此设置下主动学习的样本效率。我们的被学习系统（SUL）是根据 TLSF 规约最初合成的自动机。对于成员查询，

#原始迹 g 0 &! c 0&r 0 &! g 1 &! c 1&r 1 ; ! g 0&c 0 &! r 0&g 1 &! c 1 &! r 1 ; 循环 {1}

#前缀–闭合表示（!c 0 & r 0 & !c 1 & r 1），（g 0 & !g 1）（!c 0 & r 0 & !c 1 & r 1；c 0 & ! r 0 & !c 1 & !r 1），（!g 0 & g 1）

图 4：样本迹的前缀封闭表示

L^* 向 SUL 提交一个输入序列，SUL 对自动机进行仿真，并返回相应的输出符号序列——每个输出都是输出命题上的一个布尔赋值。这提供了教师的响应，而无需访问自动机的内部结构。对于等价查询，我们使用随机游走算法，对输入序列进行采样，以寻找假设与 SUL 之间的反例。这一设置较为简单，且将 SUL 选为原始系统并不能反映现实应用。然而，由于所有数据都来自无噪声的干净来源，它为与其他学习方法进行比较提供了有用条件。

我们在每个 SYNTCOMP 基准测试上进行三次主动学习试验，并报告查询数量的平均值作为样本复杂度。对于每个恢复出的自动机，我们使用数据生成流水线生成 1000 条长度为 20 的轨迹，并计算原始系统接受的轨迹百分比。报告的准确率为三次试验的平均值。试验在 AWS Lambda 上运行；超时的试验标记为失败，并从最终结果中排除。（自动机学习的输出结构请参见附录 A.1。）

被动自动机学习 被动学习系统使用 RPNI 来学习与底层摩尔机等价的 Mealy机。我们通过在 DOT 文件表示上进行随机游走，利用数据生成流水线生成轨迹，并将其转换为前缀封闭表示，以便从数据中学习自动机。图 4 展示了从 DOT 文件生成并转换为适用于被动学习设置的形式的数据。

与主动学习设置不同，被动学习需要预先提供数据。我们在 5000 到 30000 条长度为 20 的轨迹范围内评估被动学习设置。准确率的测量方式与主动学习设置相同：我们从学习到的系统生成 1000 条轨迹，并计算原始系统接受的轨迹百分比。我们将达到最高准确率所需的样本数量作为样本复杂度进行报告。达到 100% 准确率的系统不再使用更大的样本量进行测试。

SSM 学习 我们使用公式 2 中的状态空间模型（SSM）学习这些正则语言。模型采用自回归方式进行训练：给定时间步 0 到 T 的输入序列，预测时间步 T 的输出，并使用 PyTorch 库 [18]。图 5 展示了用于学习 SYNTCOMP 基准测试的 SSM 架构。

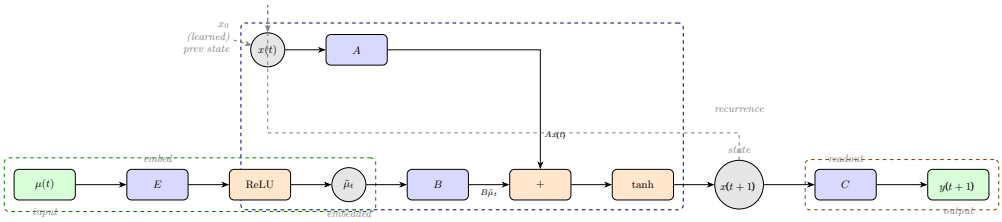

图 5：用于学习 SYNTCOMP 正则语言的状态空间模型架构

训练时使用二元交叉熵损失，预测维度为 $\Lambda \times 1$ 的布尔输出张量，其中 Λ 表示原始系统中的输出 AP 数量。训练过程中，模型会生成一个位于 $\mathbb{R}^{\Lambda \times 1}$ 中的实值输出向量，并将其输入 logit 函数，以获得位于 $\{0,1\}^{\Lambda \times 1}$ 中的布尔预测。随后，计算该预测与真实布尔输出之间的二元交叉熵损失。我们采用标准学习率 .001 和 Adam优化器 [2]。

每个时间步的输入都表示为 $\{0,1\}^{\Sigma \times 1}$ 中的布尔向量，其中 Σ 表示系统中的输入原子命题数量。通过一个经学习得到的线性嵌入层，再经过 ReLU 激活，将这些输入嵌入到 32 维连续空间中。这种嵌入过程是标准做法，既能让状态空间模型基于连续表示运行，又能保留输入字母表的离散结构。为了计算时间 T 的输出，模型会接收完整输入序列，并将其作为 $\{0,1\}^{\Sigma \times T}$ 中的矩阵。状态空间模型沿时间展开，使用公式 2 中的状态更新方程迭代计算每一步的潜在状态。随后，最终的潜在状态通过输出函数映射，生成输出向量。

数据的生成方式与被动学习相同。我们生成 10000 条长度为 20 的轨迹，其中 10% 留作测试。尽管模型以自回归方式训练，但评估采用更严格的轨迹接受标准，而非逐步预测准确率。具体而言，给定从时间 0 到 20 的输入序列，我们要求模型正确预测轨迹中的~~每~~个输出。若预测的输出序列与原始机器生成的真实输出序列完全一致，则认为该测试轨迹被接受。我们报告按此标准接受的测试轨迹百分比，以衡量状态空间模型对目标系统完整输入—输出行为的模拟效果。与被动和主动学习方法不同，梯度下降可以迭代优化自身。我们在样本复杂度计算中对此予以考虑，报告准确率最高时的训练轮次 $\times$9000；这表示训练数据被用于学习正则语言的次数。为保持一致，所有任务的训练轮次上限均设为 1000。

最后，我们强调，在本实验中，矩阵 A、B 和 C 均按常见做法随机初始化。若按照引理 1 初始化这些矩阵，将得到一个已与目标摩尔机完全对齐的系统，从而使学习任务变得平凡。其目的

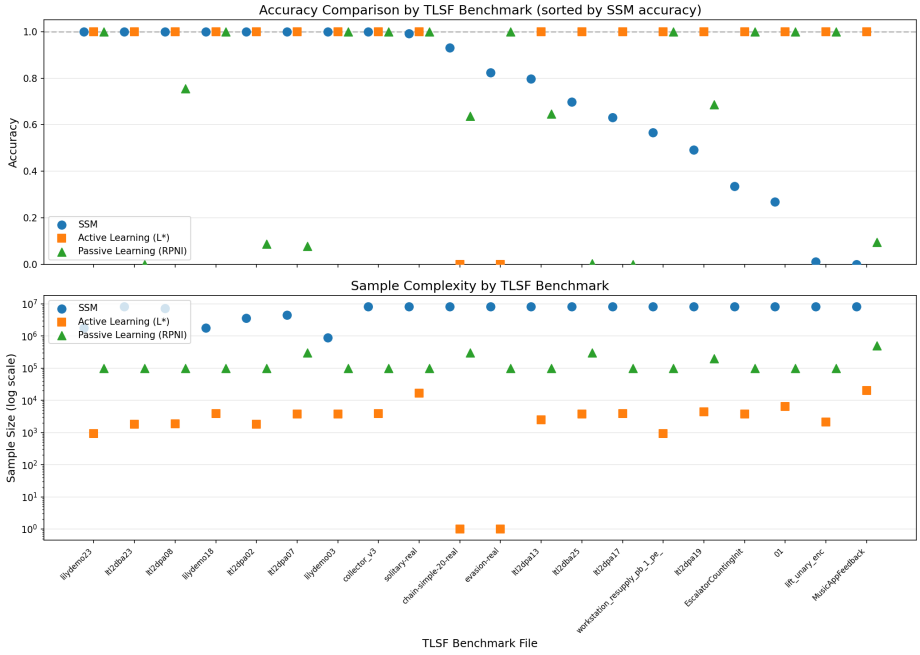

图 6：学习随机抽取的 20 个 SYNTCOMP 示例时的准确率和样本复杂度（对数尺度）

则是研究通过显式利用底层问题中的符号结构所获得的样本效率提升，并评估此类结构能否通过基于梯度的优化被自然发现。

5.2 符号学习与基于梯度的学习的样本效率

图 6 展示了使用主动学习、被动学习和基于梯度的方法学习一组 20 个 SYNTCOMP 基准测试时的准确率和样本复杂度。附录 A.2 的表格列出了所有评估过的基准测试。第一张图报告了各学习方法在相应基准测试上的轨迹接受任务准确率，第二张图则以对数尺度报告样本复杂度。表 2 显示了各学习方法所学习的系统中被完美仿真的系统所占总百分比。该表表明，总体而言，在学习这些正则语言问题时，状态空间模型的表现远不如被动学习或主动学习方法。

5.3 基于梯度的模型中的符号结构

尽管摩尔机可以精确实现为状态空间模型，但在具有符号结构的任务上对状态空间模型进行基于梯度的训练，并不能恢复出接近符号结构的表

学习方法	完美仿真
基于梯度的状态空间模型	33.3%
L^*	77.3%
RPNI	56.0%

表 2：各学习方法以 100% 准确率解决的 SYNTCOMP 基准测试百分比。

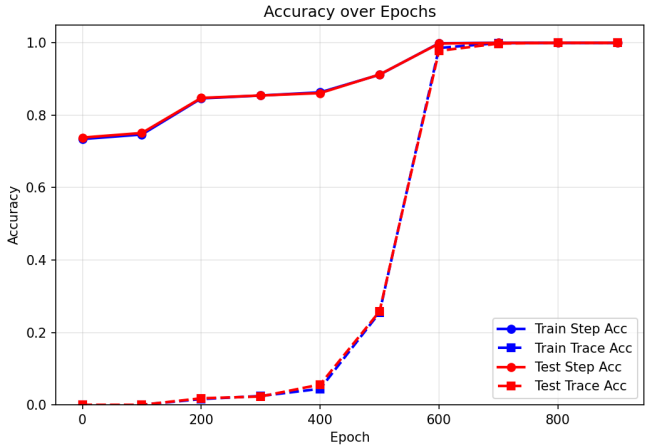

图 7：collector/parametric/collector v2 在预测下一步和轨迹接受情况上的训练与测试准确率，N=3

示。相反，这些模型学会了完美模拟系统的输入—输出动态，却没有形成底层的符号组织。图 7 显示，在 collector v2 正则语言上训练的状态空间模型经过 700 个训练轮次后，在测试数据上的轨迹接受准确率达到 100%，这表明该模型能够完美模拟模型的输入—输出关系。然而，状态空间模型处理各种输入序列时生成的状态，在欧几里得空间中并未呈现任何符号组织。

图8展示了一个经过训练、用于模拟参数化 collector v2 的状态空间模型（SSM）的潜在状态。我们从*collector v2*的 1000 条合法轨迹中，收集 SSM 的离散状态及其对应的潜在状态。我们根据这些状态的前两个主成分进行聚类，并按照离散状态为其着色。随后，我们映射真实状态各自的质心。该投影中真实状态之间的重叠表明，SSM 未能保留目标自动机的离散结构。我们使用三项指标量化聚类质量：分离比、调整兰德指数（ARI）和归一化互信息（NMI）。分离比衡量不同类别元素之间的方差与同一类别元素内部方差之比。此处，1.89 的比值表明分离效果较弱。

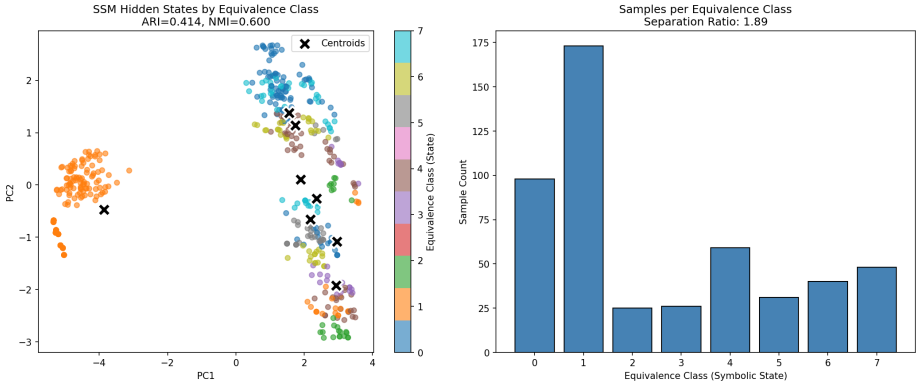

图8：处理 1,000 条随机长度为 5 的轨迹后，状态空间模型隐藏状态的主成分分析投影，按真实自动机状态着色。计数表示终止于各状态的轨迹数量。

ARI 衡量预测聚类与真实标签之间的一致程度，并对随机因素进行校正——取值为 0 表示随机分配，取值为 1 表示完全一致。NMI 同样量化聚类分配与真实标签之间的互信息，并将其归一化至 $[0,1]$。观测值（ARI $= 0.414$，NMI $= 0.600$）表明二者存在部分但并不完美的对应关系：非零的 ARI 表明 SSM 捕捉到了一些结构，但本应彼此不同的状态之间仍存在明显重叠。

这些结果表明，针对序列预测目标的梯度下降并不能以任何稳健的方式恢复离散自动机结构。尽管如此，我们在第 5.2 节中的结果表明，符号结构对于高效学习正则语言至关重要。此外，引理 1 提供了一种直接编码该结构的机制：通过用摩尔机的转移动态初始化状态空间模型，我们得到的模型既继承了符号归纳偏置，又保留了连续状态空间的表示灵活性。我们利用这两点洞见来学习超出经典自动机学习能力范围的复杂系统——尤其是那些需要无界记忆或对噪声具有鲁棒性的系统。

6 利用符号结构热启动状态空间模型

我们重新审视第 2 节中关于学习云资源分配动态仲裁策略的动机示例。我们证明，借助引理 1 利用符号初始化，能够高效学习云资源分配的动态仲裁策略。具体而言，我们将问题定义如下：

问题 *2*（动态仲裁）。给定一个有限状态仲裁器 A，我们定义一个默认遵循 A 动态的系统 A^*。然而，A^* 必须跟踪授予 g_1、g_2、$\cdots$、g_n 的授权数量历史

算法 1：从摩尔机进行状态空间模型的符号初始化

输入：摩尔机 $\mathcal{A} = (S, S_0, \Sigma, \Lambda, T, G)$**输出：**状态空间模型矩阵 A、B、C**步骤 1（噪声尺度）**。选择 $\varepsilon \in (0, 1)$，并添加高斯噪声 $\mathcal{N}(0, \varepsilon)$。将此噪声尺度添加到矩阵中的零值，以使学习更加平滑，同时保留与符号编码接近的结构。**步骤 2（状态矩阵）**。构造 $I \in \mathbb{R}^{|S|\times|S|}$

$$A \leftarrow I + \mathcal{N}(0, \varepsilon), \text{ where } A_{ij} = 0$$

步骤 3（输入矩阵）。按照证明中的方法构造 $B \in \mathbb{R}^{|S|\times(|S|\cdot|\Sigma|)}$：对于每个满足 $T(s_i, \sigma_j) = s_k$ 的 (s_i, σ_j)，将对应的列设为 $e_k - e_i$。添加噪声：

$$B_{ij} \leftarrow B_{ij} + \mathcal{N}(0, \varepsilon), \text{ where } B_{ij} = 0$$

步骤 4（输出矩阵）。按照证明中的方法构造 $C \in \mathbb{R}^{|\Lambda|\times|S|}$：对于每个输出为 $\lambda_\ell = G(s_i)$ 的状态 s_i，设置 $C_{\ell i} = 1$。添加噪声：

$$C_{ij} \leftarrow C_{ij} + \mathcal{N}(0, \varepsilon), \text{ where } C_{ij} = 0$$

返回 (A, B, C)

作为 $|g|/n$，其中 $|g|$ 表示自时间步0以来授予的授权总数。虽然 A^*默认表现为 A，但它有一项现有的安全约束：任何 g_i 都不能收到超过$(|g|/n) + k$ 份授权，其中 k为某个常数。我们使用$K = 3$进行评估，它提供了噪声最优的设置。

问题 2 对于被动和主动自动机学习而言并非易处理问题，因为该问题需要无限记忆。状态空间模型可以像循环神经网络一样利用递归，从而在仅基于有限长度轨迹 [31] 训练的情况下解决这些问题。我们表明，符号热启动是一种在该问题上使用状态空间模型获得更高准确率并加快学习的方法。我们在来自 SYNTCOMP 基准测试 [19] 的五类仲裁器上评估了我们的方法：

– arbiter zoo/parametric/arbiter.tlsf– arbiter zoo/parametric/arbiter with cancel.tlsf– full arbiter/parametric/full arbiter.tlsf– round robin arbiter/parametric/round robin arbiter.tlsf– 优先级仲裁器/parametric/prioritizedarbiter.tlsf

每个仲裁器都在多种参数化设置下进行评估，其中 $n \in \{2, \ldots, 6\}$。

6.1 方法

我们的整体神经网络架构遵循图 5 所示的架构。不过，状态空间模型层使用算法 1 进行初始化，并且我们省略了线性输入嵌入层，以保留输入的符号结构。与状态空间模型架构

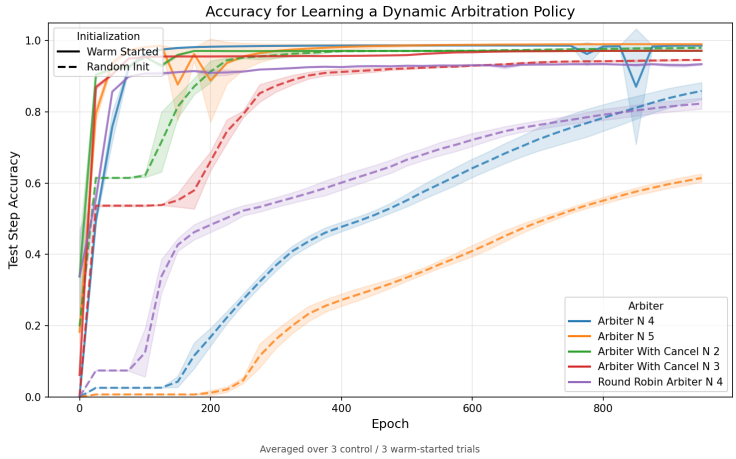

图 9：在各种仲裁器设置下学习动态仲裁策略的状态空间模型的学习与性能。热启动与随机初始化。N 表示参数化

第 5.1 节所述的架构不同，输入和输出分别表示为基于 $2^{\Sigma} \times 1$ 和 $2^{\Lambda} \times 1$ 的独热编码向量。此外，传入状态空间模型的输入表示为状态与独热编码输入的克罗内克积

$$\mu_i = S_i \otimes \alpha_i, \text{ with } S_0 \text{ being the initial state}$$

打破方程 3 中状态与输入的共同依赖，使我们能够充分运用引理 1 并构建 Moore-状态空间模型。我们使用原始的数据生成流程生成 10000 条轨迹。接着，我们遍历这些轨迹，并将其转换为符合问题 2 中定义的系统规则的形式。我们注意到，对于每个系统，新生成的数据都不会被原始自动机接受，这进一步表明我们的新系统与初始化的系统存在根本差异。此外，热启动系统在此任务上的初始准确率为 10%-50%。如果新问题与旧问题本质上相同，那么我们的初始化无需经过任何学习就能完成此任务。我们使用 9000 条轨迹进行训练，并在测试集上评估自回归任务，即预测下一个输出 λ 。

6.2 热启动评估

符号热启动使模型在各项基准测试中学习这些任务的效率大幅提升。我们将成功收敛定义为测试准确率至少达到 90%。这一门槛有意设得较高，但所有模型仍都能够达到。在所有任务中，Moore-状态空间模型平均比随机初始化的状态空间模型早 243 个训练轮次达到这一基准

。为量化统计显著性，我们采用 Mann-Whitney U 检验，该检验适用于非参数收敛时间分布。所得 p 值为 0.0122，表明这些系统的收敛速度存在统计学显著差异。

图 9 展示了使用 5 种不同仲裁方案进行学习的结果，每种方案均进行了 3 次试验。（完整结果见附录 A.3。）在所有这些情况下，热启动系统都能更高效地学习，并达到更高或相当的准确率。热启动尤其适用于学习字母表较大的系统。例如，使用热启动时，从 arbiter zoo/parametric /arbiter_tlsf 派生、包含 5 个请求和 5 个授权的系统，在 300 个训练轮次后准确率接近完美。对于同一问题，采用随机初始化的模型在经过 950 个训练轮次后，准确率仅为 60%。

尽管符号热启动在这些任务上效果显著，但它会大幅增加模型维度，可能导致内存约束。所有实验均在配备 10GB 显存的 NVIDIA RTX 3080 GPU 上进行。对于规模较大的问题实例，尤其是参数化为 $n = 5$ 和 $n = 6$ 的轮询仲裁器，符号热启动偶尔会因 GPU 内存耗尽而失败。虽然使用更大的 GPU 可能会解决这个问题，但对更大的系统进行热启动可能会带来意料之外的内存问题。我们推测，对这些 Moore-状态空间模型使用谱学习，或许能以更低的维度得到相近的系统，但这些问题留待未来研究。

7 相关工作

热启动热启动是机器学习中广泛使用的一种技术，通过用从密切相关任务中学到的表示来初始化模型，从而提高学习效率。这一思想在强化学习中尤为成功，其中对初始系统进行自举可以显著加速策略优化 [32]。热启动已成为大语言模型训练的核心，其中学习在很大程度上通过自监督微调进行 [26]。然而，先前的工作尚未探索如何利用符号结构在复杂场景中初始化学习。我们的工作首次探索了显式利用这种自动机结构的效果，并证明即使是简单的摩尔机，在学习更复杂的场景时也能发挥有效作用。

自动机-神经网络等价性神经网络与自动机之间的关系，已通过加权有限自动机（WFA）与循环神经网络（RNN）之间的形式化联系得到广泛研究。该领域以往的工作主要聚焦于从训练好的循环神经网络中提取加权有限自动机表示，以揭示其底层符号结构，并为模型行为和训练提供理论保证 [22,34]。沿着这一研究方向，诸如 L^* 等经典自动机学习算法已

扩展至加权场景，从而能够直接从训练好的循环神经网络中恢复加权有限自动机表示 [24]。

尽管取得了这些进展，现有工作主要聚焦于相对简单的神经架构及其与加权自动机的对应关系，而自动机结构如何在更具表达能力的神经网络模型中得到高效表示这一问题仍未解决。据我们所知，此前尚无工作正式建立摩尔机与状态空间模型之间的联系。

互模拟自动机之间的行为接近性概念已通过互模拟 [28]，得到形式化，它将状态划分为在所有可能的未来行为下不可区分的等价类。这种严格的等价概念支持有原则的抽象，并已被扩展到涉及大型乃至潜在无限状态系统的学习场景 [1]。基于互模拟的抽象所具有的稳健性也促成了其在强化学习中的成功应用 [30]。然而，由于互模拟通过等价而非程度来定义接近性，它在近似相似性更为自然的机器学习情境中仍然脆弱。我们的工作通过将自动机作为状态空间模型嵌入欧几里得空间来应对这一局限，从而提供一种更适合基于梯度的学习的连续接近性概念。

8 结论

本研究确立了摩尔机与一类状态空间模型之间的形式等价，从而为离散计算结构之间的行为接近性提供了连续的概念。通过将自动机嵌入欧几里得空间，我们构建了一个框架，其中符号结构可作为序列学习的归纳偏置。实验表明，与单独使用基于梯度的训练相比，符号化热启动可将样本效率提升数个数量级，并使学习能够应用于需要无限记忆的系统。未来的研究应探索如何扩展这些方法，以了解如何将形式自动机更深入地整合到神经架构中。

参考文献

1. Abate, A., Giacobbe, M., Schnitzer, Y.：互模拟学习。载于：国际计算机辅助验证会议。第 161–183 页。施普林格（2024） 2. Adam, K.D.B.J. 等：一种随机优化方法。arXiv 预印本 arXiv:1412.6980 **1412**(6)（2014） 3. Aichernig, B.K., Muˇskardin, E., Pferscher, A.：主动与被动：网络协议自动机学习范式的比较。arXiv 预印本 arXiv:2209.14031（2022） 4. Aichernig, B.K., Tappler, M., Wallner, F.：自动机学习中学习算法与测试算法组合的基准测试。计算的形式方面 **36**(1)，1–37（2024）

5. Almarhabi, A.K.：分布式仲裁器：一种轻量且增强型的云计算访问控制机制。《热科学》**28**(6 Part B)，4969–4977（2024） 6. Angluin, D.：通过查询和反例学习正则集合。《信息与计算》**75**(2)，87–106（1987） 7. Babiak, T., Blahoudek, F., Duret-Lutz, A., Klein, J., Kˇret´ınsk`y, J., M¨uller, D., Parker, D., Strejˇcek, J.：Hanoi ω 自动机格式。载于：国际计算机辅助验证会议。第 479–486 页。施普林格（2015） 8. Boubaker, S., Mammar, A., Graiet, M., Gaaloul, W.：使用 Event-B 对业务流程中的云资源分配进行形式化验证。载于：2016 年 IEEE 第 30 届国际高级信息网络与应用会议（AINA）。第 746–753 页。IEEE（2016） 9. Bourdois, L.：状态空间模型（SSM）简介。Hugging Face 博客（2024 年 7 月），https://huggingface.co/blog/lbourdois/get-on-the-ssm-train 10. Di Stefano, L.：使用 HOAX 执行和监控 HOA 自动机。载于：国际运行时验证会议。第 44–53 页。施普林格（2025） 11. Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Gbaguidi Aisse, A., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C. 等：从 Spot 2.0 到 Spot 2.10：有哪些新变化？载于：国际计算机辅助验证会议。第 174–187 页。施普林格（2022） 12. Fan, G., Yu, H., Chen, L.：一种用于云计算中自适应资源调度建模与分析的形式化面向方面方法。《IEEE 网络与服务管理汇刊》**13**(2)，281–294（2016） 13. Fremont, D.J., Kim, E., Pant, Y.V., Seshia, S.A., Acharya, A., Bruso, X., Wells, P., Lemke, S., Lu, Q., Mehta, S.：自动驾驶汽车的形式化基于场景测试：从仿真到现实世界。载于：2020 年 IEEE 第 23 届国际智能交通系统会议（ITSC）。第 1–8 页。IEEE（2020） 14. Gansner, E.R.：使用 Graphviz 绘制图。技术报告，AT&T 贝尔实验室，默里，技术报告，技术报告（2009） 15. Garfatta, I., Klai, K., Graiet, M., Gaaloul, W.：业务流程中云资源分配的形式化建模与验证。载于：OTM 联合国际会议“迈向有意义的互联网系统”。第 552–567 页。施普林格（2018） 16. Gu, A., Dao, T.：Mamba：使用选择性状态空间进行线性时间序列建模。载于：首届语言建模会议（2024） 17. Hamilton, J.D.：状态空间模型。《计量经济学手册》**4**，3039–3080（1994） 18. Imambi, S., Prakash, K.B., Kanagachidambaresan, G.：PyTorch。载于：《使用 TensorFlow 编程：边缘计算应用解决方案》，第 87–104 页。施普林格（2021） 19. Jacobs, S., Perez, G.A., Schlehuber-Caissier, P.：来自 SyntComp 2023 的数据、脚本和结果。8161423 20 2023 10 5281 ( )。https://doi.org/ . /zenodo. . Klimovich, A., Solov’ev, V.：通过拆分内部状态将米利有限状态机转换为穆尔有限状态机。《国际计算机与系统科学杂志》**49**(6)，900–908（2010） 21. Lei, Y., Cai, Z., Li, X., Buyya, R.：基于状态空间模型和排队网络的网状 Web 系统云资源配置。《IEEE 并行与分布式系统汇刊》**33**(12)，3787–3799（2022） 22. Li, T., Precup, D., Rabusseau, G.：通过谱学习连接加权自动机、张量网络与循环神经网络。《机器学习》**113**(5)，2619–2653（2024）

23. Muˇskardin, E., Aichernig, B.K., Pill, I., Pferscher, A., Tappler, M.：Aalpy：一个主动自动机学习库。《系统与软件工程创新》**18**(3), 417–426 (2022) 24. Okudono, T., Waga, M., Sekiyama, T., Hasuo, I.：通过对状态空间进行回归，从循环神经网络中提取加权自动机。AAAI'20 论文集 (2020) 25. Peng, Y., Bao, Y., Chen, Y., Wu, C., Meng, C., Lin, W.：DL2：一种面向深度学习集群的深度学习驱动调度器。《IEEE 并行与分布式系统汇刊》**32**(8), 1947–1960 (2021) 26. Rani, V., Nabi, S.T., Kumar, M., Mittal, A., Kumar, K.：自监督学习：简明综述。《计算工程方法档案》**30**(4), 2761–2775 (2023) 27. Renkin, F., Schlehuber-Caissier, P., Duret-Lutz, A., Pommellet, A.：剖析 ltl-s ynt。《系统设计形式化方法》**61**(2), 248–289 (2022) 28. Sangiorgi, D.：互模拟与余归纳的起源。《ACM 编程语言与系统汇刊》(TOPLAS) **31**(4), 1–41 (2009) 29. Sharara, M., Hoteit, S., V`eque, V.：一种基于循环神经网络的云无线接入网无线与计算资源分配协调方法。载于：2021 IEEE 第 22 届高性能交换与路由国际会议 (HPSR)。第 1–7 页。IEEE (2021) 30. Shi, L., Jianye, H., Tang, H., Dong, Z., Zheng, Y.：用于高效强化学习的自监督互模拟操作块表示。载于：NeurIPS 2024 安全生成式人工智能研讨会 (2024) 31. Siegelmann, H.T., Sontag, E.D.：论神经网络的计算能力。载于：第五届计算学习理论年度研讨会论文集。第 440–449 页 (1992) 32. Tio, S., Li, W., Karunasena, R., Jimmy, H.T.S., Varakantham, P.：从机器学习到人类学习：探索利用强化学习智能体热启动教师算法。OpenReview (2025) 33. Tˆırnˇaucˇa, C.：极限中的 DFA 识别状态合并策略综述。Triangle：语言、文学、计算 pp. 121–136 (2012) 34. Wei, Z., Zhang, X., Sun, M.：从循环神经网络中提取自然语言的加权有限自动机。载于：形式化工程方法国际会议。第 370–385 页。施普林格 (2022) 35. Zi, Y.：利用循环神经网络进行云资源分配的时间序列负载预测。《计算机技术与软件杂志》**3**(7) (2024)

A 图

A.1 样例轨迹和日志

A.2 SYNTCOMP基准测试结果

```
g_0 &! c_0&r_0_&! g_1_&! c_1&r_1 ;_! g_0&c_0 &!_r_0&g_1 &! c_1 &! r_1 ; 循环 {1}
```

图10：使用 Spot 语义和循环的带取消功能参数化仲裁器的样本迹{1}

```
{"tlsf文件" : "基准测试/tlsf2/sweap/arbiter_—与—失败 —真实.tlsf" , "试验" : 2 ,"样本大小" : 154482 , "准确率" : 1 0 0 . 0 , "状态" : "成功"}
```

图11：主动自动机学习一次试验的样本输出

表3：SYNTCOMP 基准测试上的学习方法比较。

基准测试	状态空间模型 准确率 (%)	L*算法 准确率 (%)	RPNI 准确率 (%)	状态空间模型 样本	L*算法 样本	RPNI 样本
01	26.9	100.0	100.0	8,100,000	6,346	5,000
02	7.6	100.0	100.0	8,100,000	6,440	5,000
03	2.3	99.0	0.0	8,100,000	79,764	5,000
EscalatorSmart	0.0	100.0	100.0	8,100,000	6,959	5,000
Gamemodule	0.2	100.0	100.0	8,100,000	4,721	5,000
KitchenTimerV1	100.0	100.0	100.0	4,500,000	1,878	5,000
KitchenTimerV2	39.4	100.0	100.0	8,100,000	7,843	5,000
KitchenTimerV3	1.1	100.0	93.5	8,100,000	11,775	30,000
KitchenTimerV5	0.0	100.0	12.4	8,100,000	23,099	10,000
MusicAppFeedback	0.0	100.0	9.6	8,100,000	20,296	25,000
MusicAppSimple	3.0	100.0	100.0	8,100,000	5,048	5,000
SPIReadSdi	0.0	100.0	100.0	8,100,000	4,759	5,000
SPIWriteManag	10.1	0.0	100.0	8,100,000	6,533	5,000
SPIWriteSdi	0.3	0.0	100.0	8,100,000	8,624	5,000
SliderDefault	51.2	100.0	0.0	8,100,000	1,974	5,000
SliderScored	0.0	100.0	100.0	8,100,000	6,267	5,000
TorcsGearing	100.0	100.0	100.0	1,800,000	912	5,000
abcg_arbiter	100.0	100.0	100.0	3,600,000	921	5,000
arbiter	100.0	100.0	100.0	1,800,000	888	5,000
box — limited — real	0.6	100.0	96.6	8,100,000	6,563	20,000
box — real	30.3	100.0	100.0	8,100,000	291,353	5,000
chomp	1.5	100.0	100.0	8,100,000	6,044	5,000
collector_v1	81.8	100.0	0.3	8,100,000	4,484	25,000
collector_v2	100.0	100.0	6.5	8,100,000	4,345	30,000
collector_v3	100.0	100.0	100.0	8,100,000	3,877	5,000
diagonal — real	7.4	100.0	98.2	8,100,000	53,711	10,000
evasion — real	82.3	0.0	100.0	8,100,000	—	5,000
example10a	100.0	100.0	100.0	4,500,000	1,046	5,000
example10	100.0	100.0	6.7	4,500,000	959	15,000
example7a	100.0	100.0	100.0	4,500,000	1,883	5,000
example7	100.0	100.0	100.0	900,000	1,857	5,000
example8	100.0	100.0	100.0	1,800,000	1,870	5,000
example9	100.0	100.0	100.0	2,700,000	925	5,000
f — real — real	11.2	0.0	100.0	8,100,000	—	5,000
full_arbiter	100.0	100.0	100.0	3,600,000	1,912	5,000
g — real — real	88.1	0.0	100.0	8,100,000	—	15,000
gf — real — real	17.4	94.9	2.7	8,100,000	72,393	15,000
heim — buechi — real	98.9	0.0	100.0	8,100,000	—	5,000
lift_gr1+	0.0	100.0	44.2	8,100,000	19,565	30,000
lift_gr1	2.4	100.0	100.0	8,100,000	4,758	10,000
lift_unary_enc	1.1	100.0	100.0	8,100,000	2,135	5,000
lift	0.1	100.0	100.0	8,100,000	1,976	5,000
lilydemo03	100.0	100.0	100.0	900,000	3,768	5,000
lilydemo04	100.0	100.0	39.1	2,700,000	3,879	25,000
lilydemo05	100.0	100.0	100.0	3,600,000	3,850	5,000
lilydemo06	100.0	100.0	100.0	6,300,000	4,008	5,000
lilydemo07	100.0	100.0	100.0	4,500,000	3,931	5,000
lilydemo08	100.0	100.0	100.0	900,000	905	5,000
lilydemo09	100.0	100.0	100.0	6,300,000	950	5,000
lilydemo10	100.0	100.0	100.0	900,000	1,829	5,000
lilydemo14	100.0	100.0	100.0	900,000	1,844	5,000
lilydemo15	100.0	100.0	100.0	8,100,000	1,923	5,000
lilydemo16	60.2	100.0	100.0	8,100,000	5,455	5,000
lilydemo17	100.0	100.0	100.0	1,800,000	1,857	5,000
lilydemo18	100.0	100.0	100.0	1,800,000	3,880	5,000
lilydemo19	100.0	100.0	100.0	2,700,000	1,871	5,000
lilydemo20	75.3	100.0	100.0	8,100,000	1,833	5,000
lilydemo21	0.6	99.4	100.0	8,100,000	25,482	10,000
lilydemo22	40.6	100.0	100.0	8,100,000	5,026	5,000

续见下页

表3 – 接续上一页

基准测试	状态空间模型 准确率（%）	L*算法 准确率（%）	RPNI 准确率（%）	状态空间模型 样本	L*算法 样本	RPNI 样本
lilydemo23	100.0	100.0	100.0	1,800,000	913	5,000
load_balancer	74.2	100.0	100.0	8,100,000	4,020	5,000
ltl2dba01	72.3	100.0	36.4	8,100,000	3,818	10,000
ltl2dba02	54.8	100.0	48.5	8,100,000	5,515	30,000
ltl2dba03	98.2	100.0	21.1	8,100,000	3,902	5,000
ltl2dba04	99.5	100.0	0.5	8,100,000	3,802	5,000
ltl2dba05	63.6	100.0	15.6	8,100,000	6,040	20,000
ltl2dba06	19.6	100.0	13.6	8,100,000	5,942	15,000
ltl2dba07	48.2	100.0	14.9	8,100,000	156,904	15,000
ltl2dba10	100.0	100.0	100.0	2,700,000	1,852	5,000
ltl2dba12	84.2	100.0	0.0	8,100,000	1,921	5,000
ltl2dba13	100.0	100.0	100.0	4,500,000	3,730	5,000
ltl2dba14	78.8	100.0	0.5	8,100,000	3,797	10,000
ltl2dba16	31.5	100.0	100.0	8,100,000	1,821	5,000
ltl2dba18	23.1	100.0	11.3	8,100,000	3,889	10,000
ltl2dba19	100.0	100.0	6.5	6,300,000	2,090	5,000
ltl2dba20	7.6	100.0	8.4	8,100,000	5,989	20,000
ltl2dba22	100.0	100.0	0.0	4,500,000	915	5,000
ltl2dba23	100.0	100.0	0.0	8,100,000	1,824	5,000
ltl2dba25	69.9	100.0	0.4	8,100,000	3,800	15,000
ltl2dba26	58.4	100.0	8.8	8,100,000	3,772	20,000
ltl2dba_E	69.1	100.0	68.1	8,100,000	1,912	5,000
ltl2dba_Q	67.2	100.0	90.3	8,100,000	2,011	10,000
ltl2dba_U1	100.0	100.0	53.2	2,700,000	1,859	25,000
ltl2dba_alpha	100.0	100.0	100.0	3,600,000	1,870	5,000
ltl2dba_beta	62.5	100.0	52.4	8,100,000	7,631	10,000
ltl2dpa01	100.0	100.0	100.0	2,700,000	934	5,000
ltl2dpa02	100.0	100.0	8.7	3,600,000	1,814	5,000
ltl2dpa03	25.7	100.0	100.0	8,100,000	12,887	10,000
ltl2dpa04	100.0	100.0	100.0	3,600,000	1,820	5,000
ltl2dpa06	100.0	100.0	0.0	4,500,000	3,732	5,000
ltl2dpa07	100.0	100.0	7.9	4,500,000	3,693	15,000
ltl2dpa08	100.0	100.0	75.6	7,200,000	1,862	5,000
ltl2dpa09	1.7	100.0	8.6	8,100,000	3,709	5,000
ltl2dpa10	79.1	99.9	100.0	8,100,000	2,937	5,000
ltl2dpa12	24.6	100.0	100.0	8,100,000	2,648	5,000
ltl2dpa13	79.8	100.0	64.7	8,100,000	2,469	5,000
ltl2dpa14	100.0	100.0	100.0	2,700,000	917	5,000
ltl2dpa16	88.0	100.0	69.0	8,100,000	1,899	10,000
ltl2dpa17	63.2	100.0	0.0	8,100,000	3,889	5,000
ltl2dpa18	85.6	100.0	0.0	8,100,000	1,855	5,000
ltl2dpa19	49.2	100.0	68.7	8,100,000	4,417	10,000
ltl2dpa20	77.2	100.0	69.5	8,100,000	1,918	25,000
ltl2dpa23	100.0	100.0	100.0	1,800,000	1,787	5,000
ltl2dpa24	100.0	100.0	100.0	1,800,000	1,819	5,000
robot — tasks — real	100.0	0.0	100.0	900,000	—	5,000
robot_grid	0.0	100.0	35.3	8,100,000	16,342	30,000
rw_arbiter	100.0	100.0	37.5	5,400,000	1,860	5,000
solitary — real	99.2	100.0	100.0	8,100,000	16,781	5,000
tasks — real	99.2	0.0	100.0	8,100,000	—	5,000
EscalatorBidirectional	0.0	0.0	100.0	8,100,000	37,258	5,000
EscalatorCountingInit	33.6	100.0	100.0	8,100,000	3,766	5,000
EscalatorNonReactive	100.0	0.0	100.0	900,000	—	5,000
MusicAppMotivating	2.1	100.0	2.6	8,100,000	8,564	5,000
OneCounterInRangeA3	1.8	100.0	100.0	8,100,000	5,490	5,000
TorcsAccelerating	100.0	100.0	100.0	900,000	1,799	5,000
TorcsSteeringSimple	0.2	100.0	100.0	8,100,000	4,726	5,000
TorcsSteeringSmart	0.0	100.0	22.2	8,100,000	6,314	25,000
arbiter — with — failure— real	1.8	100.0	39.2	8,100,000	154,618	5,000
arbiter_on_inpchange	100.0	100.0	5.9	4,500,000	942	25,000
arbiter_with_cancel	100.0	100.0	0.1	1,800,000	1,780	10,000
batch — arbiter — real	20.4	100.0	100.0	8,100,000	333,487	5,000
chain — simple — 10— real	94.8	0.0	63.0	8,100,000	—	15,000
chain — simple — 20— real	93.0	0.0	63.7	8,100,000	—	15,000
chain — simple — 30— real	96.3	0.0	63.1	8,100,000	—	20,000
chain — simple — 40— real	94.6	0.0	64.2	8,100,000	—	20,000
chain — simple — 5 — real	96.0	0.0	42.7	8,100,000	—	5,000
chain — simple — 50— real	94.2	0.0	64.7	8,100,000	—	5,000
chain — simple — 60— real	94.4	0.0	64.0	8,100,000	—	5,000
chain — simple — 70— real	95.1	0.0	64.2	8,100,000	—	25,000
chain — simple — param— 70 — real	97.7	0.0	100.0	8,100,000	—	5,000
generalized_buffer	4.2	99.9	0.0	8,100,000	509,491	5,000
heim — double — x — real	99.4	0.0	100.0	8,100,000	—	5,000
infinite — race — real	97.0	100.0	100.0	8,100,000	9,989	5,000
infinite — race — unequal— 1 — real	98.8	0.0	100.0	8,100,000	—	5,000
load_balancer_unreal2	72.8	100.0	100.0	8,100,000	3,961	5,000

续于下一页

表3——续自上一页

基准测试	状态空间模型准确率（%）	L*算法准确率（%）	RPNI 准确率（%）	状态空间模型样本	L*算法样本	RPNI 样本
ordered — visits — real	100.0	100.0	100.0	8,100,000	266,715	5,000
precise — reachability— real	88.3	100.0	98.6	8,100,000	531,210	30,000
robot — cat — real — 1d— real	98.5	0.0	100.0	8,100,000	—	5,000
robot — grid — commute— 1d — real	83.3	0.0	100.0	8,100,000	—	5,000
robot — grid — reach— 1d — real	97.6	100.0	100.0	8,100,000	135,518	5,000
robot — grid — reach— 2d — real	78.9	0.0	93.3	8,100,000	—	15,000
robot — grid — reach— repeated — with — obstacles— 1d — real	95.3	100.0	100.0	8,100,000	201,089	5,000
robot — grid — reach— repeated — with — obstacles— 2d — real	93.3	0.0	100.0	8,100,000	—	5,000
robot_collect_samples_v2— real	99.0	0.0	100.0	8,100,000	—	5,000
storage — GF — 64 — real	92.3	96.3	82.2	8,100,000	1,053,092	20,000
thermostat — F — real	99.8	0.0	100.0	8,100,000	—	5,000
unordered — visits— real	0.6	0.0	14.6	8,100,000	—	30,000
workstation_resupply_pb_1_pe_	56.7	100.0	100.0	8,100,000	925	5,000
workstation_resupply_pb_2_pe_	58.4	100.0	0.1	8,100,000	2,108	20,000
workstation_resupply_pb_3_pe_	36.0	100.0	0.2	8,100,000	5,189	10,000

{”tlsf文件”：“基准测试/tlsf/tslpaper/KitchenTimerV3.tlsf”，“试验”：1，“轨迹长度”：20，“结果”：
[{”轨迹数量”：5000，“准确率”：11.2}，{ 10000， 26 0} ”轨
迹数量”：“准确率”：.，{ 15000， 28 6} ”轨迹数量”：“准确率”：.，
{ 20000， 13 9} ”轨迹数量”：“准确率”：.，
{ 25000， 35 6} ”轨迹数量”：“准确率”：.，
{ 30000， 93 5}]”轨迹数量”：“准确率”：.，“最终准确率”：
93.5，“最终轨迹”：30000，“状态”：“成功”}

图 12：对来自 SYNTCOMP 的 tsl paper/KitchenTimerV3 进行被动学习试运行的输出

{”文件”：”/tlsf/parametric/abcgarbiter.tlsf”，”训练样本”：10000，”收敛
训练轮次”：400，”测试轨迹准确率”：1.0，”训练轮次历史”：
{ 0， ”训练轮次”：”损失”：0.6949，”测试步准确率”：
0.1643，”测试轨迹准确率”：0.0}，
{ 100， 0 4375， ”训练轮
次”：”损失”：。”测试步准确率”：0.3726，”测试轨迹准确率”：0.0}，
{ 200， 0 4125， ”训练轮
次”：”损失”：。”测试步准确率”：0.3726，”测试轨迹准确率”：0.0}，
{ 300， 0 1361， ”训练轮
次”：”损失”：。”测试步准确率”：0.985，”测试轨迹准确率”：0.733}，
{ 400， 0 0411， ”训练
轮次”：”损失”：。”测试步准确率”：1.0，”测试轨迹准确率”：1.0 }]}

图13：在参数化/abcg 仲裁器训练运行中训练的状态空间模型

A.3 自适应仲裁结果

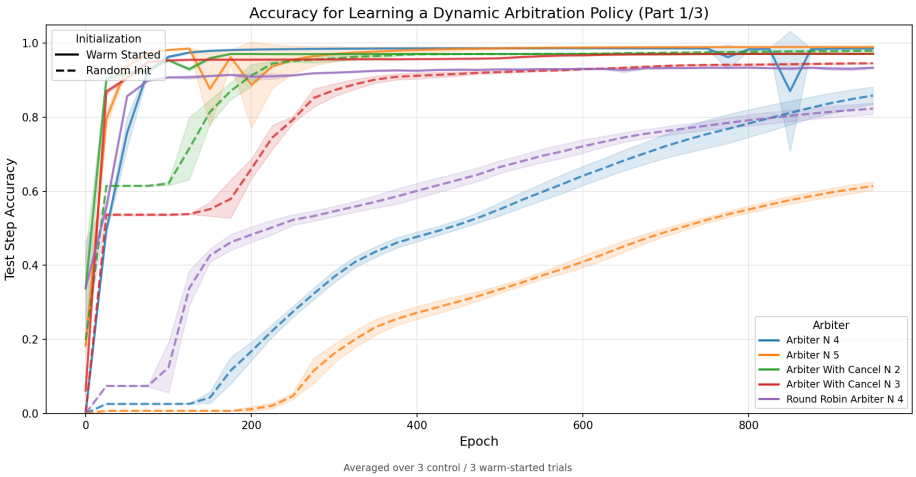

图 14：自适应仲裁器结果 1。详见正文

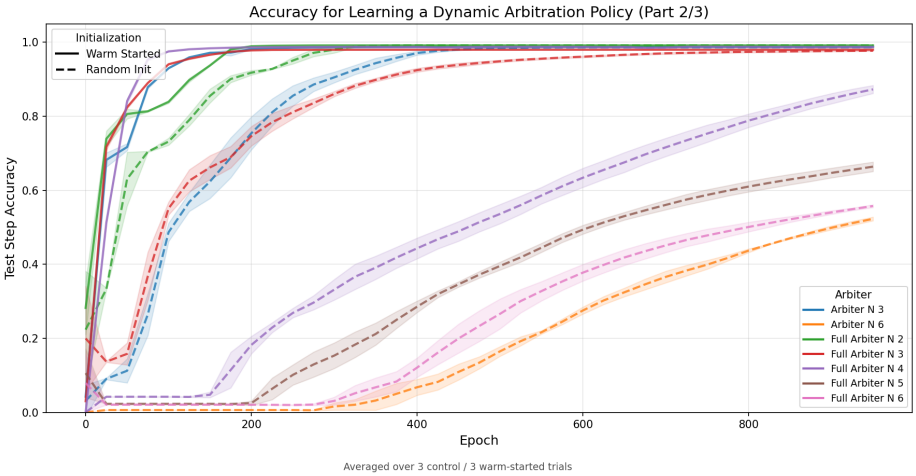

图 15：自适应仲裁器结果 2。

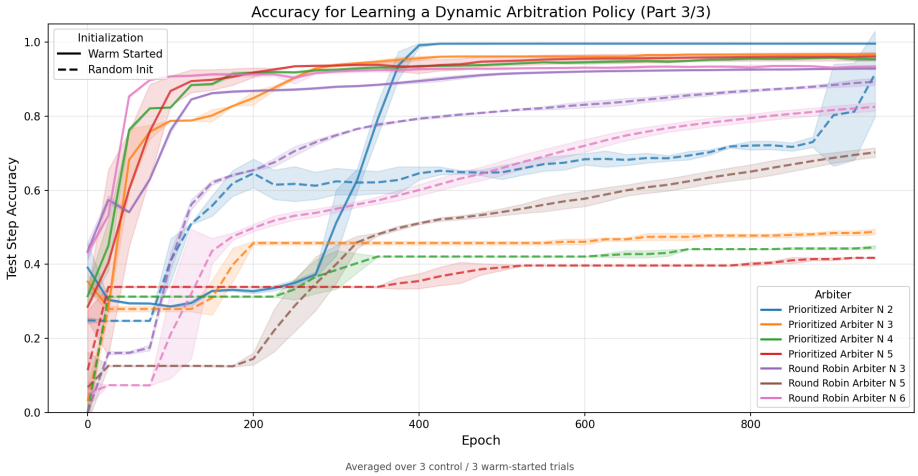

图 16：自适应仲裁器结果 3。