本文已被第 24 届 IEEE 国际机器学习与应用会议（ICMLA 2025）录用发表。

《无关机器调度的深度学习》：处理可变维度

Diego Hitzges柏林工业大学数学系¨柏林柏林，德国0009-0000-3978-0591

Guillaume Sagnol柏林工业大学数学系¨柏林柏林，德国0000-0001-6910-8907

arXiv:2512.19527v1 [cs.LG] 22 Dec 2025

摘要——**深度学习已成功应用于许多离散优化问题。然而，针对无关并行机的学习型调度仍然特别难以设计。不仅作业数量和机器数量会变化，而且每个作业—机器对都有独特的处理时间，导致特征维度动态变化。我们提出了一种新方法，采用专为任意规模无关机器上的离线确定性调度设计的神经网络。其目标是最小化一个复杂的目标函数，该函数包含最大完工时间，以及作业和机器的加权拖期。与按顺序处理作业的现有在线方法不同，我们的方法一次性考虑整个输入，生成完整的调度方案。本研究的主要贡献在于模型精巧的架构。通过利用多种受自然语言处理启发的架构，该模型能够有效处理任意数量的作业和机器，并应对无关处理时间所带来的不同特征维度。我们的方法支持在小型问题实例上进行监督训练，同时在规模大得多的调度环境中展现出良好的泛化能力。在包含 8 个作业和 4 台机器的实例上训练和测试时，成本仅比最优值高出 2.51%。在所有测试配置中，最多包含 100 个作业和 10 台机器时，我们的网络始终优于先进的派工规则；后者的平均成本高出 22.22%。由于我们的方法可以利用模拟数据快速重新训练，并适应各种调度条件，我们认为它有望成为无关机器及类似问题环境中学习型调度的一种标准方法。**

I. 引言

离散优化问题不仅具有数学研究价值，也因其对现实应用的影响而具有实际意义。作业调度是提升效率方面尤其有前景的一个领域。在这一背景下，任务（作业）被分配给适当的资源（机器），以尽可能降低特定目标函数所规定的成本。与许多组合问题一样，随着复杂度增加，作业调度很快就会变成 NP-困难问题 [1]。近似算法大多针对简化场景开发，并且适用于定义狭窄的问题设置，因此缺乏泛化能力。深度学习为应对这些挑战提供了有前景的途径。然而，传统神经网络结构往往难以处理调度问题固有的输入和输出维度可变性。考虑到作业和机器的数量会发生变化，这种灵活性尤为重要，尤其是在必须同时处理所有数据的离线场景中。自然语言处理（NLP）领域已有效解决了对这类适应能力的需求。本文提出一种新颖的网络架构，整合了多种受自然语言处理启发的结构，能够灵活处理任意数量的作业和机器，同时也能应对特征维度的变化。通过应用这一架构，我们为复杂的调度环境提供了一种离线解，并且该方案还可适应不同的问题设置。借此，我们旨在克服数学方法和基于学习的方法各自的不足。

A. 问题表述

我们研究一个确定性离线问题：在无关并行机上对 J 个作业进行不可抢占调度，记为 R_m。每个作业 j 在每台机器 m 上都有特定的加工时间 $p_{jm} > 0$，以及一个单独的截止时间 d_j 和超出该截止时间时的惩罚权重 w_j。类似地，每台机器 m 都有一个停用截止期限 δ_m 及相应的惩罚权重 ω_m。此外，机器开始时已有运行时间为 r_m 的初始占用时间。目标是最小化最大完工时间 C_{max} 与作业 T_j 和机器 τ_m 的总加权拖期之和。由此得到以下问题表述：

$$R_m \mid d_j, \delta_m, w_j, \omega_m, r_m \mid C_{\max} + \sum_{j=1}^{J} w_j T_j + \sum_{m=1}^{M} \omega_m \tau_m. \quad (1)$$

B. 我们的方法

为了将深度学习有效应用于我们复杂的调度问题，我们将其嵌入一个适合神经网络运行的框架中。为此，我们将任务重新表述为一个确定性的马尔可夫决策问题（MDP）。在该框架中，每当一台机器变为可用状态时，就将这一时刻表示为一个状态，并需要采取动作，将一个作业分配给该机器，或将其停用。从代表给定问题环境的初始状态出发，通过反复应用神经网络估计每个状态下的最优动作，系统地生成调度方案。由于每个状态都包含有关所有待处理作业和活动机器的全面信息，因此该方法可有效地作为离线解。此外，得益于框架的马尔可夫性质，每个后续状态都会形成一个独立的

调度场景，因此也支持在线应用和修改。我们将证明，我们的网络能够有效泛化到远大于训练所用问题规模的实例。尽管我们的框架兼容强化学习（RL），但网络出色的泛化能力使我们能够仅使用非常小规模的实例，通过监督学习对其进行有效训练；在这些实例中，精确计算状态-动作价值的成本很低。这样既能实现高精度决策，也能针对新的问题设置快速重新训练。训练好的模型和源代码已公开发布在在线代码库中 [2]以支持透明性、可复现性和实际应用。

II. 相关工作

我们的工作研究无关机器调度问题（1），其目标函数为复合目标函数，重点是同时最小化最大完工时间 C_{max}（即使在两台同构机器上，这一问题也已是 NP 难问题），以及无关并行机上作业的总加权拖期。即使在规模不大的环境中（约 5 台机器、30 个作业），该问题也极具挑战性 [1，页 112，143]。除基于动态规划或混合整数规划的精确求解方法（这些方法只能处理非常小规模的问题）外，人们还提出了多种启发式方法来解决这类问题；参见 [3]及其中的参考文献。这些启发式方法通常基于派工规则，通过优先级函数确定作业顺序；例如，参见 [4]，；或采用进化算法，例如 [5]使用蚁群优化算法的方法。近年来，基于学习的算法也开始出现。

现有关于基于学习的调度的大多数文献主要采用强化学习（RL）或深度强化学习（DRL），例如 [6] 使用深度学习来最小化单机上的总拖期。尽管我们的框架与DRL兼容，但鉴于监督学习具有更快的收敛速度和更高的准确率，我们采用了监督学习，从而提供了一种可能更为通用的解决方案，可针对不同的问题场景快速重新训练以实现适应。监督学习也曾被用于 [7]，，但仅限于流水车间问题中的在线应用。已有多项研究针对作业车间或流水车间问题 [8]–[10]。事实上，一项针对机器调度中强化学习的全面文献综述发现，在选定的80篇出版物中，有50篇涉及这些主题 [11]。在剩余的 30 项研究中，只有少数聚焦于无关并行机问题。所有这些研究都规避了处理大小可变的状态空间和动作空间这一问题：要么仅将剩余作业中的一个作业的信息输入神经网络 [12]，，要么将状态信息聚合为少量特征，并将动作空间限制为选择某一派工规则（而不是精确确定应启动哪个作业） [13]，[14]。此外，许多关于组合优化问题中基于学习的方法的综述论文和调查倾向于涵盖生产系统中的强化学习应用，或概述如何将深度强化学习应用于组合问题的不同网络架构，却未详细说明其在机器调度问题中的应用 [15]。这凸显出，针对复杂的非相关并行机调度问题，现有研究缺乏兼具适应性和针对性的基于学习的解决方案，而本文正试图弥补这一空白。

III. 方法

本节将概述如何把我们的调度问题（1）转换为适用于深度学习应用的马尔可夫决策过程框架。我们将详细介绍训练过程和度量指标；由于缺少基准，我们还将构建一种高级派工规则，其优先级函数针对特定的复合目标函数进行了定制，以评估网络的性能。

A. 状态作为输入

在调度过程中，每当一台机器空闲时，我们就会将状态构建为神经网络的输入。在每个状态 $s \in S$ 中，机器首先按剩余运行时间排序，其次按权重从高到低排序，最后按截止时间从早到晚排序。同样，排列

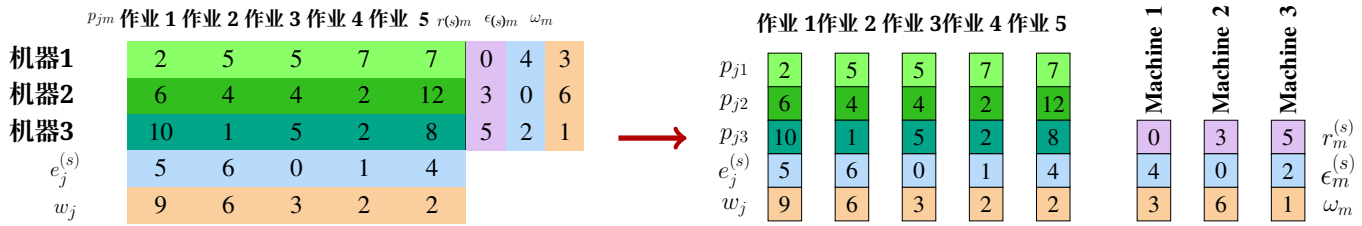
$$\Pi_m : \{1, \ldots, J\} \mapsto \{1, \ldots J\} \tag{2}$$

随后应用于作业，首先按空闲机器上的加工时间从短到长排序 m ，其次按权重从高到低排序，最后按截止时间从早到晚排序。每个状态都由每个待处理作业和运行中的机器对应的向量表示。机器的向量为三维，包含剩余运行时间 $r_m^{(s)}$、提前时间 $\epsilon_m^{(s)}$——即距离机器截止时间的时间——以及权重 ω_m。作业向量包含 p_{jm} 每台运行中机器的加工时间 m，以及作业 $'$的提前时间 $e_j^{(s)}$ 和权重 w_j。图1展示了如何从状态中提取数据。

B. 动作值作为目标

在我们的马尔可夫决策过程中，每个状态 $s \in S$ 对应一个决策点，此时一台机器变为可用。在每个状态下，可行操作 $A(s)$ 包括将任意一个剩余作业分配给该机器，或者在存在其他运行中的机器时将其停用，因为与立即分配作业相比，空闲时间始终不是最优选择。因此，如果可以停用机器，则每个状态下的动作总数等于待处理作业的数量加一。只要没有待处理作业，就达到终止状态。对于我们的神经网络，一个直观的目标是贝尔曼方程中的Q值，据此可以立即推导出最优策略 μ^* 。不过，为了着重区分最佳选项并舍弃较差选项，我们将目标值 $y^{(s)}(a)$ 定义为：用最优成本 $V^*(s)$ 对Q值进行反向缩放= $V_{\mu^*}(s)$ ，并应用Softmax：

$$y^{(s)}(a) := \text{softmax}(\frac{V^*(s)}{Q^*(s,a)}) \in [0,1]. \tag{3}$$



(a) 排序后的作业和机器 (b) 状态数据

图1：从状态信息到数据

C. 神经网络作为策略

由参数 ϑ 定义的神经网络所生成的调度算法可应用于任何由方程（1）定义的调度问题。要生成调度方案，请将问题转换为具有相应初始状态 $s \in S$的马尔可夫决策过程，并应用网络导出的策略 μ_ϑ。具体步骤如下：

1）将状态数据输入神经网络。2）从可行操作的输出分布中选择估计概率最高的动作。3）执行所选动作，转移到后继状态。

重复此过程，直到达到最终状态。由此产生的动作-状态对序列构成一个可行的调度方案，其目标成本等同于策略成本 V_{μ_ϑ}。由于 μ_ϑ近似于最优策略 μ^*，因此得出的调度方案可用于估计最优调度方案。

D. 数据模拟与监督训练

我们对神经网络采用监督训练，从而缩短训练时间，并提高模型的预测准确率和可靠性。我们根据以下参数，模拟 100.000 个形式为（1）且包含 8 个作业和 4 台机器的调度问题：

- 最大权重 $w_{\max} = 10$
- 最大截止时间： $d_{\max} = 30$ （相当于一个月）
- 最大加工时间：$p_{\max}$ 从以下范围中随机选取$[\frac{1}{3}d_{\max}\ \frac{3}{2}d_{\max}]$
- 机器的最大初始运行时间：从以下范围中随机选取$[\frac{1}{3}p_{\max}\ p_{\max}]$

作业和机器的处理时间、截止时间、权重及运行时间均在这些范围内随机生成。如果初始时没有机器空闲，则随机将一台机器的运行时间设为零。对于每个调度问题，我们计算所有可能的状态及其真实动作价值。随后，我们从中选取 18 个状态，分别对应待处理作业数量为 3 至 8、活动机器数量为 2 至 4 的各类组合。为确保数据集均衡，我们根据各可行动作中的最优动作，均匀地选取这些状态。通过以下度量指标，衡量每个状态—动作对的相对成本增幅，并将我们参数为 ϑ 的模型性能与最优成本进行比较：

$$\Delta Q(s, \mu_\vartheta(s)) := \frac{Q^*(s, \mu_\vartheta(s)) - V^*(s)}{V^*(s)}. \tag{4}$$

我们选择均方误差（MSE）作为损失函数，而非交叉熵，以便精确惩罚与近乎最优动作之间的偏差。训练使用 Python3 和 TensorFlow 完成。我们采用学习率为 0.001、批次大小为 128 的 Adam 优化器。网络在一台配备 AMD Ryzen 5 3500 中央处理器（2.1 GHz、4 核、8 线程）、8 GB 随机存取存储器和 80 GB 固态硬盘的私人笔记本电脑上训练 30 个周期，耗时约 24 小时。

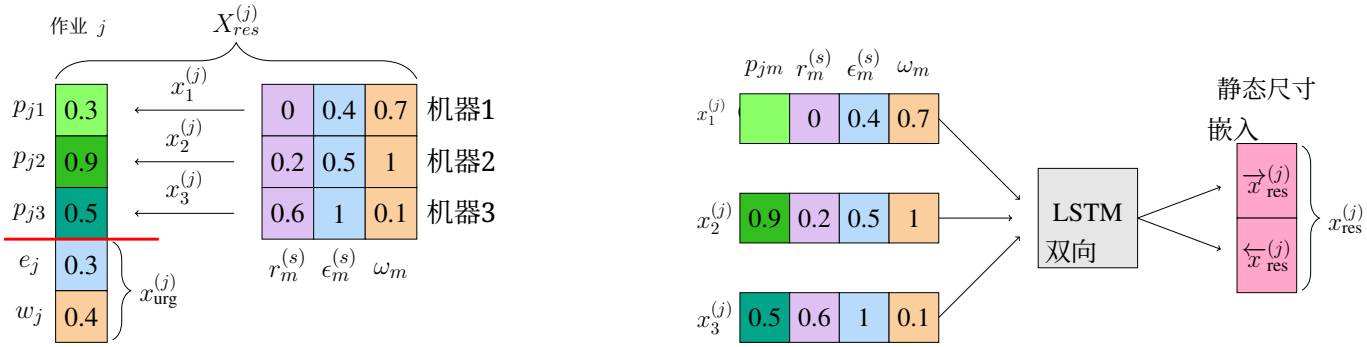
E. 基于学习的方法与启发式调度

我们设计了一种先进的派工规则，并针对特定的复合目标函数定制了优先级函数，用以评估神经网络在作业和机器数量各异的调度问题中的性能。派工规则，例如 [3]，中所述的方法，已被证明能够为复杂调度问题提供有效的启发式方法。受贪心列表调度的启发，这种方法在同构机器调度中的多个目标函数上已展现出合理的近似效果 [16, 页40],我们对这一方法进行了改进，以应对问题更高的复杂度。每当一台机器 m空闲时，就根据 Π_m (2) 对作业进行排序。我们检查排序列表中的第一个待处理作业 j_1 是否能在另一台运行中的机器上更早完成 m^*，即 $\exists\ m^*$: （$r_{m^*} + p_{j_1 m^*} < p_{j_1 m}$）。如果不存在这样的机器 m^*，则将作业 j_1 分配给机器 m。否则，就对下一个尚未处理的作业重复这一过程，直到某个作业被分配。如果迭代结束时仍未分配作业，则停用空闲机器 m 。

IV. 网络架构

我们的模型处理每个新出现的状态 $s \in S$ ，其中包含 J 个待处理作业和 M 台活动机器，以生成调度动作。排序后的状态数据是神经网络的输入。网络的输出 $\hat{} y \in \mathbb{R}^{J+1}$ 是一个关于可行动作的概率分布： J 用于将每个作业分配给可用机器 m，以及一个用于停用该机器的额外动作。该架构由三个主要组件构成，其中前两个充分利用了并行化：

1) **序列嵌入**：通过长短期记忆网络将与每个作业相关的机器数据作为时间序列进行处理和嵌入，为每个作业生成固定大小的表示。2)**Transformer编码器**：通过整合作业间的上下文信息，增强作业表示。



(a) 输入数据： $X_{\text{res}}^{(j)}$和 $x_{\text{urg}}^{(j)}$

(b) 将 $X_{\text{res}}^{(j)}$序列嵌入为固定大小向量

图 2：将机器数据嵌入为静态维度的作业表示

3) 动作指针解码器：利用作业和机器的综合上下文信息，生成所有可行动作的概率分布。

采用以带泄漏的 ReLU 作为激活函数的稠密前馈层（FF） ϕ_F ，并使用激活 $\alpha = 0.1$进行进一步增强；这些层同样可以并行计算：

$$FF(x) := W_{\text{out}} * \phi_F(W_{\text{in}} * x + b_{\text{in}}) + b_{\text{out}}. \tag{5}$$

A. 数据预处理

如第 III-A 节所述，每个状态 $s \in S$ 都具有可变的输入大小，包括 M 维度为 3 的运行中的机器向量和 J 待处理作业向量，其特征维度 $M + 2$ 还会随机器数量而变化，从而使神经网络处理的输入标准化更加复杂。为此，我们定义向量

$$x_m^{(j)} := [p_{jm}, r_m^{(s)}, \epsilon_m^{(s)}, \omega_m]^T \in \mathbb{R}^4 \tag{6}$$

表示机器 m相对于作业 j 的所有信息，随后将每个作业的数据分解为两个不同的组成部分：

$$X_{\text{res}}^{(j)} := [x_1^{(j)}, \ldots, x_M^{(j)}]^T \in \mathbb{R}^{M\times 4} \tag{7}$$

$$x_{\text{urg}}^{(j)} := [e_j^{(s)}, w_j]^T \in \mathbb{R}^2. \tag{8}$$

对于每个作业，第一个分量会将与该作业相关的每台机器的数据嵌入其表示中。具体做法是将机器数据组织成时间序列矩阵，以表示该作业不断出现的资源。第二个分量的大小固定，用于捕捉处理每个作业的紧迫性。请注意， J 和 M 分别指待处理的作业和正在运行的机器，因此会在调度过程中不断变化。为确保输入尺度一致，我们将权重除以允许的最大值进行归一化，而所有与时间相关的数据则除以每个状态中的最大时间值进行归一化。有关此数据转换过程的可视化示例，请参见图2a。这种双部分方法使我们的模型能够生成固定大小的作业向量，其中包含处理后的机器信息，如图2 所示，并在下一节IV-B中详细说明。对于所有作业，神经网络

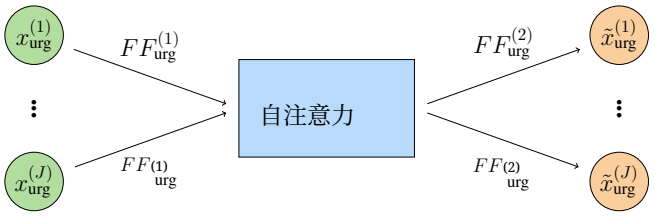


图3：作业紧迫性的情境化

因此接收资源张量和紧迫性矩阵作为归一化输入：

$$\text{Resource Input}: \left[X_{\text{res}}^{(1)}, \ldots, X_{\text{res}}^{(J)}\right]^T \in [0,1]^{J\times M\times 4} \tag{9}$$

$$\text{Urgency Input}: \left[x_{\text{urg}}^{(1)}, \ldots, x_{\text{urg}}^{(J)}\right]^T \in [0,1]^{J\times 2}. \tag{10}$$

B. 序列嵌入

网络的输入由式（9）和（10）中定义的两组作业表示组成。对于每个作业 j，其由式（ $X_{\text{res}}^{(j)} = x_1^{(j)}$、.....、 $x_M^{(j)}$ ）给出的机器资源变化时间序列取决于机器数量 9。 M同时，每个 $X_{\text{res}}^{(j)}$都经过双向 LSTM 处理，嵌入为固定大小的表示 $x_{\text{res}}^{(j)}$，如图 2b所示。 $x_{\text{res}}^{(j)}$ 随后通过一个前馈层进一步处理为 $\tilde{x}_{\text{res}}^{(j)}$ 。

同时，式（ $x_{\text{urg}}^{(j)}$ ）中的输入 10经过处理，以根据其他作业的情况对每个作业的紧迫性进行上下文化表示，如图 3 所示。具体而言，先将这些向量输入前馈层，再应用自注意力机制生成上下文紧迫性表示，最后通过另一个前馈层进一步处理输出，得到 $\tilde{x}_{\text{urg}}^{(j)}$。

每个作业的最终嵌入表示 $x^{(j)}$ emb 通过拼接 $\tilde{x}_{\text{res}}^{(j)}$ 和 $\tilde{x}_{\text{urg}}^{(j)}$ ，并将其输入额外的前馈层以进一步捕捉上下文信息来获得。由此得到的所有嵌入作业表示序列为：

$$X_{\text{emb}} := [x_{\text{emb}}^{(1)}, \ldots, x_{\text{emb}}^{(J)}]^T \in \mathbb{R}^{J\times 16}. \tag{11}$$

C. Transformer编码器

每个作业最初都会嵌入有关其紧迫性和特定机器数据的上下文。然而，它缺少关于活动机器上其他作业加工时间的知

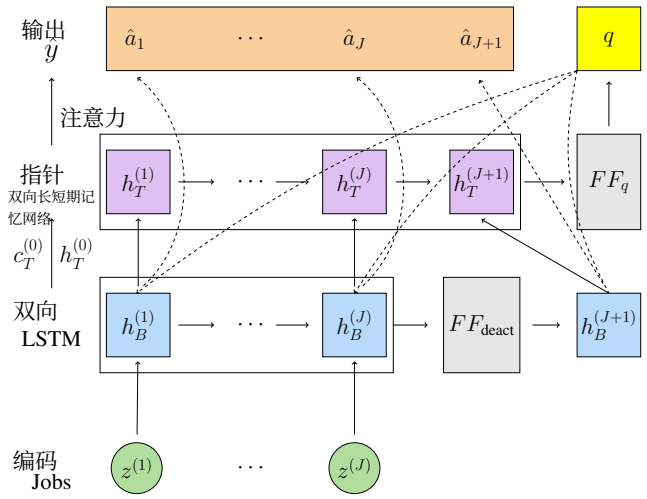


图 4：改进后的动作指针解码器会输出一个针对所有可行操作的注意力分布：将相应的待处理作业分配给当前空闲的机器 ($\hat{a}_1$、...、$\hat{a}_J$)，或停用该机器 ($\hat{a}_{J+1}$)。

识。为整合这些信息，我们将 Transformer 编码器应用于 (11)，如 [17] 所述。该编码器利用自注意力机制，有效扩展了 $x^{(j)}$ 嵌入的上下文，使其涵盖其他作业如何与相同机器交互。该编码器的输出 $z^{(j)} \in \mathbb{R}^{16}$，在给定调度问题当前状态的完整上下文中，对作业 j 进行了全面编码。

D. 动作指针解码器

我们的网络输出旨在为每个 J 待处理作业各包含一个条目，并额外包含一个用于停用机器选项的条目，因此需要支持可变维度的输出。我们采用动作指针解码器 [18] 来有效实现这种灵活性。图4展示了解码器的完整架构。

首先，将编码后的作业表示 $z^{(1)}, \ldots, z^{(J)}$ 输入双向 LSTM（底层 LSTM）进行处理，生成一系列隐藏状态 $h_B^{(1)}, \ldots, h_B^{(J)} \in \mathbb{R}^{64}$，每个隐藏状态都对应于将相应作业分配给当前空闲机器这一潜在动作。为了创建停用动作 $h_B^{(J+1)}$ 的上下文感知表示，我们将最终隐藏状态 $h_B^{(J)}$ 输入一个前馈层。随后，将底层 LSTM 的最终隐藏状态和细胞状态输入另一个前馈层，以初始化第二个双向 LSTM（顶层 LSTM）的记忆状态，从而构成我们的动作指针。顶层 LSTM 按序处理来自底层 LSTM 的隐藏状态，这些隐藏状态也用作键。再将其最终隐藏状态 $h_T^{(J+1)}$ 输入另一个前馈层，以生成查询 $q \in \mathbb{R}^{128}$。然后，神经网络的最终输出为注意力分布：通过应用 [18] 中描述的注意力机制，并使用 softmax 激活函数对各个 $J+1$ 动作的分数进行归一化，计算得到该分布：

$$\hat{y} = [\hat{a}_1, \ldots, \hat{a}_{J+1}]^T \in [0,1]^{J+1}.$$

表 I：网络参数和维度

层	参数 (共 = 117.292)	输出维度
双向 LSTM	2688	$2 \cdot 16$
FF_{res}	800	16
$FF_{\text{urg}}^{(1)}$	32	4
自注意力	308	4
$FF_{\text{urg}}^{(2)}$	40	4
拼接输入	0	20
FF_{emb}	608	16
序列嵌入	总计 = 4476	输入：$J \times M \times 4 + J \times 2$ 输出：$J \times 16$
自注意力编码器	8592	16
残差连接1	0	16
FF_{enc}	544	16
残差连接2	0	16
Transformer编码器	总计 = 9136	输入：$J \times 16$ 输出：$J \times 16$
解码器底层双向 LSTM	12544	$J \times (2 \cdot 32)$
FF_{deact}	8320	64
拼接 $h_B^{(J+1)}$ 至序列	0	$(J+1) \times 64$
$FF_{\overrightarrow{h}}$	2112	32
$FF_{\overleftarrow{h}}$	2112	32
$FF_{\overrightarrow{c}}$	2112	32
$FF_{\overleftarrow{c}}$	2112	32
动作指针：解码器顶层双向 LSTM	24832	$(J+1) \times (2 \cdot 32)$
FF_q	24832	128
动作指针：交叉注意力	24704	$J+1$
动作指针	总计 = 103.680	输入：$J \times 16$ 输出：$J+1$

所有网络参数和维度的概览见表I。

V. 实证评估

除了用于训练的 100.000 个调度问题外，我们还模拟了 10.000 个形式为（1）的问题，并按照第 III-D节中的约束进行验证；此外还模拟了另外 10.000 个问题用于测试。对于每个此类问题，我们针对待处理作业数为 3 至 8、活动机器数为 2 至 4 的每种组合，随机选取了一个状态。为了评估我们的神经网络，我们采用了度量指标 ΔQ（s，$\mu_\vartheta(s)$），其定义见（4），用于衡量采用策略 μ_ϑ 所产生的额外成本；该策略由我们的网络及其学习得到的参数 ϑ导出，并与最优策略 μ^*进行比较。表 IIa中的结果表明，在所有设置下，我们的模型始终将额外相对成本控制在平均 1% 以下，凸显了其稳健的泛化能力。值得注意的是，模型在验证和测试期间的性能甚至往往更好。这是因为训练期间均衡地选取了状态，而测试期间的随机选取很可能包含了更多这样的状态：我们的初始排序置换 Π_m 已能较好地近似 μ^*，从而简化了调度优化。

表 II：调度 8 个作业和 4 台机器时，相对于最优成本的增幅

（a）调度方案中的每个状态决策

作业数	机器数	训练 ΔQ [%]	验证 ΔQ [%]	Test ΔQ [%]
3	2	0.45	0.14	0.15
	3	0.33	0.26	0.23
	4	0.36	0.46	0.45
4	2	0.63	0.16	0.13
	3	0.51	0.15	0.21
	4	0.24	0.31	0.32
5	2	0.58	0.17	0.17
	3	0.60	0.22	0.24
	4	0.53	0.31	0.31
6	2	0.28	0.20	0.21
	3	0.47	0.28	0.32
	4	0.45	0.32	0.35
7	2	0.31	0.20	0.19
	3	0.45	0.40	0.43
	4	0.45	0.41	0.40
8	2	0.34	0.32	0.32
	3	0.30	0.60	0.57
	4	0.45	0.63	0.63

（b）对于总调度方案

方法	策略	ΔV_μ [%]
神经网络早期最优截断	$\mu_{\vartheta,\ \text{最优}}$	2.51
神经网络	μ_ϑ	4.47
派工规则早期最优截断	$\mu_{\text{rule},\ \text{opt}}$	34.46
派工规则	μ_{rule}	45.33

表 III：使用派工规则相对于使用网络在不同作业数量（ [%] ）和机器数量（J）下调度成本的相对增幅。数值通过（13）定义，以百分比表示，代表我们网络的性能优势。

J \ M	2	3	4	5	6	8	10	12
8	19.57	27.47	31.17	32.09	29.79	18.13	08.32	01.47
9	21.11	25.38	32.39	33.79	32.44	20.22	09.38	-01.44
10	20.09	25.62	32.26	35.43	34.43	23.02	11.09	00.29
11	18.02	26.86	33.03	36.21	35.88	25.22	11.65	02.04
12	18.83	26.33	32.43	36.53	37.09	26.36	12.61	02.90
15	17.98	22.07	30.30	36.99	38.27	29.24	15.17	03.48
20	15.14	21.29	26.90	34.23	38.14	31.53	17.09	04.87
30	13.36	16.73	21.20	27.74	32.62	31.72	18.04	04.39
50	10.65	12.52	14.94	18.56	23.40	26.13	16.36	04.13
75	09.91	10.09	10.56	12.69	14.97	18.02	12.68	03.72
100	09.94	08.96	07.97	07.59	08.45	11.15	09.10	03.27

A. 8 个作业和 *4* 台机器的监督性能

我们通过针对一组新的 10,000 个调度问题生成完整调度方案，评估了神经网络的性能；每个问题包含 8 个作业和 4 台机器。我们将采用网络策略 μ_ϑ 产生的成本与最优成本，以及第 III-E节中启发式派工规则产生的成本进行比较，后者记为 μ_{rule}。为聚焦于更复杂的决策场景，我们为每种策略定义了第二个版本，记为在其名称后添加 *opt*；当只剩两个作业，或仅有一台运行中的机器且待处理作业不超过 8 个时，该版本会直接计算剩余的最优调度方案（这对下一节 V-B很重要）。对于由初始状态 s表示的每个调度问题，我们测量了调度成本相对于.....的相对增幅

相对于最优成本 $V^*(s)$，使用以下度量指标：

$$\Delta V_\mu(s) := \frac{V_\mu(s) - V^*(s)}{V^*(s)}. \tag{12}$$

结果汇总于表IIb中，验证了我们的网络能够紧密逼近最优调度成本。此外，该网络的表现显著优于派工规则，其产生的额外相对成本不到启发式算法所产生成本的 10%。

B. 更高阶调度问题的性能

我们评估了与派工规则相比，我们的神经网络在更大、更复杂的调度问题上的泛化能力。为此，我们针对作业的各种组合，额外模拟了 10.000 个具有第 III-D 节所述约束的调度问题

数量 J 和机器数量 M，如表III所示。该表中的条目表示商

$$\frac{V_{\mu_{\text{rule,opt}}}(s) - V_{\mu_{\vartheta,\text{opt}}}(s)}{V_{\mu_{\vartheta,\text{opt}}}(s)} \tag{13}$$

以百分比表示，说明使用高级派工规则 $\mu_{\text{rule, opt}}$ 相比于我们的神经网络 $\mu_{\vartheta,\text{ opt}}$ 及其训练参数 ϑ所产生的额外相对调度成本。结果表明，在最多包含 100 个作业和 10 台机器的所有问题场景中，我们的神经网络性能均显著优于启发式方法；在这一范围内经过测试的设置中，该派工规则导致平均调度成本高出 22.22%。当机器数量接近 12 台时，两者的性能趋于持平。有趣的是，最大的优势并未出现在包含 8 个作业和 4 台机器的训练场景中，而是出现在以下范围：8 至 20 个作业和 4 至 6 台机器，以及 15 至 30 个作业和 5 至 8 台机器。在最多包含 30 个作业且机器数量介于 3 至 8 台的设置中，除两种配置外，该派工规则生成的调度方案平均成本均比我们的神经网络生成的方案高出 20% 以上，且通常超过 30%。

VI. 局限性

尽管我们的网络仅针对限制为 8 个作业和 4 台机器的调度问题进行训练，但它对规模大得多的环境展现出了出色的泛化能力，能够处理作业数量超过其训练数量十倍的配置。借助更多的计算资源，并引入图形处理器以利用网络架构所带来的固有并行化能力，可以扩大训练集，从而进一步提升其泛化能力，尤其是适应更多机器的情况。

尽管我们在第 III-D 节中预先设定了参数限制，但训练后的网络可以直接应用于超出这些范围的调度问题。将所有与时间相关的数值按某一因子进行缩放，既不会改变策略之间的成本关系，也不会影响网络对状态—动作的推荐。由于我们在第 IV-A 节中以状态为单位，将输入动态归一化到 0 到 1 之间，因此模型对绝对尺度保持不变，能够解释训练期间未使用的时间比率。

虽然我们的方法只针对一个特定且复杂的问题（1），但通过构造边界情况，可以轻松地将其调整为适用于更简单的环境。例如，将相应的权重设为零，即可忽略机器或作业延迟；将处理时间设为成比例，则可对同类机器调度进行建模。只需对网络架构进行极少改动，即可纳入其他约束。未来研究可通过加入各种约束来扩展我们的框架，使当前的特定问题成为更通用解决方案的一个边界情况，从而适用于范围广泛的问题设置。由于采用了监督学习方案，针对不同目标函数重新训练模型的效率较高；相比之下，标准的基于规则的算法通常需要针对每个问题定制解决方案；而依赖深度强化学习（DRL）的方法则收敛速度明显更慢。

尽管目前仅限于确定性环境，但通过引入期望处理时间，该方法论可以适用于随机环境；对于离散分布，则可以使用单独的向量表示潜在的处理时间及其对应的概率。

VII. 结论

我们开发的网络架构展现了在复杂问题环境中进行无关机器调度所需的灵活性。它成功处理了输入和输出规模以及输入特征维度的固有变化。我们的网络在调度 8 个作业和 4 台机器时表现出色。与最优策略相比，它在每次状态决策中产生的额外成本不到 1%，在完整调度方案中的额外成本仅为 2.51%，远低于先进派工方法产生的 34.46%。尽管我们的网络仅在这些小规模调度环境中接受训练，但它对规模大得多的问题展现出了出色的泛化能力，可处理多达 100 个作业和 10 台机器，并始终优于派工规则，在大多数测试配置中实现至少 10% 的成本降低。除两种配置外，在机器数量为 3 至 8 台、作业数量最多为 30 个的所有配置中，派工规则生成的调度方案成本都比我们的网络高至少 20%，最高达 38.27%。

这些结果凸显了我们的方法在无关并行机上应对确定性调度问题时，具备所需的灵活性、适应性和泛化能力。更多的计算资源将有助于在更多样化的数据集上高效训练，并纳入作业数和机器数更高的调度问题，从而进一步提升模型的泛化能力。此外，还可以利用该框架与深度强化学习的兼容性，在坚实基础上进一步提升模型性能。尽管本研究仅关注特定的问题约束，但我们架构的适应性，加上高效监督学习方法所支持的快速再训练，为未来研究探索其在不同运行条件和目标函数下的有效性提供了契机。

参考文献

[1] M. L. Pinedo，《调度：理论、算法与系统》。Springer 出版公司，2008年。[2] D. Hitzges，《无关机器调度的深度学习》，2025年3月。[在线]。可访问： https://github.com/DiegoHitzges/Deep-Learning-for-Unrelated-Machines-Scheduling [3] M. Durasevi´c 和 D. Jakobovi´c，《并行无关机器调度问题的启发式与元启发式方法：综述》，《人工智能评论》，第56卷，第4期，第3181–3289页，2023年。[4] L. Yang-Kuei 和 L. Chi-Wei，《带释放日期的无关并行机调度派工规则》，《国际先进制造技术杂志》，第67卷，第269–279页，2013年。[5] H. Zhou 、 Z. Li 和 X. Wu，《使用蚁群优化使无关并行机调度的总加权拖期最小化》，载于*2007*年 *IEEE* 自动化与物流国际会议。IEEE，2007年，第132–136页。

[6] M. Bou˘ska、P. ˘Scha、A. Nov´ak 和 Z. Hanz´alek，《用于最小化总拖期的单机问题的深度学习驱动的调度算法》，《欧洲运筹学杂志》，第308卷，第3期，第990–1006页，2023年。[7] B. Skadel、B. ´Snie˙zy´nski 等，《机器调度的在线监督学习方法》，*Schedae Informaticae*，2016年，第25卷，第165–176页，2017年。[8] V. Kumar、S. Bhambri 和 P. G. Shambharkar，《使用深度强化学习进行多资源管理和突发时间预测》，载于第八届计算、通信与信息技术进展国际会议（*CCIT*），2019年，第8页。[9] H. Mao、M. Alizadeh、I. Menache 和 S. Kandula，《基于深度强化学习的资源管理》，载于第*15*届 *ACM* 网络热点问题研讨会论文集，2016年，第50–56页。[10] P. Tassel、M. Gebser 和 K. Schekotihin，《作业车间调度的强化学习环境》，*arXiv* 预印本 *arXiv:2104 03760*，2021。。[11] B. M. Kayhan 和 G. Yildiz，《强化学习在机器调度问题中的应用：全面的文献综述》，《智能制造杂志》，第34卷，第3期，第905–929页，2023年。[12] G. ´Abrah´am、P. Auer、G. D´osa、T. Dulai 和 ´A. Werner-Stark，《一种受强化学习启发的过程优化算法》，《*Periodica Polytechnica* 土木工程》，第63卷，970，2019年4月，第961页。[13] Z. Zhang、L. Zheng、N. Li、W. Wang、S. Zhong 和 K. Hu，《使用强化学习最小化无关并行机调度中的平均加权拖期》，《计算机与运筹学》，第39卷，第7期，第1315–1324页，2012年。[14] L. Guo、Z. Zhuang、Z. Huang 和 W. Qin，《采用多阶段强化学习优化动态多目标非同类并行机调度》，载于*2020*年 *IEEE* 第*16*届自动化科学与工程国际会议（*CASE*）。IEEE，2020年，第1215–1219页。[15] M. Khadivi、T. Charter、M. Yaghoubi、M. Jalayer、M. Ahang、A. Sho-jaeinasab 和 H. Najjaran，《机器调度的深度强化学习：方法论、最新进展与未来方向》，*arXiv* 预印本 *arXiv:2310.03195*，2023年。[16] D. P. Williamson 和 D. B. Shmoys，《近似算法设计》。剑桥大学出版社，2010年。[17] A. Vaswani、N. Shazeer、N. Parmar、J. Uszkoreit、L. Jones、A. N. Gomez、Ł. Kaiser 和 I. Polosukhin，《注意力就是你所需要的一切》，《神经信息处理系统进展》，第30卷，2017年。[18] O. Vinyals、M. Fortunato 和 N. Jaitly，《指针网络》，《神经信息处理系统进展》，第28卷，2015年。