计算机科学与工程学院
School of computer science and engineering

Enhancing QoE in Collaborative Edge Systems With Feedback Diffusion Generative Scheduling

TMC'25

Changfu Xu [1], Jianxiong Guo [2], Yuzhu Liang [2], Haodong Zou [3], Jiandian Zeng [2], Haipeng Dai [4], Weijia Jia [2], Jiannong Cao [5], and Tian Wang [2]

Reporter: Yuxiang Lu

[1] Jiangxi University of Finance and Economics

[2] Institute of Artificial Intelligence and Future Networks, Beijing Normal University

[3] Anhui University

[4] Nanjing University

[5] Hong Kong Polytechnic University

2025.12.15

作者团队

Changfu Xu

助理教授，
江西财经大学

博士，
北师香港浸会大学

主要研究方向：Mobile Edge Computing, DRL, AIGC, Medical Robot, and Social Network.

TMC’24 - Dynamic Parallel Multi-Server Selection and Allocation in Collaborative Edge Computing (CCF A)

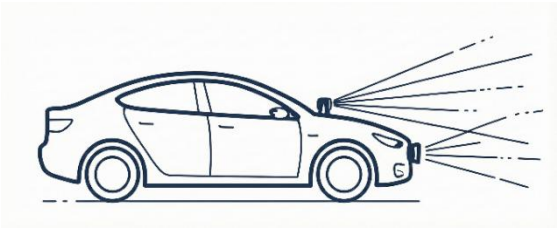
IWQoS’24 - Incorporating Startup Delay into Collaborative Edge Computing for Superior Task Efficiency (CCF B, Best paper runner-up)

ICDCS’24 - Enhancing AI-Generated Content Efficiency Through Adaptive Multi-Edge Collaboration (CCF B)

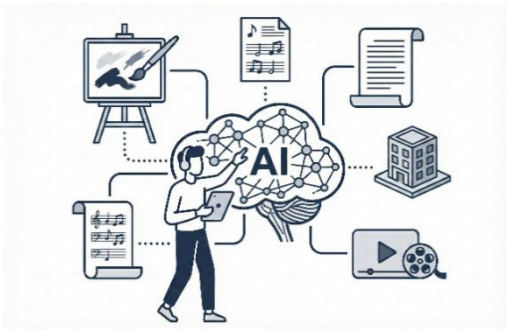
目录

目录

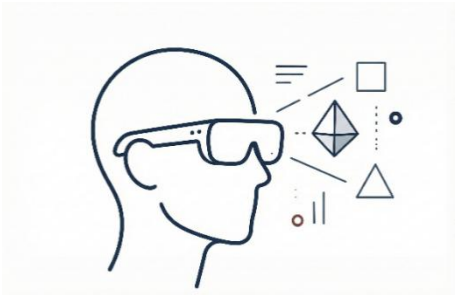
背景介绍



自动驾驶
（AD）



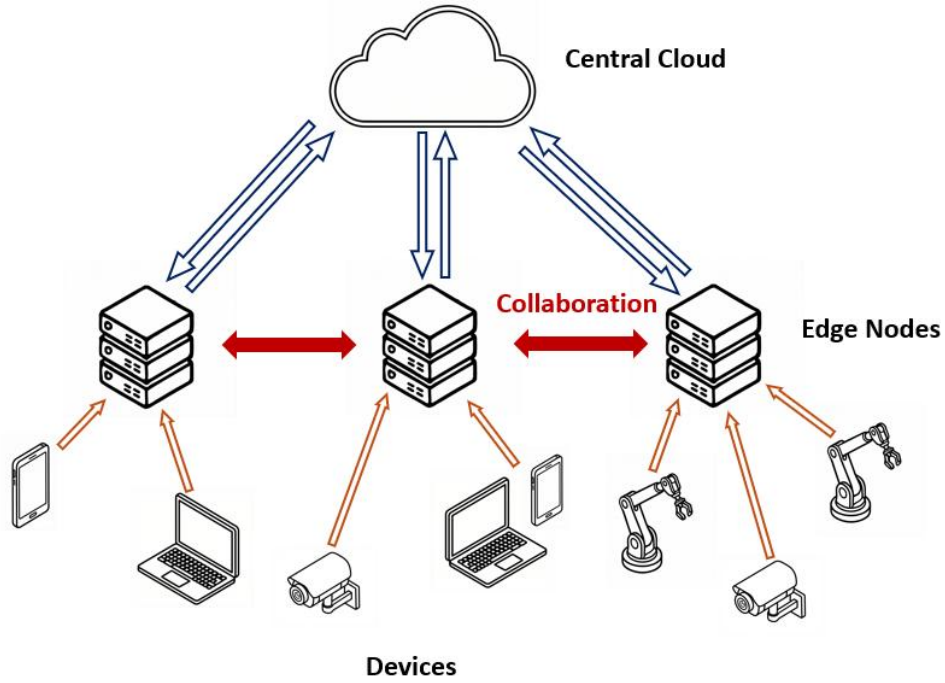
人工智能
生成内容
（AIGC）



增强现实
（AR）

服务质量QoE：
要求极低的延迟

现代物联网（IoT）应用



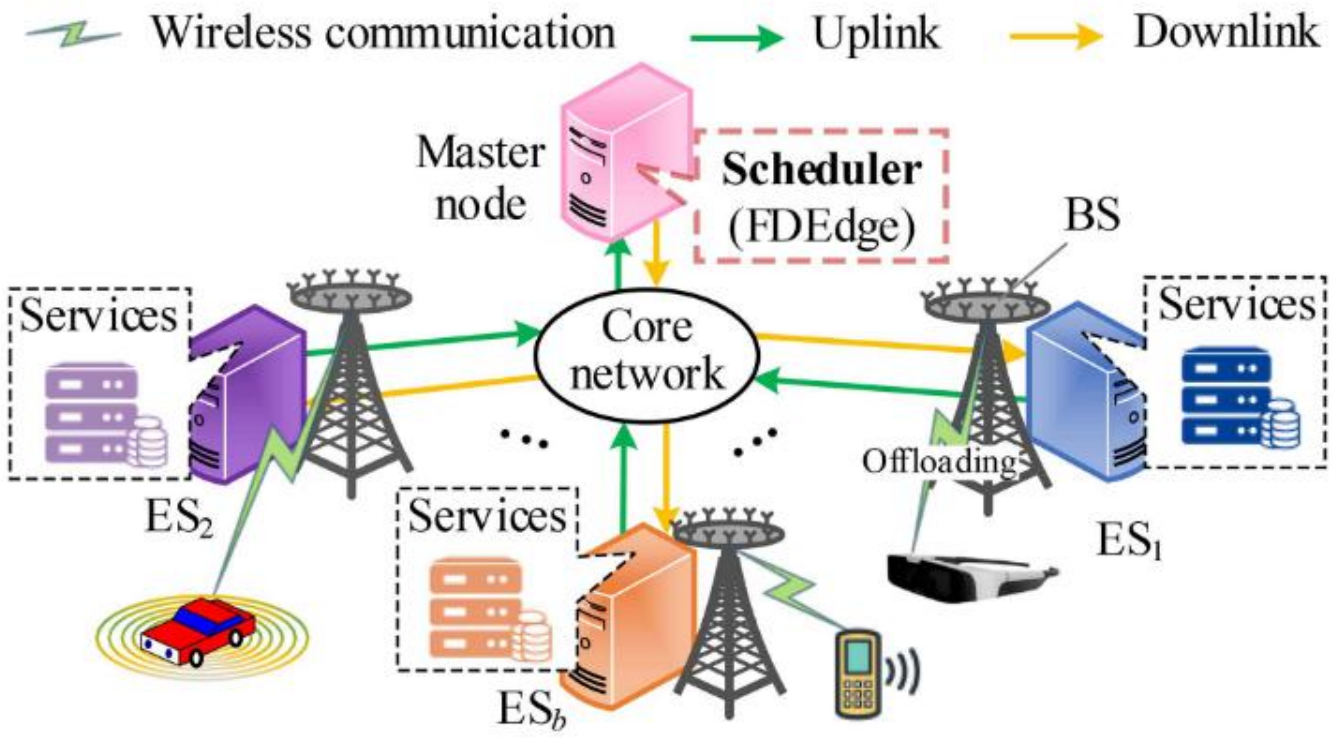
边缘计算

协同边缘计算

背景介绍

问题背景

- 一个用于处理计算密集型现代应用（如AD、AR和AIGC）的协同边缘系统。
- 该系统由多个基站（BSs）组成，采用集中调度框架。每个BS都配备有一个对应的边缘服务器（ES），并且拥有不同的计算能力。
- 每个ES都部署了多个服务来处理每个任务。



背景介绍

问题建模

建模的关键问题是对建模成为处理器数非线性规划（INLP）问题进行处理

建模的关键问题是对建模成为处理器数非线性规划（INLP）问题进行处理

$$\min_{\zeta} \lim_{|\mathcal{T}|\to\infty} \frac{1}{|\mathcal{T}|} \sum_{t\in\mathcal{T}} \sum_{n\in\mathbb{N}_t} \sum_{b\in\mathcal{B}} T^{serv}_{t,n,b}$$

$$s.t \quad \sum_{b\in\mathcal{B}} \zeta_{t,n,b} = 1, \forall n \in \mathbb{N}_t, t \in \mathcal{T}$$

$$\zeta_{t,n,b} \in \{0,1\}, \forall n \in \mathbb{N}_t, t \in \mathcal{T}, b \in \mathcal{B}$$

ζ 代表 t 时刻的任务 n 在每个ES上的状态

$T^{serv}_{t,n,b}$ 是服务时延，包含传输时延、执行时延和等待时延

$$T^{\text{serv}}_{t,n,b} = \zeta_{t,n,b} \cdot \left(\frac{d_n}{v_{t,n,b}} + \frac{\rho_n \cdot d_n}{f_b} + T^{\text{wait}}_{t,n,b} \right)$$

理论分析

任务视为货物
任务的计算负载视为货物的重量
ES的计算能力视为背包的容量
问题转化为一个**多背包（Multi-knapsack）问题。**

多背包问题是NP-hard的，因此建模出的问题从形式上，也是NP-hard的

背景介绍

问题建模

此时，问题就可以被建模成为一个整数非线性规划（INLP）问题

直接求解调度的优化问题

$$\zeta_{t,n,b} \in \{0,1\}, \forall n \in \mathbb{N}_t, t \in \mathcal{T}, b \in \mathcal{B}$$

ζ 代表 t 时刻的任务 n 在每个ES上的状态

$T_{t,n,b}^{serv}$ 是服务时延，包含传输时延、执行时延和等待时延

$$T_{t,n,b}^{\text{serv}} = \zeta_{t,n,b} \cdot \left(\frac{d_n}{v_{t,n,b}} + \frac{\rho_n \cdot d_n}{f_b} + T_{t,n,b}^{\text{wait}} \right)$$

理论分析

任务视为货物

使用一个**策略**对当前时刻的任务做出合理的即时决策

多背包问题是NP-hard的，因此建模出的问题从形式上，也是NP-hard的

目录

相关工作与动机

启发式算法

启发式算法本质上是基于人工设计的优化规则的迭代优化或贪婪搜索机制，它们依赖于明确的数学模型和约束条件，通过多轮计算或规则匹配来寻找问题的可行解。

代表方法：贪婪算法 (Greedy) 、博弈论 (Game Theory/Nash Equilibrium) 、Lyapunov 优化

由于缺乏对动态环境的全局感知，此类算法往往难以跳出局部最优，并且难以适应环境的动态变化。

启发式算法 → **DRL算法**

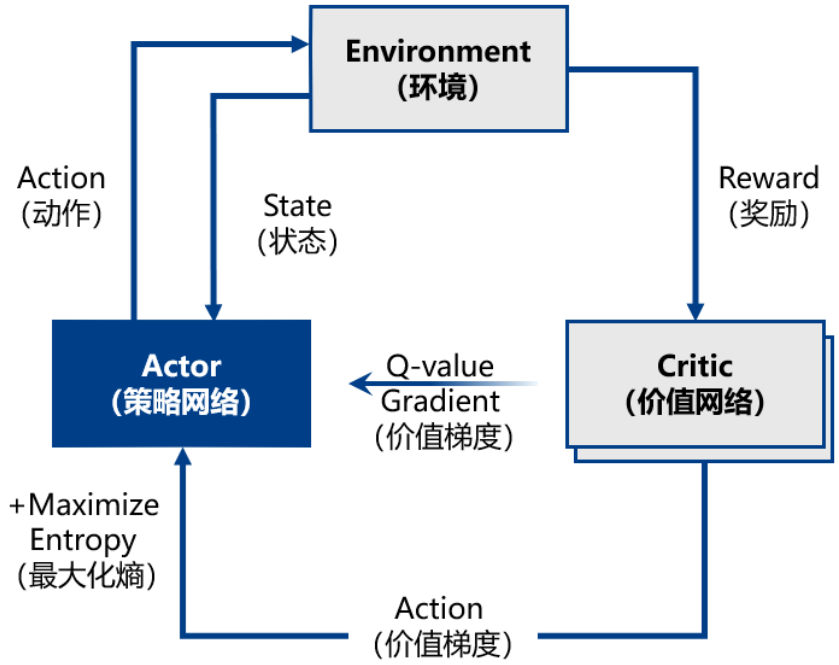
相关工作与动机

深度强化学习（DRL）

DRL 方法本质上是基于马尔可夫决策过程 (MDP) 的试错学习 (Trial-and-Error) 机制。智能体不依赖预设规则，而是通过与环境的持续交互收集“状态-动作-奖励”样本，不断训练更新参数，能够实现长期累积期望，并应对动态变化的环境。

代表方法：Deep Q-Network (DQN) 、Soft Actor-Critic (SAC) 、LDQN (LSTM + DQN)

基于SAC的算法



1. Actor：观察环境状态，输出包含探索性的动作概率分布
2. Environment：执行动作，返回奖励和新状态
3. Critic：评估某个状态下执行某个动作的好坏

* 最大化熵：SAC的创新核心，除了最大化奖励之外，还鼓励Actor的动作分布尽可能多样化，增强探索。

相关工作与动机

深度强化学习（DRL）

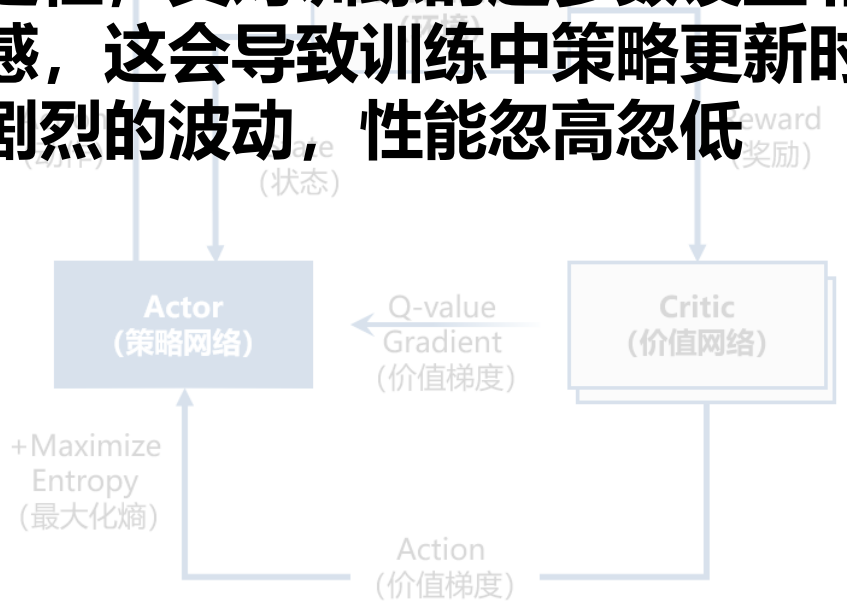
DRL 方法本质上是基于马尔可夫决策过程 (MDP) 的试错学习 (Trial-and-Error) 机制。智能体不依赖预设规则，而是通过与环境的持续交互收集“状态-动作-奖励”样本，利用反向传播更新神经网络参数，不断训练，实现长期累积期望

代表方法：Deep Q-Network (DQN)、Soft Actor-Critic (SAC)、LDQN (LSTM + DQN)

DRL算法存在的问题

1. **训练需要大量的高质量样本交互才能收敛，样本的效率低**
2. **训练过程与算法性能具有很强的不稳定性，其对训练的超参数设置非常敏感，这会导致训练中策略更新时存在剧烈的波动，性能忽高忽低**

基于SAC的算法



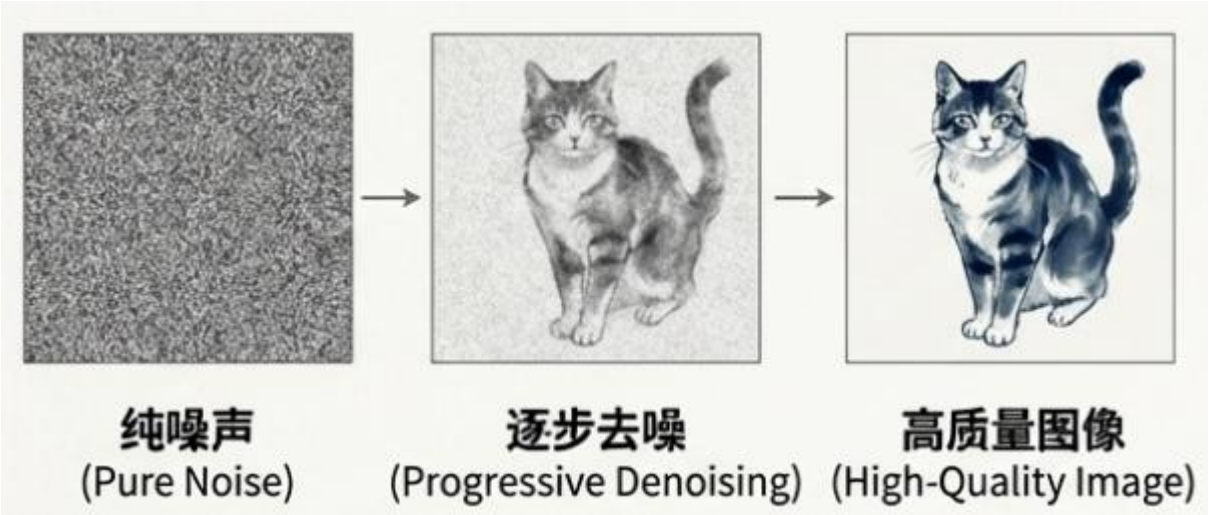
1. Actor：观察环境状态，输出包含探索性的动作概率分布
2. Environment：执行动作，返回奖励和新状态
3. Critic：评估某个状态下执行某个动作的好坏
4. 最大化熵：SAC的创新核心，除了最大化奖励之外，还鼓励Actor的动作分布尽可能多样化，增强探索。

相关工作与动机

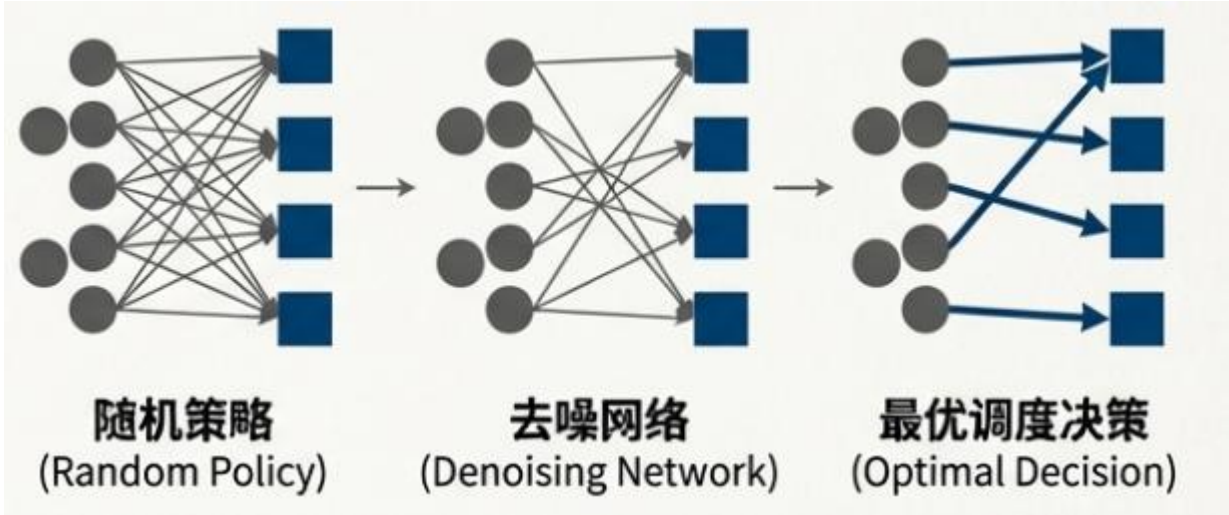
引入扩散模型来生成动作

发表在TMC24年上的文章“Diffusion-based reinforcement learning for edge-enabled AI-generated content services”所提出的算法D2SAC将扩散模型与SAC框架的结合解决边缘任务调度问题。思路是利用扩散模型代替原有SAC框架中的Actor并进行训练。

扩散模型：图像生成



去噪网络：任务调度



相关工作与动机

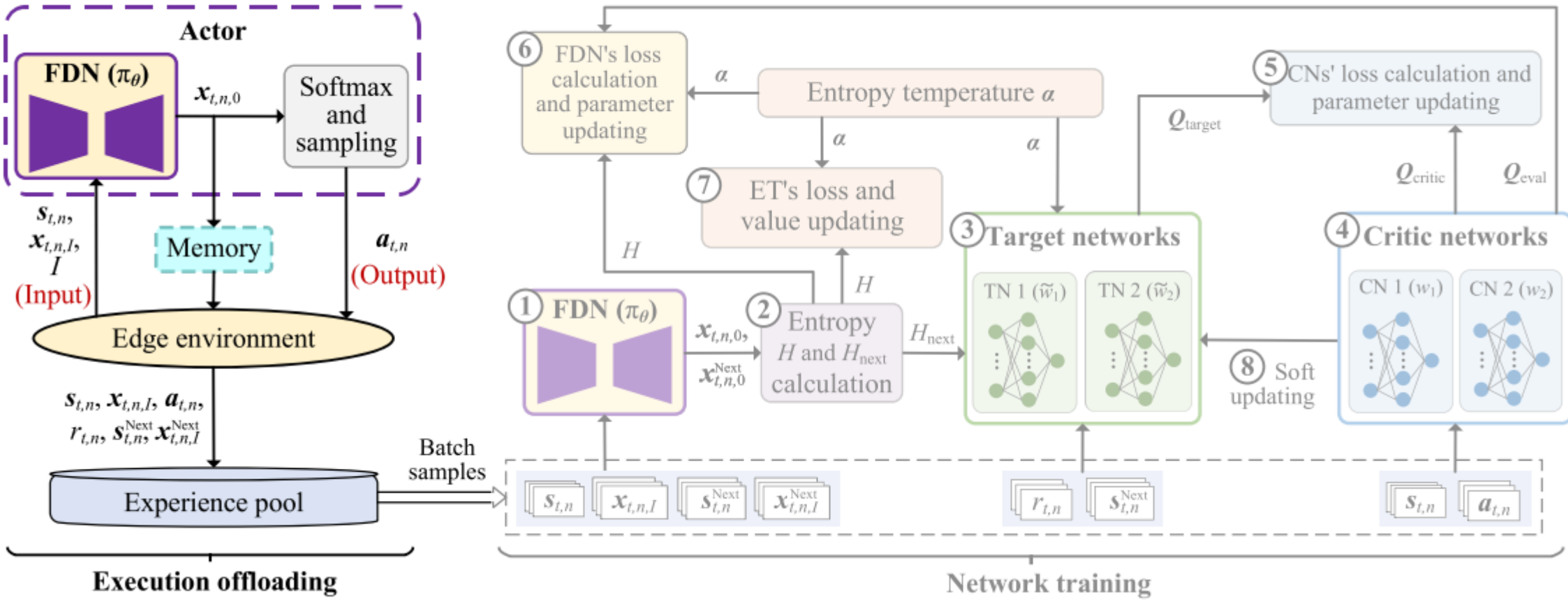
传统的扩散模型每次决策都是从完全随机的高斯噪声开始的，相当于“冷启动”，这导致算法的信息利用率很低且收敛很慢

作者在传统扩散模型的基础上提出了反馈扩散网络（FDN）模型。通过把历史动作的概率信息记录下来，作为先验知识反馈给模型的方式，在显著减少训练集需求的同时改善调度决策。

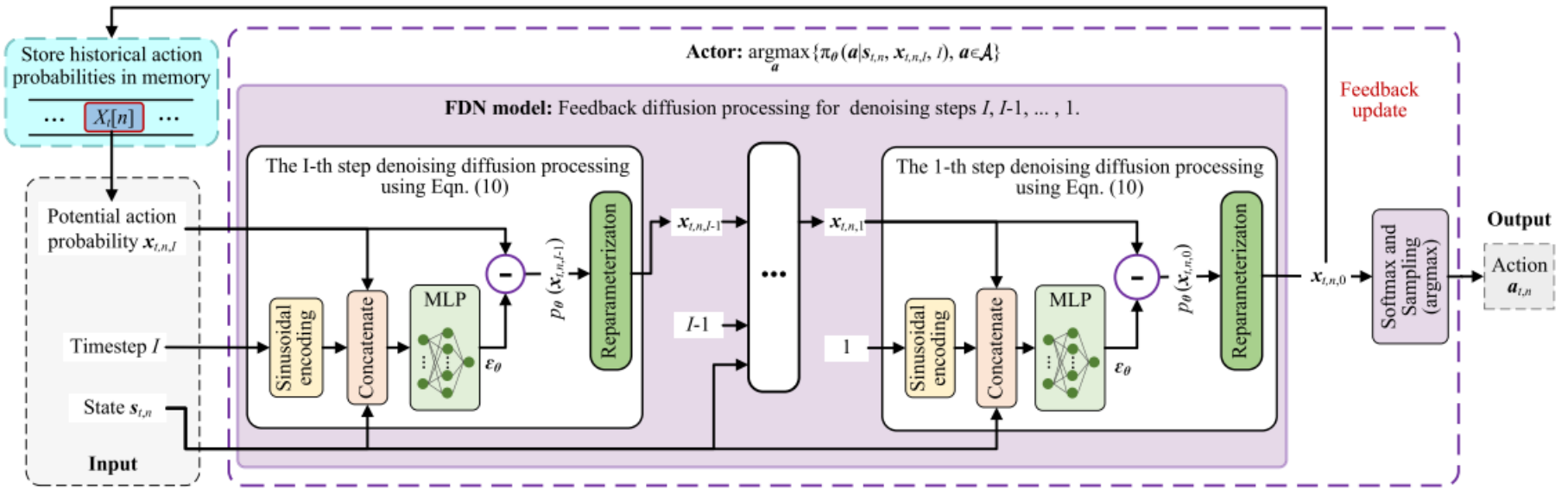
目录

算法设计

FDEdge算法框架



算法设计



Actor 的核心任务是根据当前的环境状态生成一个合理的动作概率分布，并选出最优的动作。

FDM模型的核心任务是根据输入的历史动作概率分布$x_{t,n,I}$，经过 I 步的去噪，最终得到最优的动作分布$x_{t,n,0}$。

核心公式

缩放因子

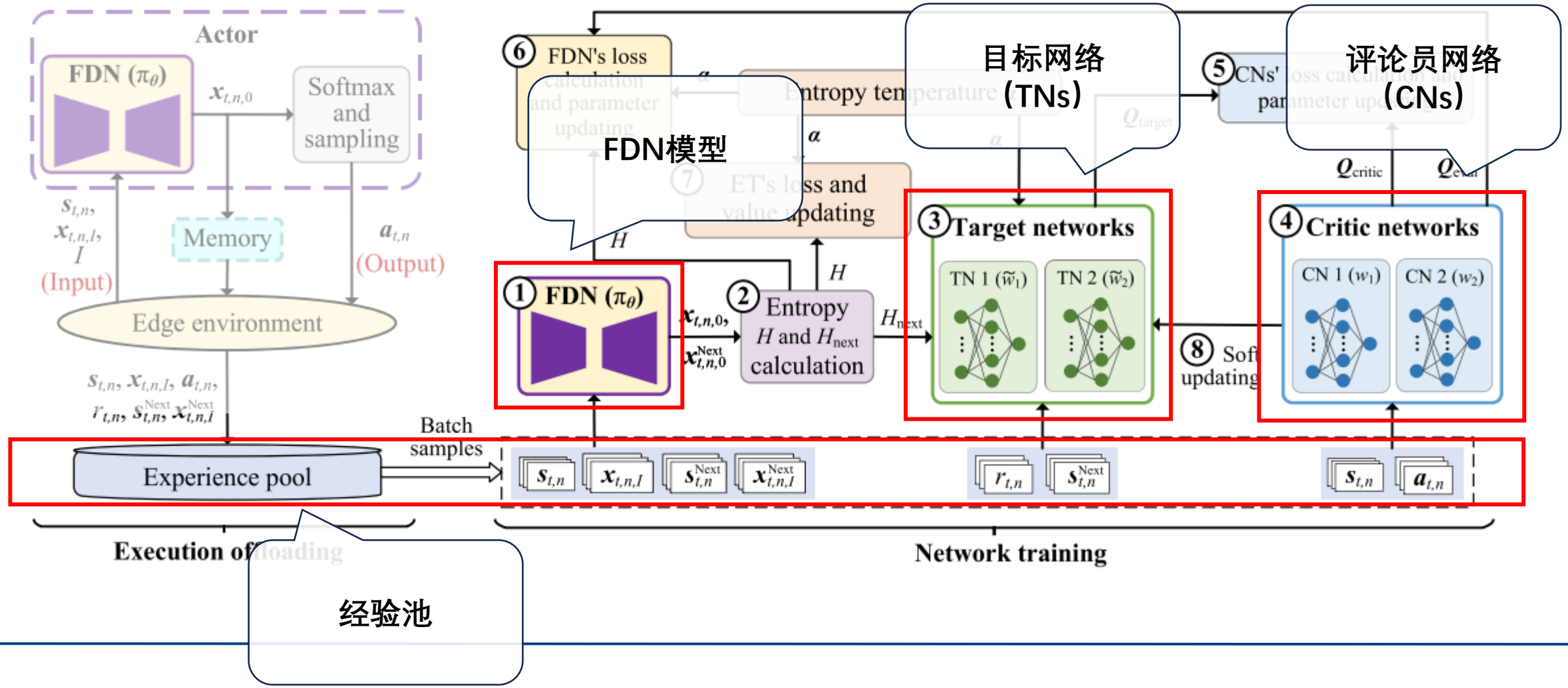
随机扰动

$$x_{t,n,i-1} = \frac{1}{\sqrt{\lambda_i}}\left(x_{t,n,i} - \frac{\beta_i}{\sqrt{1-\bar{\lambda}}}\epsilon_\theta(x_{t,n,i}, i, s_{t,n})\right) + \frac{\tilde{\beta}_i}{2}\cdot\epsilon$$

MLP预测出的噪声

算法设计

算法框架


Actor
FDN (π_θ)
$x_{t,n,0}$
Softmax and sampling
$s_{t,n}$, $x_{t,n,I}$, I (Input)
Memory
$a_{t,n}$ (Output)
Edge environment
$s_{t,n}$, $x_{t,n,I}$, $a_{t,n}$, $r_{t,n}$, $s^{Next}_{t,n}$, $x^{Next}_{t,n,I}$
Experience pool
Batch samples
Execution offloading
Network training
6 FDN's loss calculation and parameter updating
7 ET's loss and value updating
Entropy temperature
α
H
1 FDN (π_θ)
$x_{t,n,0}$, $x^{Next}_{t,n,0}$
2 Entropy H and H_{next} calculation
H_{next}
3 Target networks
TN 1 ($\widetilde{w}_1$)
TN 2 ($\widetilde{w}_2$)
Q_{target}
5 CNs' loss calculation and parameter updating
4 Critic networks
CN 1 (w_1)
CN 2 (w_2)
Q_{critic}
8 Soft updating
$s_{t,n}$, $x_{t,n,I}$, $s^{Next}_{t,n}$, $x^{Next}_{t,n,I}$
$r_{t,n}$, $s^{Next}_{t,n}$
$s_{t,n}$, $a_{t,n}$
FDN模型
目标网络（TNs）
评论员网络（CNs）
经验池

算法设计

Step 1

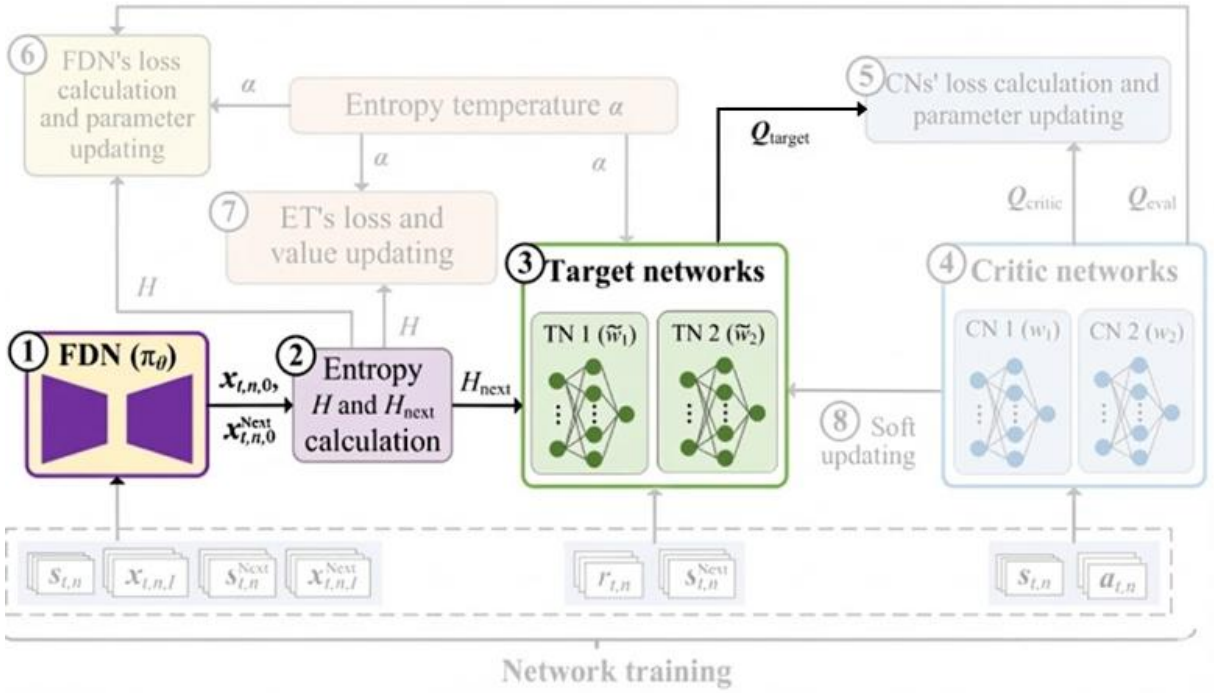
数据采集：
从经验池中随机取出经验，每条经验都是一个六元组，包含 $s_{t,n}$、$x_{t,n,I}$、$a_{t,n}$、$r_{t,n}$、$s_{t,n,I}^{Next}$、$x_{t,n,I}^{Next}$。

Step 2

② ③

计算目标Q值：
目的：为Critic网络的学习设定一个相对稳定的参考值
过程：计算当前和下一状态的熵，并根据下一状态 $s_{t,n,I}^{Next}$、下一历史动作概率 $x_{t,n,I}^{Next}$ 和奖励 $r_{t,n}$，通过目标网络（TN）计算出Actor动作对未来收益的稳定预估值 Q_{target}。
公式：

$$Q_{target} = r + \gamma * [E(Q_{target}^{next}) + \alpha * H_{next}]$$



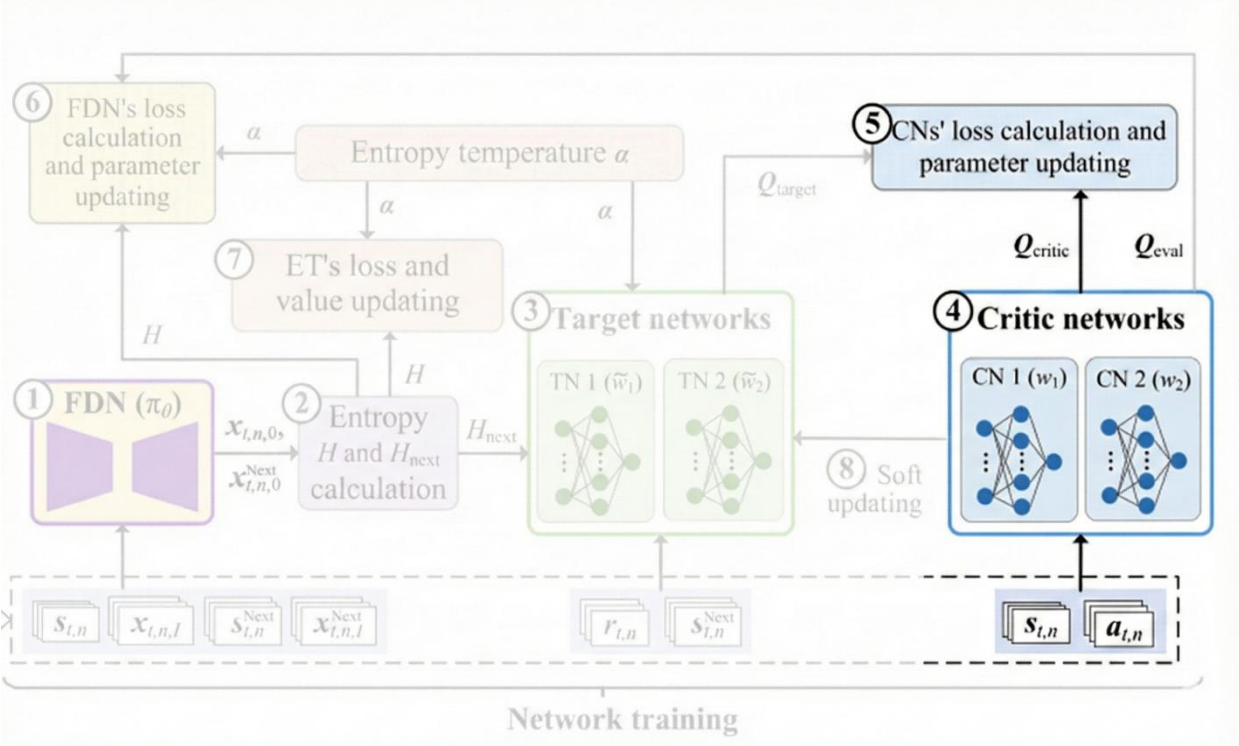
算法设计

Step 3 ④

计算当前Q值：
将当前状态 $s_{t,n}$ 和实际采取的动作 $a_{t,n}$ 输入到Critic网络（CNs），得到一个对当前动作的评价值 Q_{critic}。

Step 4 ⑤

计算损失并更新：
目的：通过梯度下降，更新Critic网络的参数，使其打分能力越来越接近“标准答案”。
损失函数：均方误差损失（MSE Loss）



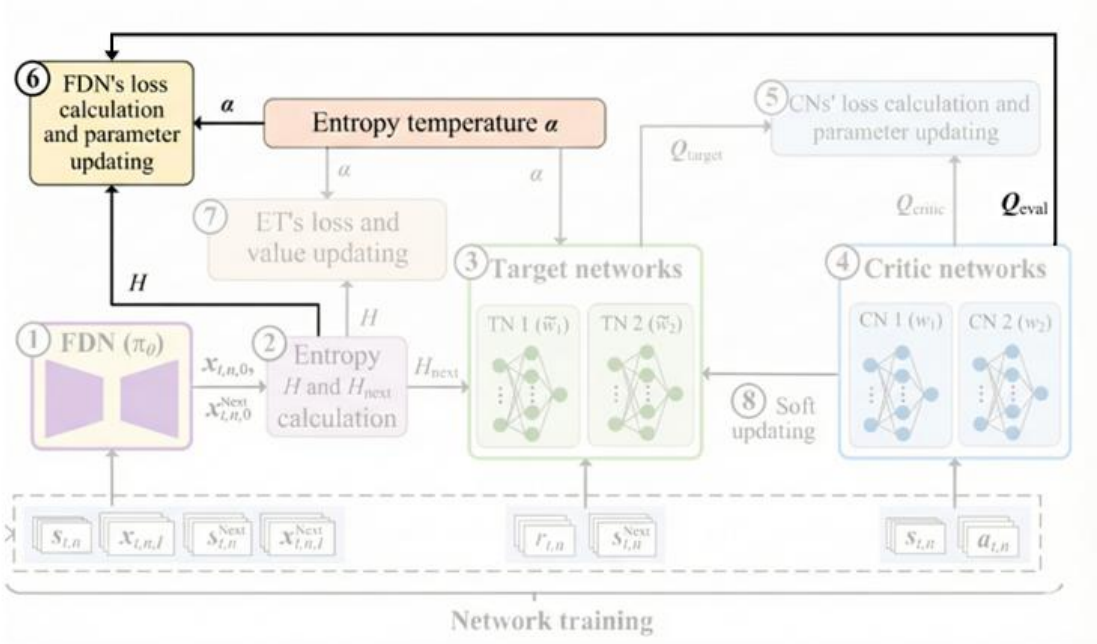
算法设计

Step 5 ⑥

更新Actor：
目的：Actor本质上的目标是生成能够让Critic网络打出尽可能高分的动作。
损失函数：Actor的损失函数的构造平衡了两个目标：

- **最大化Q值**：生成的动作要在Critic评估下的得分更高
- **最大化熵**：生成的动作分布需要保持随机性和多样性，防止过早收敛到局部最优

过程：根据这个损失函数，通过梯度下降更新FDN的参数。



算法设计

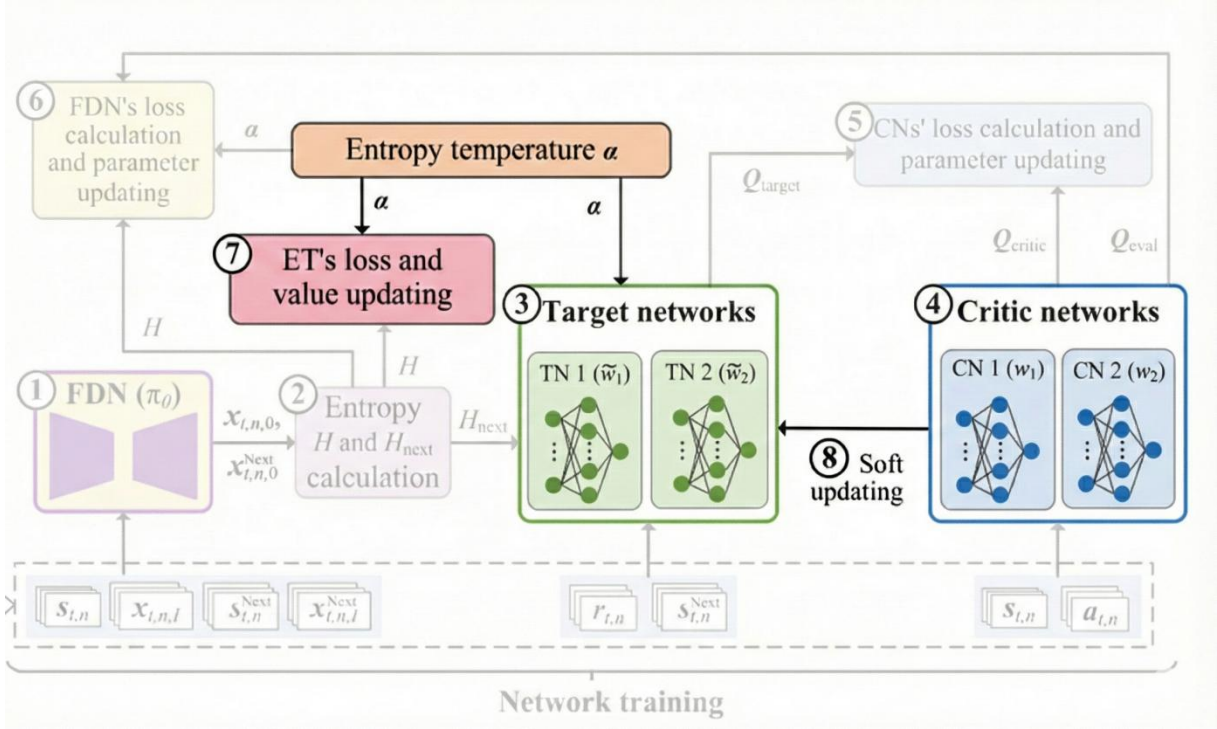
稳定化技巧

Step 6 ⑦

温度参数更新：
熵的权重 α 不是一个固定的超参数，而是可以自动学习和调整的。
目的：实现动态地平衡“探索”和“利用”。

Step 7 ⑧

目标网络软更新：
目的：防止 Q_{target} 值剧烈波动导致训练不稳定。
做法：目标网络的参数不是直接从Critic网络复制过来，而是通过一个很小的学习率进行缓慢、平滑的更新。



目录

实验评估

实验设置

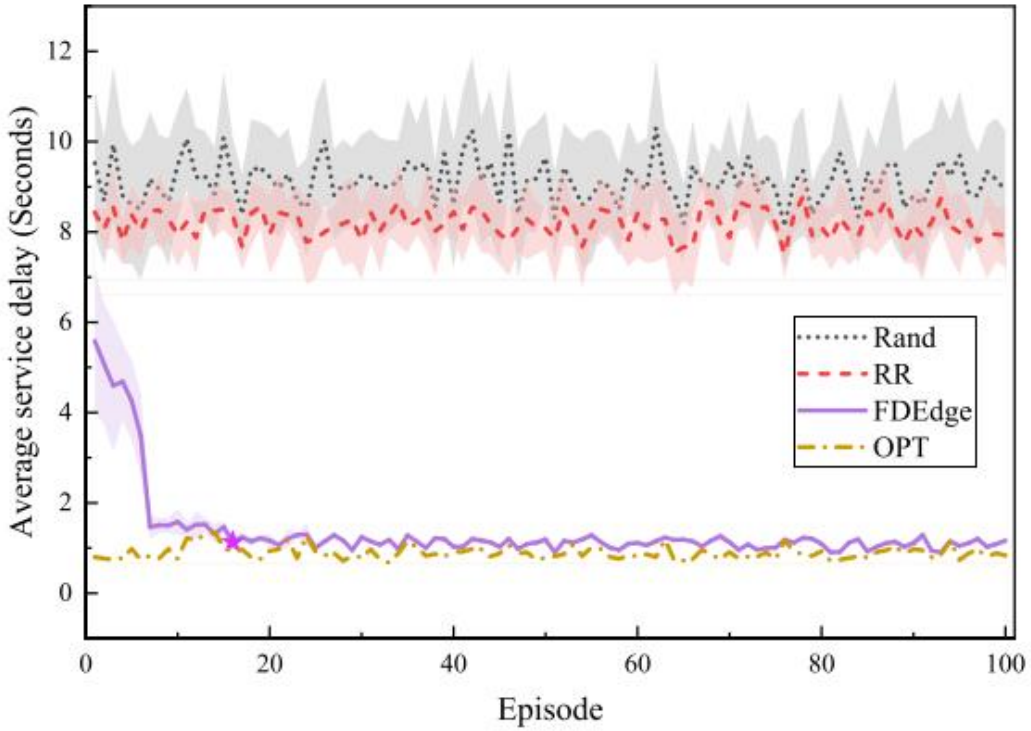
基本设置
编程框架：PyTorch
CPU：Intel Core i7
内存 (RAM)：32 GB

实验采用**模拟方式**
数据大小随机分布在10 ~ 40Mbits
计算密度100 ~ 300CPUcycles / bit

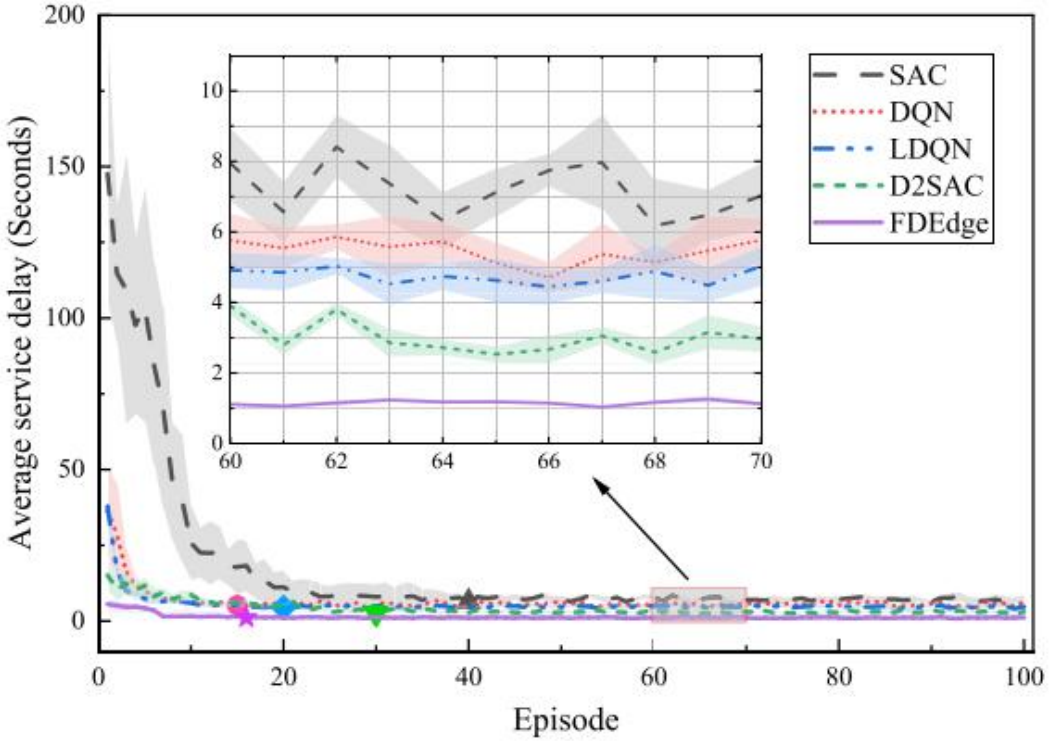
Baseline
1. **Rand**: 随机卸载（下限）。
2. **RR**: 轮询调度。
3. **SAC**: 标准的软动作-评论家算法（FDEdge 的基础框架）。
4. **LGDQN**：集成了长短期记忆（LSTM）网络的DQN模型
5. **D2SAC**: 基于扩散的 SAC（不带反馈机制），用来验证 Feedback 的有效性。
6. **OPT**: 理论最优解（假设已知所有信息，暴力搜索找到最优），作为性能上限。

实验评估

训练过程效果评估



(a) Comparison of our method against heuristic baselines and the OPT.

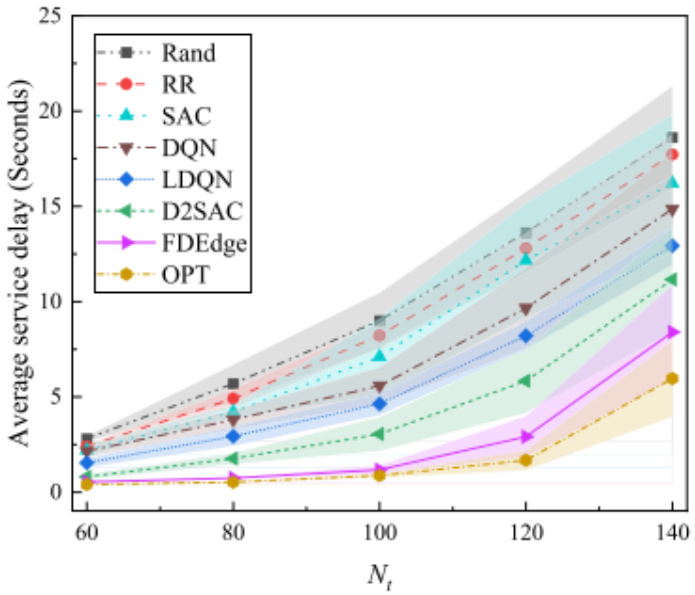


(b) Comparison of our method against the baselines of DRL and diffusion.

实验评估

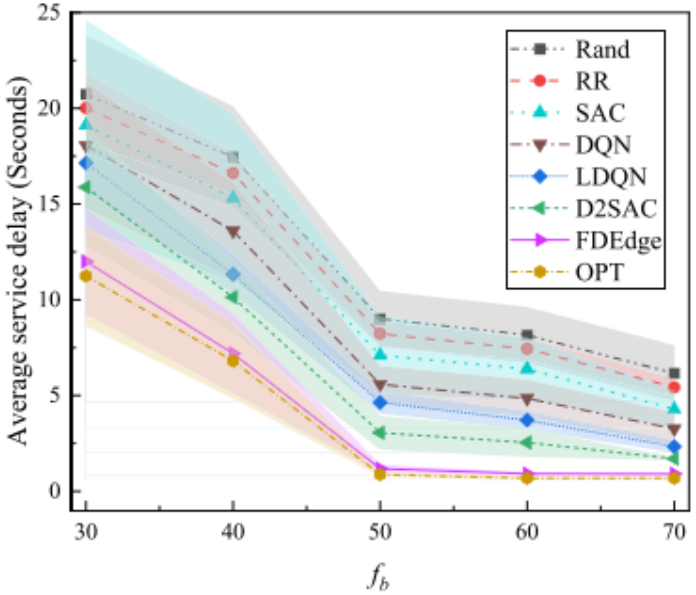
环境参数的影响评估

任务数量的影响



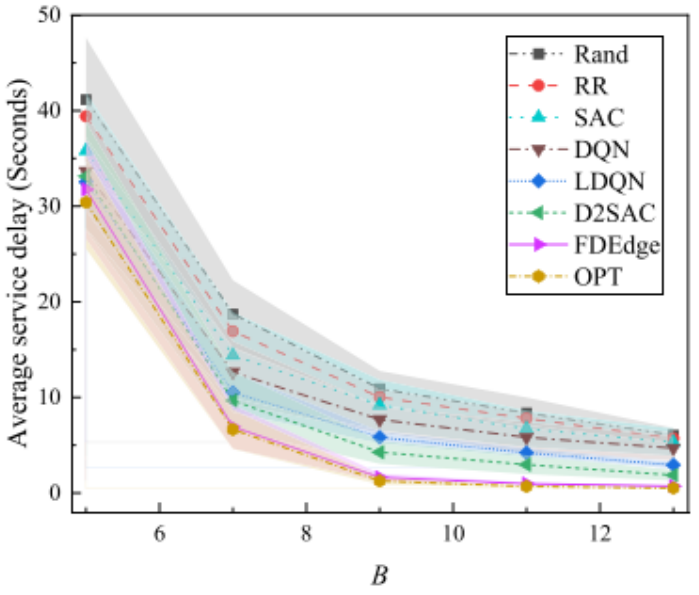
(a) Varying the number of tasks.

ES计算能力的影响



(b) Varying ES's computing capacity.

BS数量的影响



(c) Varying the number of BSs.

目录

总结与思考

- **总结**

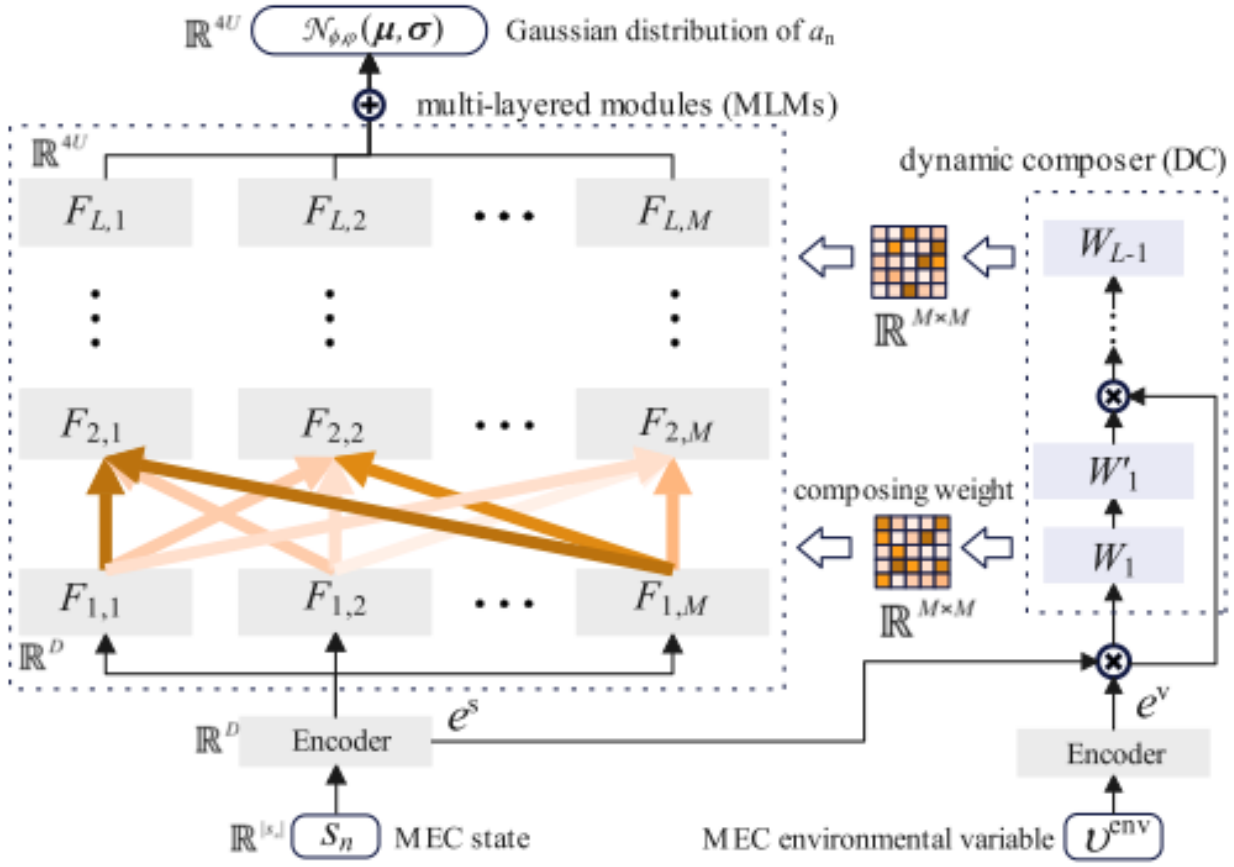
  本文提出的FDEdge实现边缘协同场景下的任务调度，提高了任务处理的服务质量（QoE）。作者首先将协作边缘计算系统的任务卸载问题表述为一个以最小化服务延迟为目标的在线INLP问题。为解决这个问题，作者设计了一个创新的FDN模型，将历史动作概率分布引入扩散模型，以提升模型的收敛速度。并基于此提出了一种新的FDEdge方法，通过将FDN模型与DRL技术相结合，在协作边缘系统中有效地进行任务调度。通过理论分析和实验验证证明了算法的合理性、有效性与调度效率。

总结与思考

- **基于本文**
  - FDEdge 的训练和推理假设任务分布和网络状态是相对稳定的，但在现实世界中，边缘环境是高度动态的。面对这种环境的切换，FDEdge也能够处理，但是却存在两个很致命的问题：
    1. **再适应成本高**：对于新的环境，FDEdge要花费大量的训练适应新的环境
    2. **灾难性遗忘**：当环境变化后，模型学习适应新环境，就会丢失旧环境中的策略，无法在多个环境间灵活切换

总结与思考

为了解决单一的神经网络模型在不同的环境中学习，易产生“灾难性遗忘”并且不同环境的策略之间互相干扰的问题，M^3OFF[1]提出了一种模块化组合的DRL算法。



M^3OFF针对移动边缘计算的场景

采用SAC的框架

模块组合Actor的结构

多层模块（MLMs）

由 L 层、每层 M 个模块（MLP）组成，这些 MLP 只负责处理专状态信息，提取特征。

动态组合器（DC）

DC负责接收状态state和环境特征的结合信息，根据当前环境的特点，输出一个权重矩阵。这个权重矩阵决定了MLMs中模块的连接的强弱

[1] INFOCOM’24 - M^3OFF: Module-Compositional Model-Free Computation Offloading in Multi-Environment MEC

总结与思考

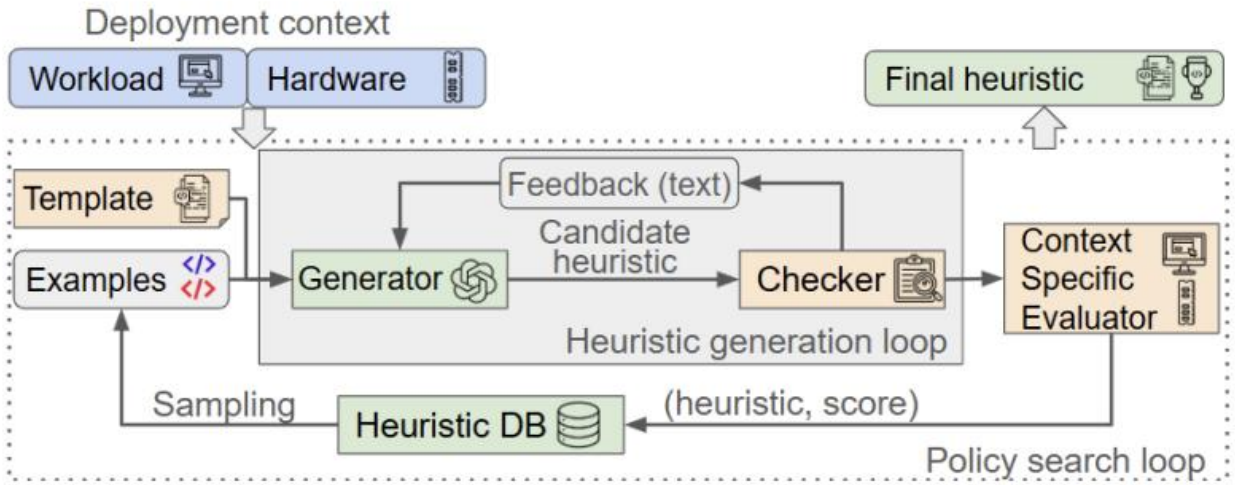
- **基于本文**
  - 现在的边缘任务调度领域，有众多为解决不同问题设计的调度算法，D2SAC将扩散模型引入SAC框架解决了样本利用率低的问题，FDEdge在D2SAC的基础上通过历史动作概率加速了模型的收敛， M^3OFF 利用模块化设计解决多环境适应的问题。我可以将这些论文分为两类：
    1. **针对某一特定问题，设计一种更好更优的算法方案**
    2. **设计一种算法方案能够针对多样化的场景**

    现阶段这两者通常是不能兼得的，因为设计一个更好更优的算法往往费时费力，而针对多样化场景的通用方法又往往会牺牲其在特定领域的表现。

  但是最近我看到的一篇代码生成领域的工作给了我一些启发

总结与思考

这是一篇发表在Hotnet'25上的一篇文章“Man-Made Heuristics Are Dead. Long Live Code Generators!”



“实例最优（Instance-Optimal）”且“可解释”

这篇文章提出了一种系统设计的新范式：利用大语言模型驱动的进化搜索（POLICYSMITH）来取代人类专家手工设计的启发式算法。

核心流程分为3个阶段：

1. 输入与初始化
   由用户提供模板来定义搜索的空间
2. 内层循环：生成与验证
   基于LLM的生成器会根据用户的模板和从数据库中提取的示例生成代码，并由检查器进行检查反馈
3. 外层循环：评估与进化
   特定场景评估器会将代码放到特定环境中跑，并打出一个性能分数。所有的代码都会被放进数据库中。当生成器需要示例时系统就会从数据库挑选表现最好的，供生产器进行突变和进化

总结与思考

- **泛化性**
  - FDEdge提出的将历史信息作为反馈引入扩散模型训练的思想是值得借鉴的，在除了任务调度这样的场景之外，在系统状态是随时间连续变化的动态组合优化的问题上都可以考虑利用这种时间相关性，引入历史的信息分布作为下一时刻的优化起点。
  - 在机器人控制，或者比如说具身智能的场景中，采用扩散模型去生成动作的轨迹，能够生成连续的控制轨迹，在这种连续控制的场景中，是否也可以将这种反馈机制引入其中以提升策略的响应速度。

请老师同学们批评指正!