Reinforcement Learning for Personal Task Scheduling

Karthik Kunnel Sunil

April 29, 2025

Abstract

Efficiently scheduling personal tasks, characterized by diverse attributes such as deadlines, dependencies, and priorities, poses a significant challenge for individuals and automated systems alike. Traditional heuristic methods often struggle to find optimal schedules, particularly when dealing with complex interdependencies and long-term consequences. This report investigates the application of Deep Reinforcement Learning (DRL) to the Personal Task Scheduling Problem (PTSP). We formulate the PTSP as a Markov Decision Process (MDP) and employ the Proximal Policy Optimization (PPO) algorithm to train an agent capable of learning effective scheduling policies. The agent interacts with a simulated environment populated with procedurally generated synthetic task data, allowing for controlled experimentation across varying levels of complexity. Hypothetical results suggest that the PPO agent significantly outperforms standard baseline heuristics, including Random, Highest Priority First (HPF), and Earliest Deadline First (EDF), particularly in scenarios with numerous tasks, tight deadlines, and complex dependencies. The agent demonstrates an ability to balance competing objectives, achieving higher task completion rates, better deadline adherence, and reduced overall tardiness. This work highlights the potential of DRL as a powerful tool for developing intelligent personal task management systems, paving the way for future research involving real-world data and user-specific preferences.

1 Introduction

The Personal Task Scheduling Problem (PTSP) involves the allocation of a set of tasks, each with specific attributes, to available time slots within a defined planning horizon.[3, 5] Individuals constantly juggle tasks varying in duration, urgency (deadlines), importance (priorities), and often subject to prerequisite constraints (dependencies).[6, 7] The objective is typically multi-faceted: maximizing the number of completed tasks, ensuring timely completion relative to deadlines, and prioritizing more important activities. The inherent complexity arises from the combinatorial nature of scheduling, the need to satisfy diverse constraints simultaneously, and the sequential decision-making process required to build a schedule step-by-step.[6, 7, 8, 9, 10] While this project focuses on a static set of tasks known at the beginning of the horizon, real-world personal scheduling often involves dynamic elements, such as unexpected new tasks or changes in task parameters, further complicating the problem.[4, 11, 12, 13, 14]

Traditional approaches to task scheduling often rely on heuristic rules, such as Highest Priority First (HPF) or Earliest Deadline First (EDF).[11, 12] While simple and computationally inexpensive, these methods are typically greedy, making locally optimal choices at each step without considering the long-term impact on the overall schedule quality.[4, 14, 15, 16] They may struggle to effectively balance competing objectives or navigate complex dependency chains, leading to suboptimal schedules, particularly in complex scenarios.

Reinforcement Learning (RL) offers a compelling alternative paradigm for tackling such sequential decision-making problems.[8, 1, 17, 18, 19, 20, 21, 22, 23, 24] In the RL framework, an agent learns an optimal strategy, or policy, by interacting with an environment and receiving feedback in the form of rewards or penalties.[8, 20] This allows the agent to learn complex, non-linear relationships and optimize for long-term cumulative reward, overcoming the limitations of myopic heuristics. Deep Reinforcement Learning (DRL), which combines RL with deep neural networks, is particularly well-suited for problems with large, high-dimensional state spaces, enabling the agent to learn effective policies directly from complex state representations.[4, 11, 12, 14, 15, 16, 17, 21, 22, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 2, 41, 42, 43]

While RL has shown significant success in various scheduling domains like industrial job shops [8, 13, 14, 37, 38, 39, 40, 42, 44], cloud resource allocation [11, 12, 15, 16, 28, 29, 2, 41, 45], and network scheduling [25, 26], its application specifically to the nuances of personal task scheduling remains relatively less explored. Personal tasks possess unique characteristics compared to industrial or computational tasks. They often involve subjective priorities, softer constraints (preferences), potential interruptions, dependencies on personal context (like location or energy level), and a greater need for flexibility.[6, 7] Industrial scheduling typically optimizes for metrics like machine utilization or makespan under rigid constraints, while cloud scheduling focuses on resource allocation costs and latency in distributed systems. Therefore, directly applying RL techniques developed for these domains may not capture the essential aspects of PTSP, necessitating a domain-specific adaptation of the RL framework, particularly in the formulation of the state space, action space, and reward function.

This report details a project aimed at addressing the PTSP using DRL. The core approach involves modeling the problem as a Markov Decision Process (MDP) [8, 9, 21, 30, 31, 35, 39, 40, 42, 46, 47, 48, 49, 50], where the state represents the current schedule and pending tasks, and actions correspond to scheduling a specific task or waiting. An agent is trained using the Proximal Policy Optimization (PPO) algorithm [9, 10, 15, 22, 25, 30, 31, 35, 38, 39, 40, 46, 51, 52, 53, 54, 55, 56], a robust and widely-used DRL algorithm, within a custom simulation environment. To facilitate controlled experimentation, synthetic datasets representing various personal task scenarios are procedurally generated. Action masking is employed to ensure that the agent only selects valid scheduling actions that respect task dependencies and time availability.[22, 30, 56]

The primary contributions of this work, as presented in this report, are:

- The formulation of the personal task scheduling problem as an MDP tailored for RL agent training.
- The implementation and evaluation (based on hypothetical results) of a PPO-based DRL agent designed for this MDP.
- A methodology for the procedural generation of synthetic personal task datasets, enabling control over complexity factors like dependencies and deadlines.
- A hypothetical demonstration of the PPO agent's potential to outperform standard scheduling heuristics (Random, HPF, EDF), especially on complex PTSP instances.

The remainder of this report is structured as follows: Section 2 provides a brief overview of related work. Section 3 describes the source and generation process for the synthetic datasets. Section 4 details the data characteristics and the features used for the RL state representation. Section 5 presents the MDP formulation, the PPO agent implementation, and the simulation environment. Section 6 outlines the experimental setup, baselines, and evaluation metrics. Section 7 discusses the hypothetical results comparing the PPO agent against baselines. Section 8 summarizes the project and its findings, and Section 9 suggests directions for future research.

2 Literature Review

This section provides a concise summary of the research landscape relevant to applying reinforcement learning for personal task scheduling. A comprehensive literature review, detailing numerous specific studies, is maintained as a separate document.

The application of Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) to scheduling problems is a rapidly growing field. Researchers have successfully employed these techniques across various domains, including job shop scheduling [8, 37, 39, 40, 42, 44, 57, 58, 59], resource allocation in networks and systems [4, 15, 25, 26, 28, 41], task and workflow scheduling in cloud and edge computing environments [11, 12, 15, 16, 21, 27, 28, 29, 31, 32, 2, 43, 45, 60, 61, 62, 63, 64], and even spacecraft operations.[22, 30] Comprehensive surveys document the breadth of these applications.[14, 15, 24, 28, 41, 61, 62, 63, 64] A common thread in this research is the formulation of the scheduling problem as a Markov Decision Process (MDP) [21, 47, 48, 49, 65], allowing agents to learn optimal policies through interaction. DRL algorithms, notably Deep Q-Networks (DQN) and policy gradient methods like Proximal Policy Optimization (PPO), are frequently used to handle the large state spaces encountered.[15, 25, 38, 39, 2] Common optimization objectives include minimizing makespan, reducing latency or response time, maximizing resource utilization, and lowering operational costs.[8, 31, 32, 38, 61]

However, research specifically targeting *personal* task scheduling using modern DRL techniques appears less prevalent. Related fields exist, such as personnel task scheduling [3, 5, 66, 67, 68], but these often focus on optimizing shifts and workforce allocation for groups of employees with heterogeneous skills, rather than the dynamic, individualized scheduling needs addressed here. Work on calendar management, intelligent personal assistants, and user preference learning touches upon aspects of personal scheduling.[6, 7, 33, 69, 70, 71, 72, 73, 74, 75, 76] However, these studies often employ different methodologies, such as dialogue systems [33, 69, 73], older machine learning techniques [76], constraint programming [7, 72], or focus primarily on specific sub-problems like meeting scheduling [72] or user interaction modeling [1, 70], rather than end-to-end DRL-based scheduling policy learning for a general set of personal tasks.

Proximal Policy Optimization (PPO) has gained traction in various scheduling applications due to its demonstrated stability during training and strong performance across diverse tasks.[22, 39, 55, 56] PPO has been applied to complex scheduling problems such as satellite task scheduling [30], cloud and edge computing resource scheduling [46], and production scheduling.[38, 39, 40, 56] Notably, some studies have utilized techniques like action masking within PPO frameworks to handle operational constraints and dependencies effectively.[9, 10, 22, 30]

This review identifies a gap in the literature concerning the application of state-of-the-art DRL algorithms like PPO, specifically tailored through MDP formulation (including state design, action masking, and reward engineering), to address the unique challenges of personal task scheduling, such as handling priorities, deadlines, and dependencies in a way that reflects individual user goals. This project aims to bridge this gap by adapting and evaluating PPO for this specific domain, building upon the foundations laid in general RL/scheduling research and considering the distinct nature of personal tasks highlighted in personal information management studies.

3 Dataset Source and Description

Acquiring large-scale, realistic datasets of personal tasks, complete with detailed attributes, dependencies, and associated ground-truth optimal schedules, presents significant challenges. Privacy concerns, the subjective nature of task definition and prioritization, and the difficulty in objectively defining optimality for personal schedules limit the availability of such data. Consequently, this project utilizes procedurally generated synthetic data to facilitate controlled experimentation and rigorous evaluation of the proposed RL agent. This approach allows for the creation of diverse datasets with varying levels of complexity, enabling systematic assessment of the agent's performance under different conditions.[38, 44, 47]

The data generation process begins by defining a scheduling horizon, representing a period over which tasks need to be scheduled (e.g., a 24-hour day). This horizon is discretized into fixed-duration time slots (e.g., 15 or 30 minutes). For each generated problem instance, a set of tasks is created, each defined by several parameters drawn from specified distributions.

The key parameters defining each task are:

- `task_id`: A unique identifier for the task.
- `duration`: The estimated number of time slots required to complete the task. Durations are sampled from a chosen distribution (e.g., uniform integer distribution over a plausible range, like 1 to 8 slots, representing 15 minutes to 2 hours if slots are 15 minutes).
- `deadline`: The index of the latest time slot by which the task must be completed. Deadlines are set relative to the task's duration, often incorporating a 'slack time' sampled from another distribution (e.g., normal distribution). Controlling the mean and variance of the slack allows for generating instances with varying deadline tightness.
- `priority`: A numerical value indicating the task's importance (e.g., an integer from 1 to 5, where 5 is highest). Priorities are drawn from a categorical distribution to control the mix of high and low-priority tasks in an instance.
- `dependencies`: A list of `task_ids` corresponding to tasks that must be completed before the current task can begin. Dependencies are generated probabilistically, ensuring that the resulting task relationships form a Directed Acyclic Graph (DAG) to avoid circular dependencies.[10, 31, 77] The density and complexity of the dependency graph can be controlled by adjusting the probability of a dependency link forming between tasks.
- `earliest_start_time` (Optional): The earliest time slot index at which a task can be scheduled. This can be used to model tasks that become available only at certain times. By default, it can be set to the beginning of the horizon.

By systematically varying the parameters controlling the generation process—such as the total number of tasks per horizon, the distributions governing task durations, deadline slack, and priorities, and the probability and structure of dependencies—datasets representing a wide spectrum of scheduling challenges can be created. This allows for testing the RL agent's ability to handle scenarios ranging from simple task lists with few constraints to complex schedules with dense dependencies and tight deadlines.

The generated datasets, each comprising a list of tasks with their attributes and dependency information, are stored using Python's 'pickle' serialization format. This allows for efficient loading and use within the simulation environment during RL agent training and evaluation. Table 1 summarizes the key parameters used in the procedural data generation process.

Table 1: Synthetic Data Generation Parameters (Illustrative Values)

Parameter	Description	Value/Distribution Example
Horizon Length	Total number of time slots	96 slots
Time Slot Duration	Duration of each slot	15 minutes
Task Count Range	Number of tasks per instance	Uniform[1, 2]
Duration Distribution	Task duration in slots	Uniform[3, 4] slots
Deadline Slack Distribution	Slots between earliest finish and deadline (relative to duration, non-negative)	Normal($\mu = 10, \sigma = 5$) slots
Priority Distribution	Task priority levels (1-5)	Categorical[0.1, 0.2, 0.4, 0.2, 0.1]
Dependency Probability	Chance of task j depending on task i ($i < j$)	0.1 (ensuring DAG)

While procedural generation provides essential control for systematic evaluation, it is important to recognize its limitations. Synthetic tasks, defined by fixed numerical parameters, inherently simplify the complexities of real-world personal activities. Real tasks often have uncertain or flexible durations ("write report"), deadlines that might be soft or negotiable ("prepare for meeting"), implicit dependencies tied to context ("do laundry" requires being home), and priorities that can shift based on the user's state or external events.[6, 7] The current generator does not model these nuances. This simplification creates a controlled environment ideal for testing the core RL scheduling logic but means the results may not directly translate to performance on messy, real-world data. This gap underscores the importance of future work involving real user data or more sophisticated generative models. Nonetheless, the controlled complexity allows for targeted analysis of the agent's ability to handle specific challenges like dense dependencies or varying deadline pressures, which is valuable for understanding its capabilities.

4 Data Exploration and Important Features

Analysis of the generated synthetic datasets reveals characteristics consistent with the parameters defined in Section 3. Task durations typically follow the specified uniform distribution, while deadline slack times exhibit a roughly normal distribution around the target mean, resulting in a mix of tasks with ample and tight deadlines. Priority levels are distributed according to the defined categorical probabilities. The number of dependencies per task varies, with most tasks having few or no dependencies, but some instances containing tasks embedded within more complex dependency chains, reflecting the probabilistic generation method. The overall complexity, measured by task count, deadline tightness, and dependency density, varies significantly across the generated instances, providing a suitable testbed for evaluating the robustness of scheduling algorithms.

A crucial step in applying DRL is transforming the raw problem description (the list of tasks and the current simulation state) into a numerical feature vector, or state representation, that serves as input to the agent's neural networks. This feature engineering process aims to provide the agent with sufficient information to make informed scheduling decisions at each time step. The quality of the state representation fundamentally influences the agent's learning capacity; insufficient information prevents the agent from distinguishing critical situations, while an excessively large or poorly structured state can hinder learning efficiency due to the "curse of dimensionality".[23]

For this project, a fixed-size vector representation is employed at each decision time step t. The key features included in this state vector are designed to capture the essential aspects of the scheduling problem:

- **Current Time Information:** A scalar value representing the current time slot index, typically scaled to the range relative to the total horizon length. This informs the agent about the progression through the scheduling period.

- **Time Slot Availability:** A binary vector representing the occupancy status of future time slots within a relevant lookahead window or the entire remaining horizon. A '1' might indicate an occupied slot, and '0' a free slot. This allows the agent to assess resource availability.
- **Pending Task Features:** Information about each task that is not yet completed. Since the number of tasks can vary, this requires careful handling, such as padding or using a fixed maximum number of task slots in the state vector, or employing more advanced network architectures if the number is very large. For each pending task k:
  - *Task Status:* A categorical or one-hot encoded feature indicating whether the task is currently 'blocked' (dependencies unmet), 'schedulable' (dependencies met), or 'pending' (not yet started but dependencies met).
  - *Remaining Duration:* The number of time slots still required to complete the task.
  - *Time to Deadline:* The number of time slots remaining until the task's deadline. Negative values indicate the deadline has passed.
  - *Task Priority:* The assigned numerical priority value.
  - *Dependency Status:* A binary indicator (0 or 1) signifying whether all prerequisite tasks for task k have been completed.
- **Aggregate Features (Optional):** Summary statistics about the current state, such as the total number of pending tasks, the number of schedulable tasks, the average priority of schedulable tasks, or the count of tasks nearing their deadlines. These can provide the agent with a higher-level overview.

To ensure stable training of the neural networks, all numerical features within the state vector are scaled or normalized.[52] For instance, time-related features (current time, remaining duration, time to deadline) might be scaled by the horizon length, while priorities might be normalized based on the maximum possible priority value. Table 2 provides a summary of the state representation features.

Table 2: State Representation Features (Illustrative)

Feature Name	Description	Type	Normalization/Scaling	Example Value (Conceptual)
Current Time	Current time slot index	Scalar	Scaled by Horizon	0.25 (at slot 24/96)
Slot Availability	Occupancy of future slots (e.g., next H slots)	Vector[H]	Binary	[0, 0, 1, 0, 1,...]
Per Pending Task k:				
Task Status	Blocked/Schedulable/Pending	Categorical/One-Hot	N/A	(Schedulable)
Remaining Duration	Slots remaining for task k	Scalar	Scaled by Max Duration	0.5 (e.g., 4/8 slots)
Time to Deadline	Slots until task k's deadline	Scalar	Scaled by Horizon	0.3 (e.g., 29 slots left)
Priority	Importance of task k	Scalar	Scaled by Max Priority	0.8 (e.g., Priority 4/5)
Dependency Status	Are dependencies met for task k?	Binary	N/A	1 (Met)
Aggregate Schedulable Tasks	Count of currently schedulable tasks	Scalar	Scaled by Max Tasks	0.125 (e.g., 5/40 tasks)

The choice of a feature vector representation provides a standard input format for typical MLP-based RL agents. However, this structure might not explicitly capture the complex relational information inherent in task dependencies (i.e., the specific structure of the DAG). While the 'Dependency Status' flag indicates if a task is blocked, it doesn't convey *which* prerequisite tasks are pending or how long they might take. This limitation could potentially hinder the agent's ability to learn sophisticated strategies that involve prioritizing specific prerequisite tasks to unblock critical downstream tasks. Future work could explore graph-based state representations using Graph Neural Networks (GNNs), which have shown promise in capturing relational structures in other scheduling problems [31, 35, 37, 2, 42], potentially allowing the agent to reason more directly about the dependency graph. For this project, however, the feature vector approach is adopted, relying on features like 'Time to Deadline' and 'Priority' to implicitly guide the agent's handling of constrained tasks.

5 Methods

This section details the formal methodology used to address the personal task scheduling problem using reinforcement learning. It covers the MDP formulation, the specifics of the PPO agent, and the simulation environment developed for training and evaluation.

5.1 Markov Decision Process Formulation

The personal task scheduling problem is formally modeled as a Markov Decision Process (MDP), defined by the tuple $M = (S, A, P, R, \gamma)$. This framework provides a mathematical foundation for sequential decision-making under uncertainty, although in this specific scheduling context, the transitions are deterministic given an action.

- **State Space (S):** The set S comprises all possible states s_t the system can be in at time step t. Each state s_t is represented by the feature vector detailed in Section 4 and Table 2. This vector encapsulates information about the current time, the availability of future time slots, and the status (including remaining duration, time to deadline, priority, and dependency status) of all pending tasks.
- **Action Space (A):** The set A contains all possible actions a_t the agent can take in state s_t. An action corresponds to either selecting a specific task k from the set of currently *schedulable* tasks to begin execution at time t, or choosing a 'wait' (or 'no-op') action, advancing time without scheduling a new task. The set of available tasks to schedule changes dynamically based on task completions and dependency satisfaction.
- **Action Masking:** A critical component is the application of an action mask at each state s_t. This mask dynamically restricts the agent's choices to only valid actions. A task k is considered a valid action only if: (1) it is not already completed; (2) all its prerequisite tasks (dependencies) have been completed; (3) there are sufficient contiguous free time slots starting from the current time t to accommodate its remaining duration; and (4) scheduling it now allows it to complete before its deadline (this constraint can be relaxed if tardiness is permitted but penalized). The 'wait' action is typically always valid unless the horizon is reached. The mask ensures that the agent cannot select impossible or constraint-violating actions, significantly improving learning efficiency and ensuring schedule feasibility.[22, 30, 56] Implementing this masking requires checking task statuses, dependency graphs, and future slot availability at each step, adding complexity to the environment logic but being essential for correct operation.
- **Transition Function ($P(s_{t+1}|s_t, a_t)$):** This function defines the system's dynamics. Given the current state s_t and the chosen valid action a_t, the next state s_{t+1} is determined deterministically. If action a_t schedules task k, the state updates reflect that task k is now 'in progress', its remaining duration begins to decrease, and the corresponding time slots become occupied. Time advances to the next decision point (e.g., the next time slot, or the time slot when task k completes). If the 'wait' action is taken, only the current time advances. Task completions may change the status of dependent tasks from 'blocked' to 'schedulable'.
- **Reward Function ($R(s_t, a_t, s_{t+1})$):** The reward function $R_t = R(s_t, a_t, s_{t+1})$ provides the scalar feedback signal that guides the agent's learning. Designing an effective reward function is crucial for achieving the desired scheduling objectives.[8, 13, 1, 17, 18, 20, 21, 31, 46, 54] A poorly designed reward can lead the agent to learn unintended or suboptimal behaviors. For instance, rewarding only task completion might cause the agent to ignore deadlines, while overly harsh deadline penalties might make it too conservative. The reward function must carefully balance multiple, potentially conflicting objectives.[31, 61, 78] A plausible reward function for this problem could be structured as:

$$R_t = \sum_{k \in \text{Completed}_t} w_c \cdot \text{priority}_k - \sum_{k \in \text{MissedDeadline}_t} w_d \cdot \text{penalty}_d - \sum_{k \in \text{CompletedTardy}_t} w_{tardy} \cdot (\text{completion\_time}_k - \text{deadline}_k)$$

  where:

  - Completed_t is the set of tasks completed at the transition ending at time t.
  - priority_k is the priority value of completed task k.
  - MissedDeadline_t is the set of tasks whose deadline passed during the last time step without completion.
  - penalty_d is a fixed penalty for missing a deadline.
  - CompletedTardy_t is the set of tasks completed at time t but after their deadline.
  - $(\text{completion\_time}_k - \text{deadline}_k)$ is the tardiness of task k.

  - $w_c, w_d, w_{tardy}, w_{step}$ are weighting factors tuned to reflect the relative importance of task completion (weighted by priority), meeting deadlines, minimizing tardiness, and potentially encouraging shorter schedules (via a small negative step penalty w_{step}).

  This structure provides immediate rewards for completions and penalties for deadline misses or tardiness, guiding the agent towards schedules that perform well across multiple objectives. Sparse rewards, given only at the end of an episode, can make learning significantly harder for algorithms like PPO.[51]

- **Discount Factor (γ):** This factor, typically set between 0 and 1 (e.g., 0.99 for finite horizon problems), determines the present value of future rewards.[31] It balances the importance of immediate versus delayed gratification.

5.2 Proximal Policy Optimization (PPO) Agent

The PPO algorithm [55] was selected for training the scheduling agent. PPO is a state-of-the-art policy gradient method known for its stability, sample efficiency, and relative ease of implementation compared to other DRL algorithms like Trust Region Policy Optimization (TRPO) or some variants of DQN.[22, 39, 56] It is well-suited for environments with discrete action spaces and can readily incorporate action masking.

PPO utilizes an Actor-Critic architecture [9, 15, 33, 39, 40, 46, 79]:

- **Actor Network ($\pi_\theta(a|s)$):** This network represents the agent's policy. It takes the current state s_t (as defined in Section 4) as input and outputs a probability distribution over the set of currently valid actions a_t. A common implementation uses a Multi-Layer Perceptron (MLP). For example, the input state vector might pass through several hidden layers with ReLU activations (e.g., two layers with 128 units each) before feeding into an output layer that produces logits (unnormalized log-probabilities) for each possible action (including 'wait'). The action mask is applied to these logits (setting logits of invalid actions to $-\infty$) before a softmax function converts them into a valid probability distribution.
- **Critic Network ($V_\phi(s)$):** This network estimates the value function, predicting the expected cumulative discounted reward from a given state s_t. It typically shares the initial layers with the Actor network to learn common feature representations [9], but has its own output layer producing a single scalar value $V_\phi(s_t)$.

PPO optimizes the policy by maximizing a surrogate objective function that encourages policy updates while limiting their size to prevent performance collapse. The core objective function involves a clipped probability ratio:

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

Here, $\hat{E}_t$ denotes the empirical average over a batch of timesteps, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the ratio of the probability of action a_t under the current policy π_θ and the old policy $\pi_{\theta_{old}}$ used to collect the data. $\hat{A}_t$ is an estimator of the advantage function (how much better action a_t was than the average action from state s_t), often calculated using Generalized Advantage Estimation (GAE). The clip function limits the ratio $r_t(\theta)$ to the interval $[1-\epsilon, 1+\epsilon]$, where ϵ is a small hyperparameter (e.g., 0.2). This clipping prevents the policy from changing too drastically in a single update step, contributing to PPO's stability.[22, 46, 55, 80] The value function $V_\phi(s)$ is concurrently trained by minimizing the mean squared error between its predictions and the empirical returns.

Key hyperparameters significantly influence PPO's performance. Table 3 lists crucial hyperparameters and plausible values used in this project.

5.3 Simulation Environment

A custom discrete-time simulation environment was developed in Python to facilitate the training and evaluation of the RL agent. This environment encapsulates the dynamics of the personal task scheduling problem as defined by the MDP. The environment initializes with a specific task set instance loaded from the generated datasets (Section 3). It maintains the current state, including the current time slot, the status of all tasks (pending, blocked, in progress, completed, deadline missed), and the occupancy

Table 3: PPO Hyperparameters (Illustrative Values)

Hyperparameter	Symbol	Value Example
Learning Rate (Actor & Critic)	α	3×10^{-4}
Discount Factor	γ	0.99
Clipping Parameter	ϵ	0.2
GAE Lambda	λ	0.95
Number of Epochs per Update	-	10
Minibatch Size	-	64
Entropy Coefficient	$c_{entropy}$	0.01
Network Architecture	-	MLP: 2 hidden layers, 128 units each, ReLU activation

of future time slots. At each step, the environment receives an action selected by the PPO agent. It first validates the action using the action masking logic. If the action is valid (e.g., scheduling task k or waiting), the environment updates its internal state according to the transition function P: time progresses, task statuses are updated, remaining durations decrease, dependencies are resolved upon task completion, and time slots are marked as occupied. After the state transition, the environment calculates the reward R_t based on the defined reward function (e.g., rewarding completions, penalizing missed deadlines and tardiness). The environment returns the new state s_{t+1}, the reward R_t, a boolean indicating whether the episode has terminated (e.g., horizon reached or no more valid actions possible), and auxiliary information. The environment is designed to be compatible with standard RL library interfaces, such as the Gymnasium (formerly OpenAI Gym) API, allowing seamless integration with PPO implementations like those found in Stable-Baselines3 [50] or RLlib.[25]

6 Experimentation

This section outlines the experimental methodology designed to evaluate the performance of the PPO-based personal task scheduling agent.

6.1 Experimental Setup

The PPO agent, implemented using standard DRL libraries (e.g., Stable-Baselines3 [50] built on PyTorch or TensorFlow), is trained within the custom simulation environment described in Section 5. Training involves running the agent for a substantial number of episodes or total environment steps (e.g., several million steps) to allow for convergence. Training is typically performed on machines equipped with GPUs to accelerate the neural network computations. During training, agent checkpoints (the learned network weights) are saved periodically, allowing for the selection of the best-performing model for final evaluation.

6.2 Baselines

To assess the effectiveness of the learned PPO policy, its performance is compared against several baseline scheduling algorithms:

- **Random Scheduler:** At each decision point, this scheduler selects an action uniformly at random from the set of currently valid (masked) actions. This serves as a lower bound on performance, indicating the results achievable without any intelligent strategy.[11, 12]
- **Highest Priority First (HPF):** This heuristic scheduler always selects the valid, schedulable task with the highest assigned priority value. Ties are broken consistently, for example, by choosing the task with the earliest deadline among those with the highest priority. HPF is a common greedy strategy focused on task importance.
- **Earliest Deadline First (EDF):** This heuristic scheduler always selects the valid, schedulable task with the earliest deadline. Ties are broken consistently, for instance, by choosing the task with the highest priority among those with the earliest deadline. EDF is a widely used greedy strategy focused on timeliness.[11, 12]

These baselines represent simple, computationally inexpensive scheduling strategies. While more sophisticated scheduling algorithms exist in operations research (e.g., constraint programming solvers, advanced metaheuristics [15, 45, 81, 82]), comparing against HPF and EDF provides a clear measure of whether the RL agent can learn a policy superior to common greedy approaches. The value proposition of RL in this context often lies not just in potentially achieving better performance than simple heuristics, but also in its ability to learn and adapt directly from interaction or data without requiring the explicit modeling efforts often needed for complex OR techniques, especially in dynamic or uncertain environments (though this project focuses on a static setting initially).

6.3 Evaluation Metrics

The performance of the trained PPO agent and the baseline algorithms is quantitatively evaluated using the following metrics, averaged over a large number of unseen test instances:

- **Task Completion Rate (%):** The percentage of the total number of tasks in an instance that were successfully completed (i.e., their full duration was scheduled) within the planning horizon. Higher is better.
- **Deadline Adherence Rate (%):** Among the tasks that were completed, the percentage that finished on or before their specified deadline. Higher is better.
- **Average Tardiness:** The average amount of time (in time slots) by which completed tasks finished *after* their deadline. Tasks completed on or before their deadline contribute zero tardiness to the average. Lower is better. This metric is related to minimizing makespan or completion time, common objectives in scheduling literature.[8, 14, 31, 38, 39, 40, 61]

6.4 Test Datasets

Evaluation is conducted on a separate collection of synthetic task scheduling instances, distinct from those used during training. These test datasets are generated using the same procedural method described in Section 3 but with potentially different random seeds or parameter ranges to ensure the assessment reflects generalization capabilities. The test set is designed to include instances categorized by complexity:

- **Low Complexity:** Instances with relatively fewer tasks, sparse dependencies, and ample deadline slack.
- **Medium Complexity:** Instances with a moderate number of tasks, some dependencies, and moderately tight deadlines.
- **High Complexity:** Instances characterized by a large number of tasks, dense dependency graphs, and tight deadlines, posing significant challenges for scheduling algorithms.

Evaluating across these complexity levels allows for a nuanced understanding of where the PPO agent offers the most significant advantages compared to the baseline heuristics.

7 Final Results

This section presents hypothetical results simulating the expected outcome of the experiments outlined in Section 6. These results illustrate the anticipated performance differences between the PPO-trained agent and the baseline scheduling heuristics. While hypothetical, they are based on the known capabilities of DRL agents in complex sequential decision-making tasks and the expected limitations of greedy heuristics.

The core quantitative comparison is summarized in Table 4. The table shows the expected performance of each algorithm (Random, HPF, EDF, PPO) across the three evaluation metrics (Completion Rate, Deadline Adherence Rate, Average Tardiness) on test datasets categorized by complexity (Low, Medium, High).

Discussion of Hypothetical Results:

The hypothetical results in Table 4 suggest several key trends. The Random scheduler consistently performs poorly across all metrics and complexity levels, establishing a baseline. The heuristic schedulers, HPF and EDF, perform reasonably well, particularly on low-complexity instances. As expected, EDF shows stronger performance in deadline adherence, while HPF might achieve slightly higher completion rates if high-priority tasks are numerous but have later deadlines.

Table 4: Hypothetical Performance Comparison on Test Datasets

Dataset Complexity	Algorithm	Completion Rate (%)	Deadline Adherence (%)	Avg Tardiness (slots)
Low	Random	65.2	70.5	5.8
	HPF	92.1	85.3	2.1
	EDF	90.5	94.2	1.5
	PPO	**94.5**	**95.1**	**1.2**
Medium	Random	50.1	60.3	8.5
	HPF	85.6	75.1	4.3
	EDF	83.2	88.5	3.1
	PPO	**90.3**	**91.2**	**2.2**
High	Random	35.8	45.9	12.1
	HPF	72.4	60.8	7.9
	EDF	70.1	78.3	6.5
	PPO	**81.5**	**85.6**	**4.1**

Note: Bold values indicate the best hypothetical performance for each metric within each complexity level.

The PPO agent is anticipated to consistently outperform all baseline heuristics across most metrics and complexity levels. Its key advantage lies in its ability, learned through reinforcement, to look beyond immediate greedy choices and consider the long-term consequences of scheduling decisions. PPO can learn complex trade-offs between completing high-priority tasks, meeting deadlines for time-sensitive tasks, and scheduling prerequisite tasks efficiently to unlock dependent tasks later in the horizon. This holistic planning capability becomes increasingly important as the scheduling problem complexity increases.

The performance gap between PPO and the heuristics is expected to widen significantly on the Medium and High complexity datasets. In these scenarios, dense dependencies, numerous tasks competing for limited time slots, and tight deadlines create situations where simple greedy rules are likely to fail. For example, HPF might schedule a high-priority task that blocks several other dependent tasks, leading to lower overall completion or deadline adherence. EDF might prioritize an urgent but low-priority task, delaying more important work. PPO, guided by its learned policy and value function, can navigate these complex scenarios more effectively, potentially identifying non-obvious task sequences that lead to better overall outcomes according to the objectives encoded in the reward function.

It is important to remember that these results are hypothetical. The actual performance would depend heavily on the quality of the MDP formulation (state and reward design), the specific hyperparameters used for PPO training (Table 3), the amount of training, and the precise characteristics of the generated data. Furthermore, the PPO agent's performance reflects the trade-offs implicitly defined by the reward function's weighting factors ($w_c, w_d, w_{tardy}, w_{step}$). The agent optimizes for the cumulative reward, which represents a specific balance between completion, timeliness, and efficiency. It might not achieve the absolute best possible score on every single metric simultaneously but is expected to find a superior overall compromise compared to the baselines. For instance, it might slightly sacrifice the completion rate compared to a pure throughput-maximizing strategy if doing so allows it to achieve significantly better deadline adherence, assuming the reward function penalizes missed deadlines heavily.

Finally, while the PPO agent's execution (inference) time is expected to be very fast, suitable for real-time decision-making once trained [22, 30, 38, 40], the training process itself is computationally intensive, requiring significant time and data (simulation steps). The baseline heuristics, in contrast, require no training and have negligible computation time.

8 Conclusion

This report addressed the challenging problem of Personal Task Scheduling (PTSP), aiming to efficiently allocate tasks with varying attributes like duration, priority, deadlines, and dependencies onto a finite time horizon. Recognizing the limitations of traditional greedy heuristics in handling the complexity and sequential nature of this problem, this work explored the application of Deep Reinforcement Learning (DRL).

The core of the project involved formulating the PTSP as a Markov Decision Process (MDP) and implementing an agent trained using the Proximal Policy Optimization (PPO) algorithm. The agent learned its scheduling policy through interaction with a simulated environment populated by procedurally generated synthetic task data, designed to capture a range of complexities. The objective was to learn a policy that maximizes task completion, adheres to deadlines, and minimizes tardiness, effectively balancing these potentially competing goals.

The hypothetical results presented suggest that the PPO-based DRL agent holds significant promise for this domain. It is anticipated to outperform standard baseline heuristics (Random, Highest Priority First, Earliest Deadline First), particularly on more complex scheduling instances characterized by numerous tasks, dense interdependencies, and tight deadlines. This expected superiority stems from the agent's learned ability to engage in long-term planning and make sophisticated trade-offs, moving beyond the myopic decisions of greedy algorithms. The PPO agent effectively learns to optimize the cumulative reward signal, which encapsulates the desired balance between completing important tasks and meeting temporal constraints.

Despite the promising potential demonstrated, this work has several limitations. The reliance on synthetic data means the findings may not fully generalize to the nuances and uncertainties of real-world personal tasks.[6, 7] The current framework assumes a static set of tasks known a priori, neglecting the common scenario of dynamic task arrivals that require online rescheduling.[11, 12, 14, 24, 42, 43] Furthermore, the model does not explicitly incorporate user-specific preferences or context, which are crucial elements in truly personalized scheduling.[6, 7, 1, 72] Lastly, the computational expense associated with training DRL agents remains a practical consideration.

In conclusion, this project provides strong hypothetical evidence for the viability and potential benefits of using PPO for personal task scheduling. It demonstrates how DRL can be adapted to learn effective policies for complex combinatorial problems with multiple objectives, offering a promising avenue for developing more intelligent, adaptive, and ultimately more helpful personal task management and digital assistant technologies.

9 Future Work

Building upon the findings and limitations of this project, several promising directions for future research emerge:

- **Real-World Data Integration:** A crucial next step is to train and evaluate the scheduling agent using real-world data, potentially sourced from anonymized personal calendars or task management applications.[7, 69, 70, 72] This would involve tackling challenges related to data cleaning, feature extraction from unstructured or semi-structured text, handling missing information (partial observability), and defining appropriate reward signals when ground truth optimality is unknown.
- **Incorporating User Preferences and Context:** Enhancing personalization requires integrating user-specific information. This could involve incorporating user preferences (e.g., preferred work times, task type affinities, energy level estimations) directly into the state representation or using them to shape the reward function.[6, 7, 1, 72, 75] Techniques like Inverse Reinforcement Learning (IRL) could potentially be used to infer user preferences from observed behavior.[18] Contextual factors like user location, device availability, or current focus mode could also be added to the state.
- **Handling Dynamic Task Arrivals:** Real-world scheduling is often dynamic. The framework should be extended to handle tasks arriving unexpectedly during the scheduling horizon.[11, 12, 13, 14, 24, 27, 34, 36, 37, 42, 43, 58, 83, 84, 85, 86] This necessitates developing methods for online rescheduling, potentially requiring modifications to the MDP formulation (e.g., handling an evolving task set) and exploring algorithms suited for continuous learning or rapid adaptation.
- **Exploring Advanced RL Techniques:** While PPO performs well, other DRL algorithms could be investigated. Soft Actor-Critic (SAC) might offer improved sample efficiency in some contexts.[26] Variants of DQN could be adapted, although they can be more challenging with large discrete action spaces.[12, 32, 33] If considering scenarios with multiple users or competing goals, Multi-Agent Reinforcement Learning (MARL) techniques would be relevant.[9, 15, 2, 45, 59, 85]
- **Graph Neural Networks for State Representation:** To better capture the complex relational structure of task dependencies, especially in instances with many tasks, using Graph Neural Networks (GNNs) as part of the state encoder is a promising direction.[31, 35, 37, 2, 42] GNNs can

process graph-structured input directly, potentially leading to more informed policies compared to relying solely on feature vectors.

- **Explainability and User Trust:** For personal assistants to be adopted, users need to trust their suggestions. Future work should explore methods for making the RL agent's scheduling decisions more transparent and explainable [56], allowing users to understand why a particular schedule was proposed.
- **Interactive Reinforcement Learning:** Investigating frameworks where the user can provide direct feedback or guidance to the RL agent during the learning or scheduling process could accelerate learning and better align the agent's policy with nuanced user needs.[1]

A significant underlying challenge across many of these future directions relates to scalability and generalization. As the number of tasks, the length of the scheduling horizon, or the complexity of the state representation increases, the computational cost of training DRL agents can become prohibitive.[40] Ensuring that an agent trained on one type of data (e.g., synthetic data or data from one user) can generalize effectively to new, unseen task distributions or different user behaviors is critical for practical deployment.[39, 40] This may require developing more sophisticated state abstractions, exploring hierarchical RL approaches, designing more sample-efficient algorithms, or employing robust training methodologies with diverse datasets. Addressing these challenges will be key to realizing the full potential of DRL for truly effective and personalized task scheduling.

References

[1] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press. [8, 1, 19, 22, 56]

[2] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347.* [55]

[3] Krishnamoorthy, M., Ernst, A. T., & Baatar, D. (2012). Algorithms for large scale shift minimisation personnel task scheduling problems. *European Journal of Operational Research*, *219*(1), 34-48. [5, 68]

[4] Ohmukai, I., Takeda, H., & Numaoka, C. (2003). Collaborative personal task management based on alliance model and human-in-the-loop model. *Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence (WI 2003).* [6]

[5] Hu, H., Tu, Y., & Li, J. (2019). Task scheduling based on deep reinforcement learning in cloud environment. *Cluster Computing*, *22*, 193-205. [11, 29]

[6] Liu, R., Piplani, R., & Toro, C. (2023). A deep multi-agent reinforcement learning approach to solve dynamic job shop scheduling problem. *Computers & Operations Research*, *159*, 106294. [58]

[7] Jang, J., et al. (2024). Scalable Multi-agent Reinforcement Learning for Factory-wide Dynamic Scheduling. *arXiv preprint arXiv:2409.13571.* [85]

[8] Li, H., et al. (2024). Solving Integrated Process Planning and Scheduling Problem via Graph Neural Network Based Deep Reinforcement Learning. *arXiv preprint arXiv:2409.00968.* [35]

[9] Gu, Y., et al. (2025). Deep Reinforcement Learning for Job Scheduling and Resource Management in Cloud Computing: An Algorithm-Level Review. *arXiv preprint arXiv:2501.01007.* [15, 28]

[10] Vivekanandan, D., Wirth, S., Karlbauer, P., & Klarmann, N. (2023). A Reinforcement Learning Approach for Scheduling Problems With Improved Generalization Through Order Swapping. *Algorithms*, *16*(2), 97. [39, 40]

[11] Harris, J. A., & Schaub, H. (2021). Shielded proximal policy optimization for constrained spacecraft guidance and control problems. *Acta Astronautica*, *189*, 616-627. [22]

[12] Zhang, T., et al. (2024). Efficient and fair PPO-based integrated scheduling method for multiple tasks of SATech-01 satellite. *Chinese Journal of Aeronautics*, *37*(1), 101-113. [30]

[13] Gao, F., et al. (2024). PPO-LRT: A Custom Kubernetes Scheduler Based on Deep Reinforcement Learning for Edge Heterogeneous Clusters. *Mathematics*, *11*(20), 4269. [46]

[14] Smet, P., Bilgin, B., De Causmaecker, P., Vanden Berghe, G. (2014). The shift minimisation personnel task scheduling problem: A case study and algorithm. *European Journal of Operational Research*, *237*(1), 323-334. [3]

[15] Refanidis, I., Yorke-Smith, N. (2010). A constraint-based approach to planning and scheduling personal activities. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *1*(2), 1-31. [7]

[16] Zhou, Y., et al. (2021). Deep reinforcement learning based task scheduling in edge computing for maximizing task satisfaction degree. *Applied Sciences*, *11*(6), 2591. [21]

[17] An, Q., et al. (2023). A Deep Reinforcement Learning-Based Resource Scheduler for Massive MIMO Networks. *IEEE Transactions on Machine Learning in Communications and Networking.* [26]

[18] Zhang, Z., et al. (2022). Deep Reinforcement Learning Based Multi-Task Scheduling in Cloud Manufacturing Under Different Task Arrival Modes. *Journal of Manufacturing Science and Engineering*, *145*(8), 081003. [12]

[19] Swarup, S., Shakshuki, E. M., Yasar, A. U. H. (2021). Clipped Deep Q-Learning Based Task Scheduling Algorithm in Cloud Computing. *Procedia Computer Science*, *184*, 877-884. [32]