FPT UNIVERSITY QUY NHON AI CAMPUS

FACULTY OF INFORMATION TECHNOLOGY

REINFORCEMENT LEARNING

FPT Education

FPT UNIVERSITY

MINI CAPSTONE

CUTTING STOCK 2D PROBLEM

ARTIFICIAL INTELLIGENCE

STUDENT: **HA KHAI HOAN**
QE170157
DANG PHUC BAO CHAU
QE170060
NGUYEN VAN SY THINH
SE173018
NGUYEN VAN THU
QE170147
NGUYEN QUOC VUONG
QE170168

QUY NHON CITY, 3/2025

INSTRUCTOR: PhD. **NGUYEN AN KHUONG**

FPT UNIVERSITY QUY NHON AI CAMPUS

FACULTY OF INFORMATION TECHNOLOGY

REINFORCEMENT LEARNING

FPT UNIVERSITY

MINI CAPSTONE

CUTTING STOCK 2D PROBLEM

ARTIFICIAL INTELLIGENCE

STUDENT: **HA KHAI HOAN**
QE170157
DANG PHUC BAO CHAU
QE170060
NGUYEN VAN SY THINH
SE173018
NGUYEN VAN THU
QE170147
NGUYEN QUOC VUONG
QE170168

QUY NHON CITY, 3/2025

INSTRUCTOR: PhD. **NGUYEN AN KHUONG**

Acknowledge

During the process of completing the course project on this topic, we received a lot of advice and advice from Mr. Nguyen An Khuong. We would like to send our most sincere thanks to the teachers and friends who have contributed and helped us during the implementation of the topic. In particular, we would like to thank Mr. Nguyen An Khuong, who has always been by my side during the implementation of the topic. The comments, advice, evaluations and roadmap that he set out from the first days of the thesis really left a great meaning, influencing the thinking and development direction during the implementation of this topic. Of course, the topic will still have shortcomings and limitations, we look forward to receiving frank comments and evaluations so that we can learn more knowledge, thereby being able to have a more polished product in the future. We sincerely thank you!

Project Implementation Team
(Sign and print full name)

Ha Khai Hoan

Dang Phuc Bao Chau

Nguyen Van Thu

Nguyen Van Sy Thinh

Nguyen Quoc Vuong

Promise

The graduation project team guarantees that the project is based on some previous documents and does not copy the content or results of other projects. The referenced content has been fully cited.

Project Implementation Team
(Sign and print full name)

Ha Khai Hoan

Dang Phuc Bao Chau

Nguyen Van Thu

Nguyen Van Sy Thinh

Nguyen Quoc Vuong

Abstract

The two-dimensional cutting stock problem (2D-CSP) is a fundamental optimization challenge in various manufacturing industries, such as glass, metal, and textile production. This project explores the application of reinforcement learning, specifically Proximal Policy Optimization (PPO) and Q-learning, to optimize material usage and minimize waste. Traditional heuristic methods, including First Fit and Best Fit algorithms, are also examined to compare their efficiency with AI-based approaches. By integrating deep reinforcement learning with heuristic methods, we aim to develop an intelligent cutting strategy that reduces material waste, optimizes cutting patterns, and improves overall production efficiency. The experimental results demonstrate the advantages and limitations of each method, highlighting the potential of reinforcement learning in solving industrial optimization problems.

Contents

List of Tables

List of Figures

List of Acronyms

Below is a list of abbreviations used in the report.

Acronyms	**Definition**
RL	Reinforcement Learning
FF	First Fit
BF	Best Fit
ILP	Integer Linear Programming
AI	Artificial Intelligence
CNN	Convolutional Neural Networks
PPO	Proximal Policy Optimization
TRPO	Trust Region Policy Optimization

Table 1: List of abbreviations used in the report

Chapter 1

Overview

1.1 Problem introduction

In the glass manufacturing industry, cutting large glass sheets (glass blanks) into smaller sizes according to customer requirements is an important but challenging process. Glass factories and glass processing plants often face the problem of material waste when optimizing the layout of cutting lines.

The amount of excess glass after cutting not only increases the cost of raw materials but also causes difficulties in warehouse management and waste disposal. In addition, the use of too much glass blanks can lead to higher production costs, reducing competitiveness in the market. Therefore, optimizing glass cutting to reduce the amount of glass blanks used and reduce the amount of excess glass not only helps save costs but also contributes to environmental protection, reduces the volume of industrial waste, and applies technology to manufacturing industries.

With the rapid development of artificial intelligence (AI), the application of these technologies in the manufacturing sector is opening up many new opportunities. One of the issues of interest is the cutting stock problem. The two-dimensional cutting stock problem is a fundamental optimization challenge encountered in many different manufacturing industries, where large sheets of raw material must be efficiently cut into smaller pieces as required while minimizing waste. This problem is of paramount importance in areas such as metal fabrication, woodworking, glass manufacturing, and textile manufacturing, where resource optimization has a direct impact on cost efficiency and sustainability.

1.2 Reinforcement Learning

Reinforcement learning (RL) is a subfield of machine learning that focuses on training an agent to interact with a dynamic environment. The agent learns to perform actions that maximize its cumulative reward through trial-and-error interactions. Unlike supervised learning, RL does not require explicit data labeling or immediate correction of incorrect behavior. Instead, it relies on feedback in the form of rewards or punishments to guide the agent toward optimal behavior. One advantage of RL is its ability to adapt to complex, high-dimensional problems that traditional methods struggle with. In the context of the Cutting-Stock Problem, RL offers the potential to create dynamic cutting patterns by learning from past performance, which can lead to innovative and efficient new solutions.

1.3 Target

Our team's goals in this project are:

1) Formalize the Cut-and-Stock Problem as an ILP, providing a clear mathematical representation that clearly states the constraints and objectives of the problem.

2) Analyze, discuss, and implement different approaches to solving the problem, including traditional optimization techniques, search methods, and Reinforcement Learning-based approaches. This perspective allows us to compare the strengths and weaknesses of both approaches in solving this challenging problem.

3) Propose potential extensions to our model and explore its applications in real-life situations and needs.

First, we need to precisely define the direction and core purpose of the problem. To formulate this problem as an integer linear programming (ILP) model, we define the following decision variables: Let x_{ij} be a binary decision variable whose value is: if sample j is used to cut x_{ij} of material i, otherwise $x_{ij} = 0$. Let y_j be the number of times sample j is used.

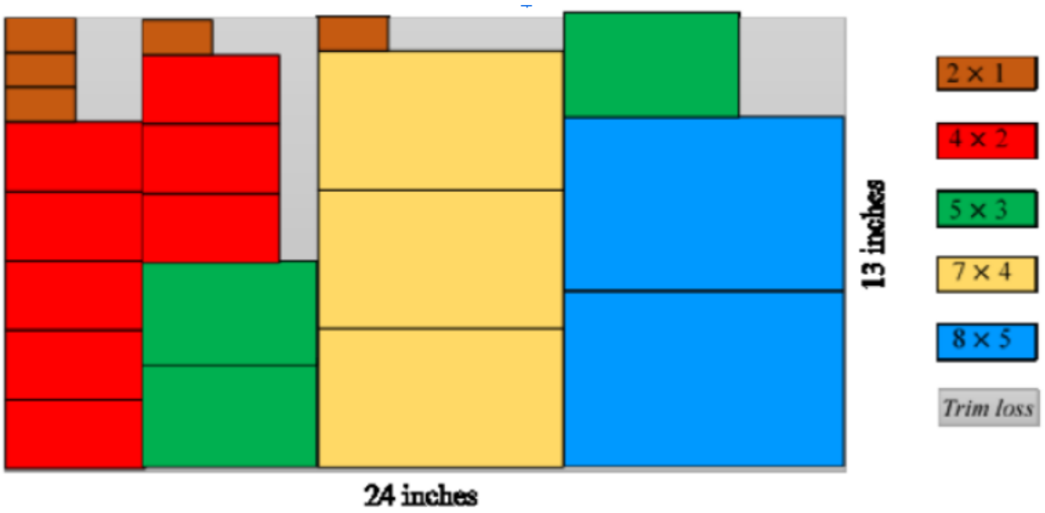


Figure 1.1: 2D material cutting illustration

The objective is to minimize the total amount of waste material generated during cutting. If C denotes the total available material and W_j denotes the waste associated with sample j, the objective function can be defined as:

$$Minimize\ Z = \sum_j W_j y_j$$

Additionally, if the cost per unit of waste is considered, the objective function becomes:

$$Minimize\ Z = \sum_j c_j y_j$$

Where c_j represents the cost incurred when using sample j.

1.4 Research limitations

The Cutting-Stock problem is often formulated as an Integer Linear Programming (ILP) problem. Various techniques have been proposed to solve this problem, including branch-and-bound algorithms, heuristics, and column generation methods with trade-offs between computational complexity and solution quality. For simplicity, this large exercise focuses on only one particular variant of the problem, namely the 2D Material Cutting Problem variant where each item is a 2D non-rotating rectangle with all integer dimensions.

1.5 Constraints

To ensure feasibility and optimize production efficiency, we set a series of strict constraints before conducting research and proposing solutions to the Cutting Material Problem. These constraints not only help to systematically guide the analysis process but also ensure that the proposed solutions can be practically applied in an industrial production environment.

In addition to considering theoretical factors, we also pay attention to practical conditions such as raw material availability, technical constraints during cutting, as well as cost optimization and loss minimization. Establishing reasonable constraints not only improves the feasibility of the solution, but also improves material utilization efficiency, minimizes waste, and optimizes production time.

Therefore, before we delve into algorithms and problem-solving methods, we take the time to carefully evaluate the factors that affect the actual process. This helps ensure that any proposed solution is highly applicable, fits the actual requirements of the business, and delivers optimal efficiency in production.

1.6 Contents

Our 2D cutting stock project report will consist of 5 chapters:

- Chapter 1: Overview
- Chapter 2: Theoretical basis
- Chapter 3: Implementation of solutions
- Chapter 4: Conclusion
- Chapter 5: Development direction.

Chapter 2

Theoretical basis

2.1 Define reinforcement learning

Reinforcement learning is a branch of machine learning that studies how an agent in an environment in a state performs an action to optimize a common reward.

A reinforcement learning problem is often modeled as a Markov decision process. MDPs are still considered the standard for modeling reinforcement learning problems. In addition, reinforcement learning algorithms are also closely related to *dynamic programming* techniques.

We consider the problem as follows:

At time t, agent $\mathscr{A}$ is in state $s_t \in \mathscr{S}$, performs action $a \in \mathscr{A}$ and has a probability $P_{s_a}(s_{t+1})$ of moving to the next state s_{t+1} and receiving an instantaneous reward r_t. P_{s_a} is a distribution representing the agent's next state when performing action $a \in \mathscr{A}$ at state $s \in \mathscr{S}$. The goal of the problem is to maximize the overall reward (return) $\mathscr{R}$.

2.2 Dataset

2.2.1 Input

We created a dataset of 10 batches, each with a list of raw material stocks and a list of products in order. We stored them in a *.csv* file to use to test the performance of all methods

Raw glass sheet dimensions: 2000 mm × 3000 mm, 1000 mm × 1000 mm,...
Ordered panes: Different sizes (e.g., 500 mm × 700 mm, 600 mm × 800 mm, etc.)
Demand quantities for each size.

Table 2.1: Input data

Optimal cutting options for dividing the raw material sheet into required pieces.
Each option shows the location and arrangement of each sub-glass sheet on the original material.
Total number of raw panels required to fulfill the entire order.
Total area of excess material that cannot be used after cutting.

Table 2.2: Output data

under the same specific conditions.

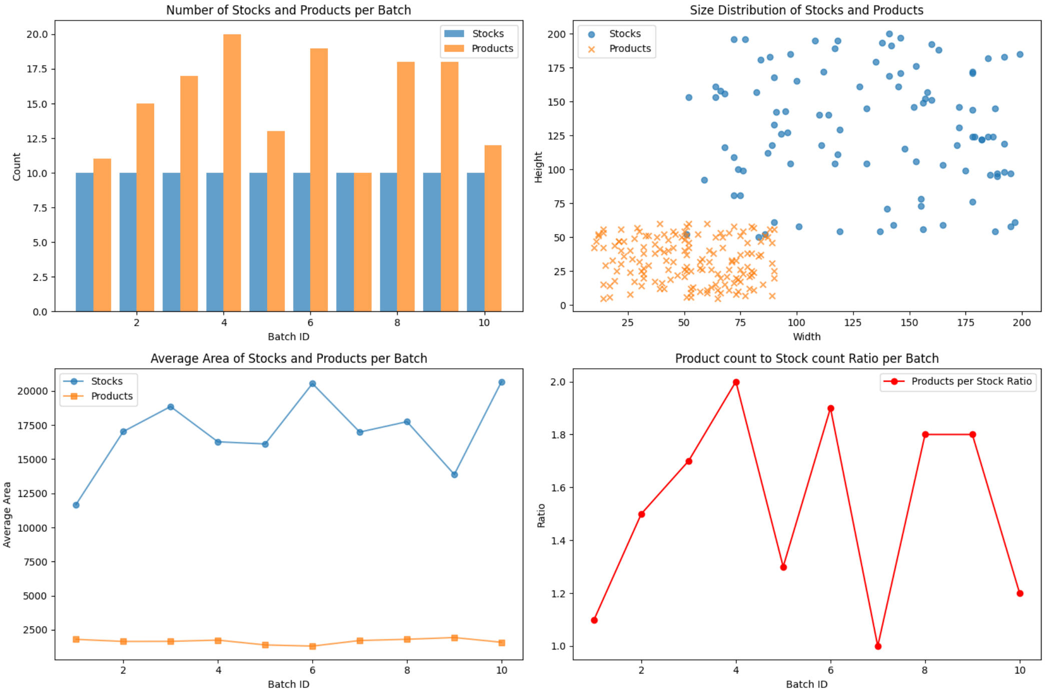


Figure 2.1: Visual illustration of data

2.2.2 Output

- Data format output

2.3 Constraints

2.3.1 Demand Fulfillment Constraint:

This constraint ensures that the number of cut pieces meets or exceeds the required demand for each item. It prevents underproduction while allowing for potential excess that may be utilized elsewhere.

$$\sum_j a_{ij} y_j \geq d_i \quad \forall j$$

Where:

a_{ij} represents the number of times item i appears in pattern j.

d_i denotes the demand for item i.

2.3.2 Material Utilization Constraint:

The total area occupied by the cut pieces within each selected pattern must not exceed the dimensions of the raw stock material. This constraint prevents infeasible cutting arrangements where pieces exceed the available space.

$$\sum_i w_i h_i x_{ij} \leq WH$$

Where:

w_i, h_i represent the width and height of item i.

H, W denote the overall dimensions of the stock material.

2.3.3 Non-Overlapping Constraint:

To prevent overlap between cut pieces within a given pattern, we enforce spatial feasibility by ensuring that no two items within the same cutting pattern share overlapping regions. This constraint is typically modeled using no-overlap conditions in combinatorial optimization or geometric constraints.

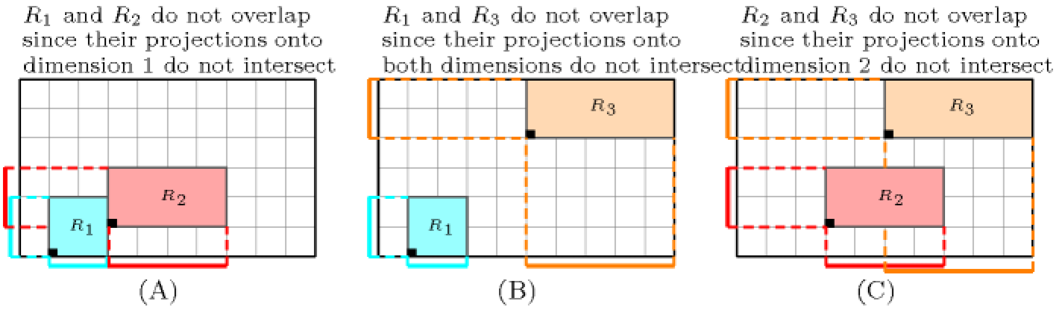

R_1 and R_2 do not overlap since their projections onto dimension 1 do not intersect
R_1 and R_3 do not overlap since their projections onto both dimensions do not intersect
R_2 and R_3 do not overlap since their projections onto dimension 2 do not intersect
R_1
R_2
R_3
(A)
(B)
(C)

Figure 2.2: Global Constraint Catalog (Reference)

2.3.4 Binary and Non-Negativity Constraints:

The binary nature of the decision variables must be maintained, ensuring logical selection of cutting patterns.

$$x_{ij} \in \{0,1\}, \quad y_j > 0, \quad \forall i,j$$

The non-negativity constraint ensures that the solution remains within practical and implementable limits, preventing negative material usage

2.3.5 Seamless and starting from the edge constraints:

Seamless: The cut pieces must be arranged so that there are no unnecessary gaps between them. This helps to optimize the use of materials, avoiding waste of excess parts that cannot be reused.

Start from the edge: The first cuts should start from the edge of the sheet of material, rather than scattered or starting in the middle. This makes the cutting process easier to perform, especially when using an automatic cutter.

If we assume the material plate has dimensions W×H and each cut piece has coordinates (x_i, y_i) with dimensions (w_i, h_i), then the constraint can be modeled as follows:

$$x_i = 0 \quad \text{or} \quad x_i = x_j + w_j$$

$$y_i = 0 \quad \text{or} \quad y_i = y_j + h_j$$

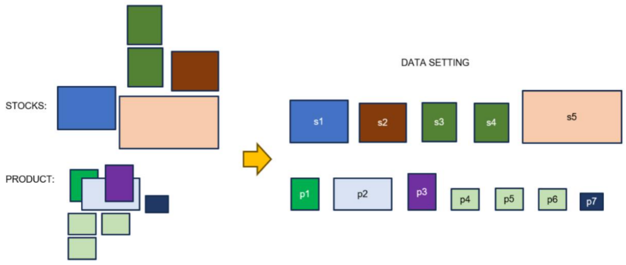


Figure 2.3: Setting product and stock

This ensures that at least one edge of each cut piece contacts the edge or another cut piece.

No large gaps between pieces:

$$x_{i+1} = x_i + w_i \quad \text{or} \quad y_{i+1} = y_i + h_i$$

2.3.6 Orientation Constraint:

Products with the same dimensions but different orientations are treated as separate types when cutting.

This ensures that even if a piece measuring 4x2 is equivalent to one measuring 2x4 when rotated, they are considered distinct and must be cut according to predefined orientations.

2.3.7 Data setting:

Stocks and products are numbered sequentially starting from 1 (index is 0 in the code) regardless of their similar size.

2.4 Evaluation Metrics

2.4.1 Runtime (s)

- Measures the execution time for each policy.
- A lower runtime indicates a faster solution, which is important in real-world production scenarios.

2.4.2 Total Trim Loss

- Represents the total unused stock material after cutting.
- Computed as:

$$\text{Total Trim Loss} = \sum_{s \in \text{Used Stocks}} \text{Unused Area in } s$$

A lower value indicates better material utilization.

2.4.3 Remaining Stocks

- The number of stock pieces that were not used during the process.
- A higher number indicates that fewer stocks were required to fulfill the demand, which is desirable.

2.4.4 Used Stocks

- The number of stock pieces actually used.
- A lower number is preferred as it indicates a more efficient allocation of stock.

2.4.5 Avg Used Stock Area

- The average area of stocks that were used in the cutting process.
- Computed as:

$$\text{Avg Used Stock Area} = \frac{\sum_{s \in \text{Used Stocks}} \text{Total Usable Area of } s}{\text{Used Stocks}}$$

Chapter 3

The solution

3.1 Heuristic Algorithm

3.1.1 Solution Approach

Define cutting patterns using integer programming.

$$Minimize\ Z = \sum_j W_j y_j$$

Where W_j is the waste associated with pattern j.

Use heuristic approach with First Fit, Best Fit algorithms to solve the problem with lower computational cost.

3.1.1.1 Implement

The Combination Algorithm is a heuristic approach designed to enhance both computational efficiency and material utilization in solving the two-dimensional cutting-stock problem. This method integrates the advantages of the First-Fit and Best-Fit algorithms to achieve an optimal balance between execution speed and minimized waste.

Key Motivation

	First Fit Algorithm (FF Algorithm)	**Best Fit Algorithm (BF Algorithm)**
Idea	Place each piece into the first available space that fits.	Place each piece in the space that minimizes the remaining unused area.
Steps	• Sort the ordered panes by size. • Try placing each pane in the first available cutting space. • If no space is available, start a new stock sheet.	• Sort the ordered panes by size. • Check all available spaces and choose the one that leaves the smallest leftover area. • If no space is available, use a new stock sheet.
Advantages	Fast and easy to implement.	Reduces waste more effectively than First Fit.
Disadvantages	May not always produce the most efficient cutting pattern.	Requires more computation compared to First Fit.

Table 3.1: Comparison of First Fit and Best Fit Algorithms

- The First-Fit algorithm is fast and straightforward but may leave significant unused space on stock sheets, leading to higher material waste.
- The Best-Fit algorithm is more effective in reducing waste by strategically placing pieces but requires more processing power and time.

To address these trade-offs, the Combination Algorithm employs a two-step strategy:

- Fast Initial Placement – Uses the First-Fit approach to quickly allocate products to stock sheets.
- Optimized Refinement – Applies the Best-Fit technique afterward to rearrange placements, further reducing material waste.

3.1.2 Inference

First Fit Algorithm

***Function First_Fit_2DCSP**:*

```
Push one of each stock_type into the stock_list
For each stock in stock_list (sorted from largest to smallest area):
    For each prod in product_list (sorted from largest to smallest area):
        If prod can fit into stock:
            Place prod into stock
        Else:
            Let stockType be the first stock_type that fits prod
            Create a new stock of type stockType
            Place prod into the new stock
            If the demand for prod is met:
                Remove prod from the item list
```

Best Fit Algorithm

Function Best_Fit_2DCSP:

```
Sort product_list from largest to smallest area
For each product in product_list:
    best_stock ← None
    min_remaining_area ← ∞
    For each stock in stock_list:
        If product can fit in stock:
            remaining_area ← compute remaining area after placing product
            If remaining_area < min_remaining_area:
                min_remaining_area ← remaining_area
                best_stock ← stock
    If best_stock is not None:
        Place product into best_stock
    Else:
        Create a new stock with dimensions ($W_{new}$, $H_{new}$)
        Place product into the new stock
        Add new stock to stock_list
    Remove product from product_list once placed
Return stock_list
```

Combination Algorithm

Function Combination_2DCSP:

```
For each stock in stock_list (sorted from largest to smallest area):
    For each prod in product_list (sorted from largest to smallest area):
        For each (x, y) position in stock where prod can be placed:
            If prod can be cut at (x, y):
                Compute $S_{ij}$ as the area of the smallest rectangle containing the cut area
        Find $(x_0, y_0)$ where $S_{ij}$ is minimized
        Cut prod from stock at position $(x_0, y_0)$
For each stock in cut_stock_list (sorted from smallest to largest area):
    For each stock_i in uncut_stock_list (sorted from smallest to largest area):
        If cut area of stock_i is smaller than stock_j:
            Move cut set from stock_i to stock_j
```

Reference: Heuristic Algorithm

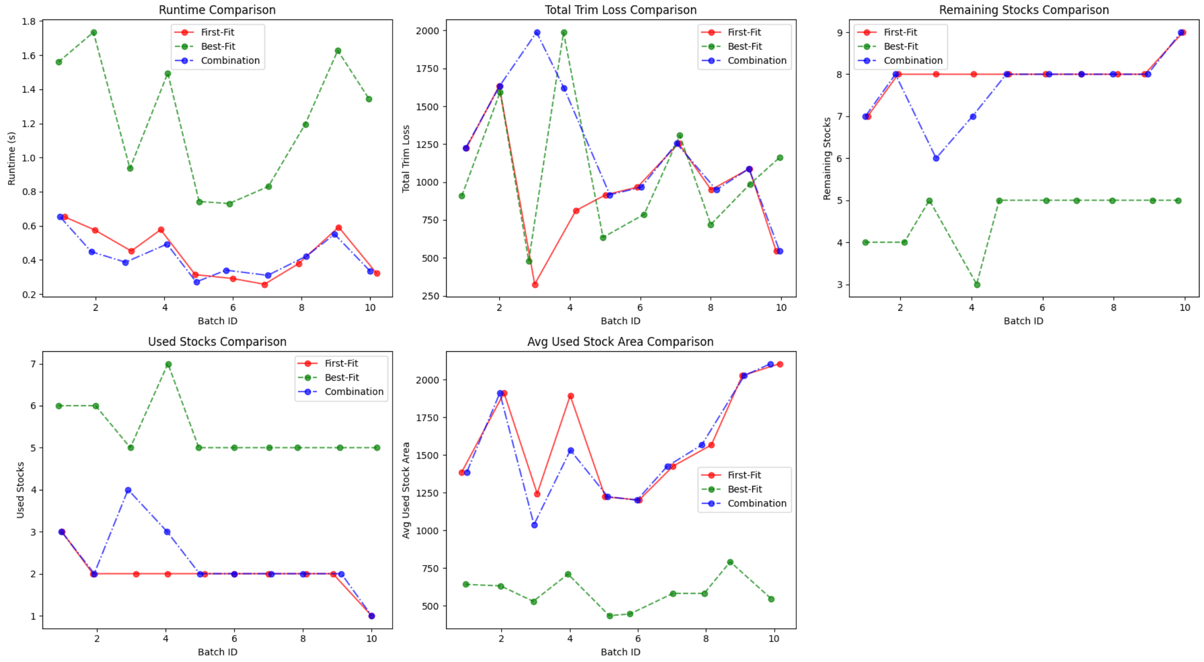


Figure 3.1: Comparison First Fit, Best Fit and Combine

- **Best-Fit** is the most efficient in minimizing trim loss but comes at the cost of significantly higher runtime and more stock usage.
- **First-Fit** and **Combination** are computationally efficient but may lead to slightly higher trim loss.
- **Combination** performs similarly to **First-Fit**, with some variations in total trim loss and used stock area.

3.2 Reinforcement Learning Algorithm

3.2.1 Q-learning Method

3.2.1.1 Overview about Q-learning and Q-tables

Before addressing the Cutting Stock environment, we provide a concise introduction to Q-Learning and the concept of a Q-Table.

3.2.1.1.1 What is a Q-Tabel?

- A Q-Table is a matrix (or 2D array) in which each row corresponds to a possible state s and each column corresponds to a possible action α.
- The entry $Q(s,\alpha)$ represents the expected future reward when the agent takes action α in state s then continues to follow the current policy or a greedy policy thereafter.
- In tabular Q-Learning, we store and update each $Q(s,\alpha)$ as we gain experience from interacting with the environment. This is especially useful in discrete or semi-discrete problems, although it can become large for high-dimensional tasks.

3.2.1.1.2 Q-learning Algorithm

- Q-Learning is an off-policy Reinforcement Learning method introduced by Watkins & Dayan (1992). Its defining characteristic is the update rule.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max Q(s',a') - Q(s,a)\right]$$

  - s is the **current state**.
  - a is the **action** taken.
  - r is the **immediate reward** received after taking action a in state s.
  - s' is the **next state**.
  - a' is a **possible action** in the next state.
  - $\alpha(0 \leq \alpha \leq 1)$ is the **learning rate**, controlling how much new information overrides old information.
  - $\gamma(0 \leq \gamma \leq 1)$ is the **discount factor**, determining how much future rewards are valued compared to immediate rewards.

- Over time, through repeated interactions (episodes) in the environment, the Q-Table converges to values that approximate the optimal Q-function—if each state-action pair is visited infinitely often and if certain learning conditions are met.

3.2.1.1.3 Action Selection: Epsilon-Greedy Strategy

- An essential component of Q-Learning is balancing **exploration** and **exploitation**:
  - With probability ε, the agent picks a random action (exploration).
  - Otherwise, it picks the action $\arg\max Q(s,a)$ (exploitation), based on the current Q-Table.
- During training, ε typically decays from a high value (encouraging exploration) to a lower value (favoring exploitation) over many episodes.

3.2.1.2 Q-learning Agent

3.2.1.2.1 State Encoding

- Directly encoding the entire 2D layout in a Q-Table can be prohibitively large.
- As a simplification, we compute two features from the environment's observation:
  - `empty_space`: the total count of free cells (-1).
  - `remaining_products`: the sum of quantities of all uncut products.
- We combine them into a single integer state index:

$$state = (\texttt{empty\_space} \times 1000 + \texttt{remaining\_products}) \mod state\_size$$

Although this approach may lose spatial information, it keeps the Q-Table manageable in size.

3.2.1.2.2 Action Selection

- We maintain an integer action index from 0 up to `action_size - 1`.
- In `get_env_action(action, observation)`, we break this index down into which product to cut and where to place it on a chosen stock.
- If no valid placement is found (the stock is fully or incorrectly used), the agent returns a dummy action.

3.2.1.2.3 Q-update

- After receiving *a* reward r and transitioning to a new state s', the agent updates:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max Q(s',a') - Q(s,a) \right]$$

- We also include the epsilon-decay to gradually shift from exploration to exploitation.

3.2.1.3 Overview of Reward and Penalty in the Project

In your implementation, the overall reward an agent receives at each step (or when querying `get_reward(...)`) is composed of three main parts:

- **Filled Ratio**: Represents how much of the used stock area has been effectively occupied by products.
- **Trim Loss**: Measures how much space remains unused in each stock that has been at least partially cut.
- **Bonus for Unused Stocks**: Provides an extra positive incentive for any stock sheets that remain completely untouched, thus potentially reducing overall material consumption.

Additionally, there is a default environment reward that returns 1 if cutting is completely finished (all products are allocated) and 0 otherwise. However, you incorporate a custom reward function that combines filled ratio, trim loss, and unused stock bonus. This custom reward can override or supplement the environment's default reward, ensuring the agent gets more fine-grained feedback on how effectively it is cutting the materials.

3.2.1.4 How Rewards Are Computed

3.2.1.4.1 Filled Ratio and Trim Loss

3.2.1.4.1.1 Filled Ratio

- Calculated as the proportion of space occupied by cut products in the stocks that the agent has actually used.

- A higher filled ratio implies more efficient packing or cutting, so it contributes positively to the reward.

3.2.1.4.1.2 Trim Loss

- Defined as the ratio of unused (but usable) area in any stock that has been partially cut.
- Trim loss is effectively subtracted (or penalized) from the reward because large unused spaces indicate poor cutting strategies.

In my code, the environment tracks these two quantities (filled_ratio and trim_loss) and stores them in the "info" dictionary after each step. When the training script calls your reward function, those values are retrieved to guide the reward calculation.

3.2.1.4.2 Bonus for Unused Stocks

- After each step, the code checks how many stock sheets have not been touched at all—meaning they remain fully intact and free of any cuts
- The assumption is that using fewer stock sheets, while still meeting product demand, can be beneficial.
- The code then calculates the fraction of completely unused stocks:

$$\frac{num\_stocks\_unused}{total\_stocks}$$

- This fraction is multiplied by a small factor (often referred to as lambda_bonus or similar in your code) to create a bonus term.
- This bonus is added to the reward, thus encouraging the agent to concentrate its cuts on as few sheets as possible.

3.2.1.4.3 Combining the Components

- These parts are brought together with an expression along the lines of:

$$Reward = (FilledRatio - TrimLoss) + \left(BonusFactor + \frac{UnusedStocks}{TotalStocks}\right)$$

3.2.1.4.3.1 Positive Contribution

- FilledRatio (encourages efficient usage).

- Bonus for Unused Stocks (incentivizes minimizing the total stock used).

3.2.1.4.3.2 Negative Contribution

- TrimLoss (penalizes wasted space).

  The result is that if the agent manages to cut products in a way that:

- Occupies a high proportion of space in the sheets it actually uses

- Keeps some sheets completely untouched

- Minimizes leftover or wasted space on used sheets It will receive **higher overall rewards**.

3.2.1.4.4 Additional Penalties or Edge Cases

Beyond these main reward terms, your project also has a couple of implicit or potential penalty mechanisms.

3.2.1.4.4.1 Zero Reward During Ongoing Steps

- If the environment is not yet "done" and you have not overridden the default reward for each intermediate step, the agent might receive 0 from the environment. This effectively penalizes the agent indirectly for lengthy, unproductive moves if it does not improve the state meaningfully.

3.2.1.4.4.2 Invalid Cutting Actions (If Implemented)

- If the agent attempts to cut a product in an invalid position—outside boundaries or overlapping existing cuts—the environment typically ignores or fails the action. While your core code does not impose a direct numeric penalty for invalid moves, such a penalty can be added if desired. The agent would then learn more quickly to avoid pointless or impossible cuts.

3.2.1.4.5 Impact on Agent Behavior

- Because the Filled Ratio and Trim Loss are directly opposed in the reward formula, and because leaving entire stocks unused provides an additional bonus, the agent learns to:
  - Fill each used sheet with products as densely as possible.
  - Avoid spreading cuts across multiple sheets unnecessarily.
  - Minimize leftover space on any partially used sheet.
- When these objectives align well with real manufacturing goals (reducing total sheets consumed and waste), the reward function drives more efficient cutting strategies.

3.2.1.5 Q-learning performance

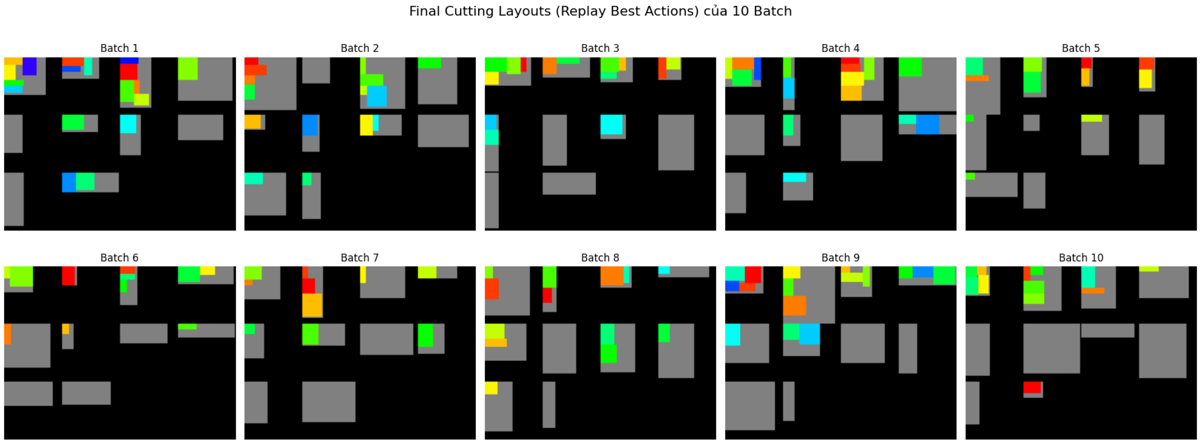


Figure 3.2: Q-learning 10 batches

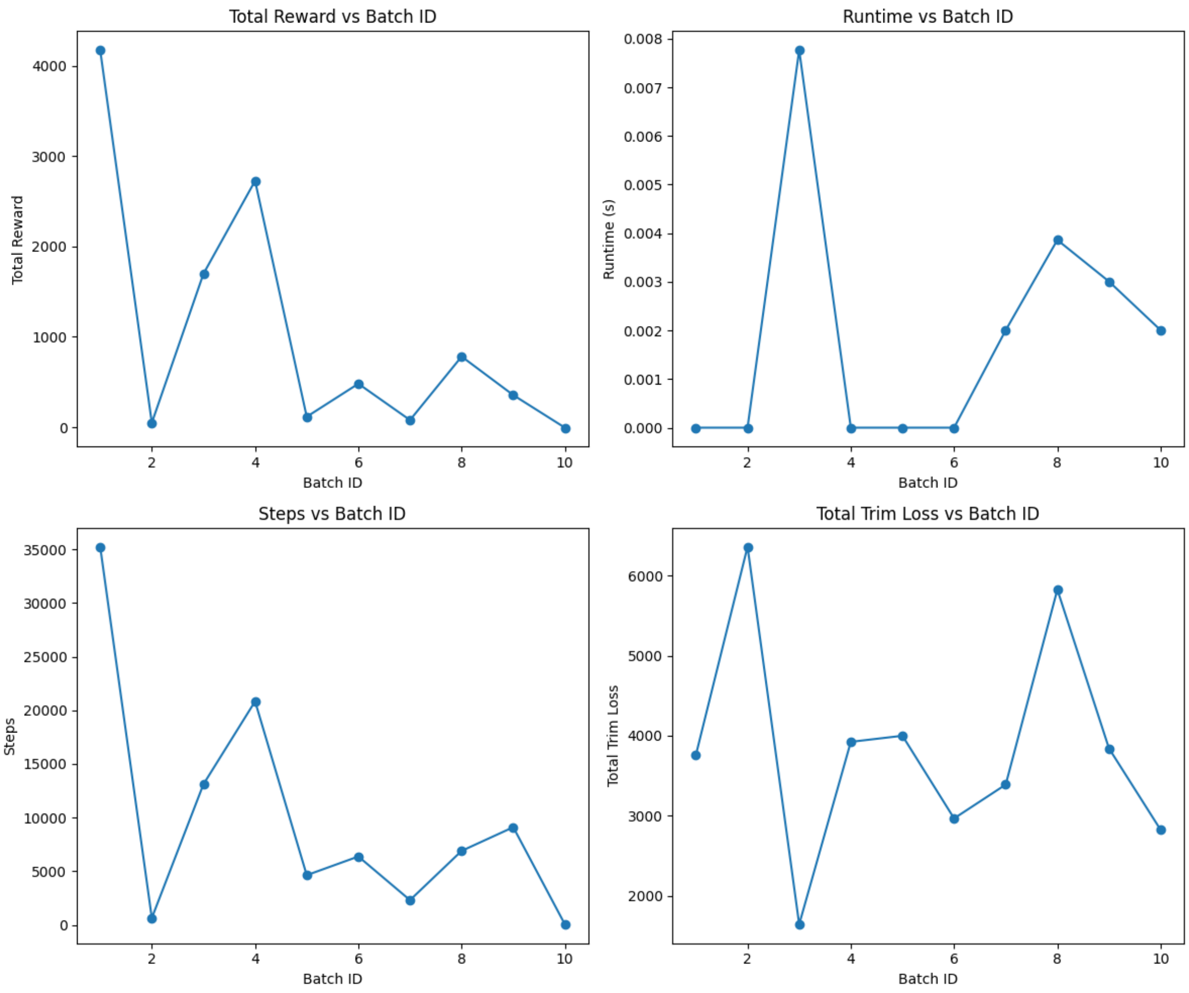


Figure 3.3: Q-learning Performance

3.2.2 Proximal Policy Optimization (PPO)

3.2.2.1 Overview about Proximal Policy Optimization

- Proximal Policy Optimization (PPO) is a reinforcement learning (RL) algorithm belonging to the policy-based methods group. It was proposed by OpenAI in 2017 to address the drawbacks of previous policy optimization algorithms such as Trust Region Policy Optimization (TRPO) and Actor-Critic.

- PPO is particularly effective in optimizing policies by ensuring that policy updates are not excessively large in each step. This makes it more stable compared to traditional policy gradient methods.

3.2.2.2 How Proximal Policy Optimization works?

3.2.2.2.1 Key algorithm elements in PPO

3.2.2.2.1.1 Probability Ratio

- Given a policy π_θ parameterized by θ, and an old policy $\pi_{\theta_{old}}$, we define the probability ratio as:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

- This ratio represents the change between the new and old policies for action a_t at state s_t.
- In that:
  - $\pi_\theta(a_t|s_t)$: probability of choosing action a_t at state s_t under the new policy.
  - $\pi_{\theta_{old}}(a_t|s_t)$: probability of choosing action a_t at state s_t under the old policy.

3.2.2.2.1.2 Advantage Estimation in PPO

- In Reinforcement Learning, the Advantage Function (A_t) helps measure how much better or worse an action is compared to what is expected. Specifically, A_t represents the difference between the value of a particular action $Q(s_t, a_t)$ and the average value of all possible actions at that state $V(s_t)$:

$$A_t = Q(s_t, a_t) - V(s_t)$$

- However, the exact calculation of $Q(s_t, a_t)$ is very difficult due to the uncertainty and large fluctuations of the rewards in RL. Therefore, PPO uses a method called **Generalized Advantage Estimation (GAE)** to get a more stable estimate.
- **Generalized Advantage Estimation (GAE)**
  - GAE helps to calculate A_t stably and accurately by using the actual reward r_t and the estimated value of the state $V(s_t)$. The GAE formula is as follows:

$$A_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+1}$$

- With:
  - γ (discount factor): controls the impact of future rewards.
  - λ (GAE parameter): helps balance bias and variance in estimation.
  - δ_t (TD error) is calculated by:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

- When λ is close to 0, we have an estimate with high bias but low variance. When λ is close to 1, we have low bias but high variance. In practice, people often choose λ around 0.95 to get the best balance.

3.2.2.2.1.3 PPO Objective Function (Clipped Surrogate Loss)

- PPO optimizes the policy by updating the old policy based on the estimated advantage A_t.
- However, if the policy change is too large, it can cause instability. Therefore, PPO limits the change by using the **Clipped Surrogate Loss**:

$$L^{CLIP}(\theta) = \mathbb{E}\Big[\Big(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t\Big)\Big]$$

- Here:
  - $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio between the new and old policies.
  - ε (usually 0.1 to 0.3) helps limit the update rate to avoid too large a change.
- This objective function helps ensure that **the new policy does not deviate too far from the old policy**, avoiding destabilizing the training process.

3.2.2.2.1.4 Complete Loss Function in PPO (optional)

PPO not only optimizes the policy, but also updates the Value Function and uses Entropy Bonus to maintain diversity in action. In summary, the loss function of PPO consists of three components:

$$L(\theta) = L^{CLIP}(\theta) - c_1 L^{VF}(\theta) + c_2 H(\pi_\theta)$$

In that:

- $L^{VF}(\theta) = (V_\theta(s_t) - V_{target}(s_t))^2$ helps update the value function.
- $H(\pi_\theta)$ is the entropy bonus, which helps avoid overfitting to a particular strategy.
- c_1, c_2 is a weighting factor that adjusts the priority between components.

3.2.2.2.2 Structure of 3 networks in PPO: CNN, Actor, and Critic.

In PPO, there are usually three important components in the Neural Network:

- **Convolutional Neural Network (CNN)** - extract features from the input (usually used if the input is an image or matrix).
- **Actor Network** - determine the probability distribution of actions to make optimal decisions.
- **Critic Network** - estimate the value of the state, support the calculation of Advantage Estimation.

3.2.2.2.2.1 Convolutional Neural Network (CNN) **Purpose:**

- If the input is an image or a matrix, CNN helps extract features from the input data before feeding it into the Actor-Critic network.
- If the input is not an image (state vector), CNN can be replaced by MLP (Multi-Layer Perceptron).

Common structure of CNN in PPO:

- Conv2D layers: Extract features from state matrix.
- Batch Normalization: Stabilize the training process.
- ReLU Activation: Apply non-linearity.
- Flatten: Convert matrix to feature vector.

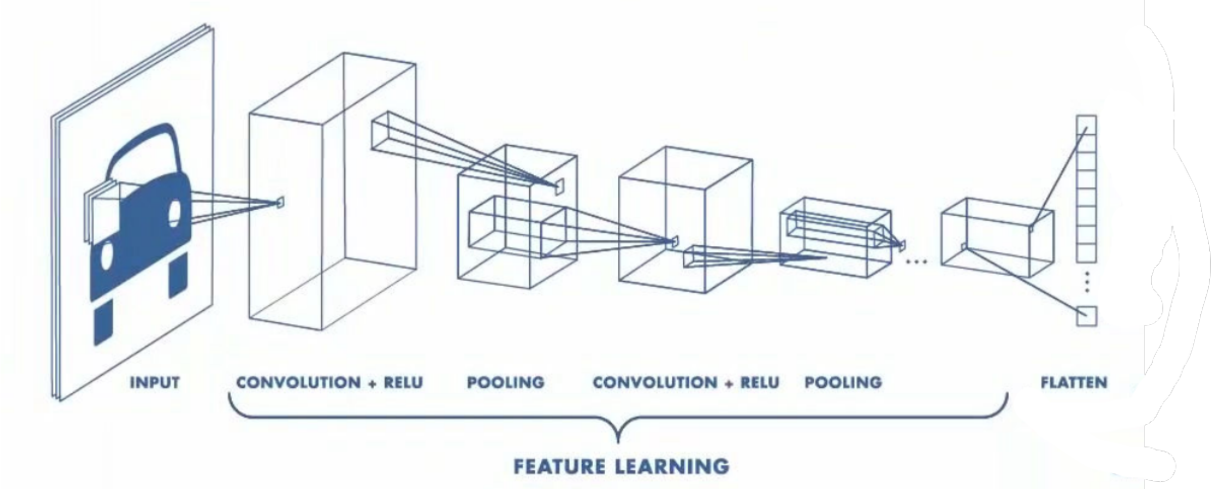


Figure 3.4: Structure of CNN (reference)

3.2.2.2.2.2 Actor Network - Action Policy

- **Purpose:**
  - Take input from CNN Feature Extractor or state vector.
  - Determine the probability distribution of actions $\pi_\theta(s)$.
- **Common structure of Actor Network:**
  - Linear layers: Predict the probability distribution for each action.
  - Softmax or Categorical distribution: To get the action probability.

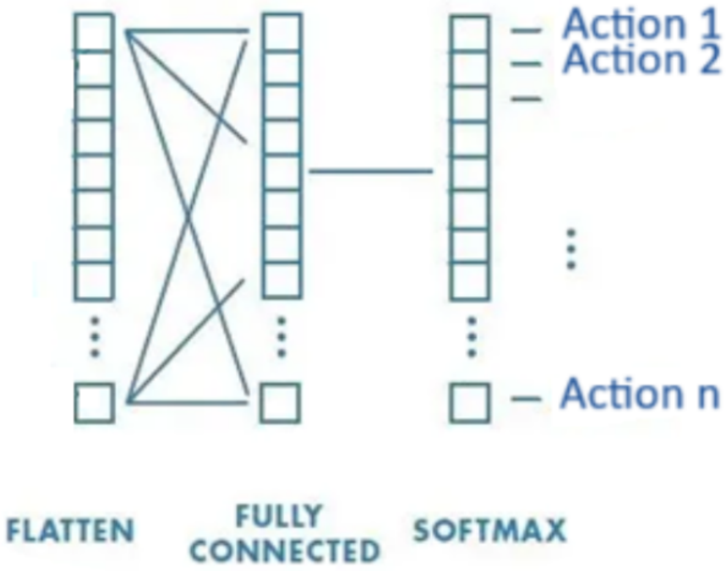


Figure 3.5: Predict the probability distribution for each action. (reference)

- PPO uses **Clipped Surrogate Loss** to optimize **Actor**: *(Mentioned above)*

$$L^{CLIP}(\theta) = \mathbb{E}\left[(r_t(\theta)A_t, \text{clip}\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon\right)A_t)\right]$$

  - **Gradient of the Actor loss function in PPO:**

$$\nabla_\theta L^{CLIP}(\theta) = \nabla_\theta\left(r_t(\theta)A_t, \text{clip}\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon\right)A_t\right)$$

  - **Gradient of First Term (Surrogate Loss Root):**
    - ∗ First, we compute the gradient of the policy update ratio:

$$\nabla r_t(\theta) = \frac{\nabla \pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

    - ∗ When multiplied by A_t, the gradient of the Surrogate Loss Root is:

$$\nabla_\theta\left(r_t(\theta)A_t\right) = A_t \cdot \nabla_\theta r_t(\theta)$$

      - · If $A_t > 0$: The gradient will increase the probability of choosing action.

· If $A_t < 0$: The gradient will decrease the probability of choosing action.

- **Gradient of Second Term (Clipped Loss):**

  ∗ Clipped Loss has the form:

$$\text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t$$

  · If $r_t(\theta)$ lies in the interval $[1-\varepsilon, 1+\varepsilon]$, then the gradient is the same as the original term.

  · If $r_t(\theta)$ goes beyond this interval $[1-\varepsilon, 1+\varepsilon]$, then the gradient will update according to the outermost allowable amplitude, $1-\varepsilon$ or $1+\varepsilon$.

- **Gradient formula of Clipped Loss:**

$$\nabla clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t = \begin{cases} \nabla_\theta(r_t(\theta)A_t) & \text{if } r_t(\theta)A_t < clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t \\ (1-\varepsilon) & \text{if } r_t(\theta)A_t < 1-\varepsilon \\ (1+\varepsilon) & \text{if } r_t(\theta)A_t > 1+\varepsilon \end{cases} \tag{3.1}$$

- **So that, Clipped Surrogate Loss Gradient Synthesis:**

$$\nabla_\theta L^{CLIP}(\theta) = \begin{cases} \nabla_\theta(r_t(\theta)A_t) & \text{if } r_t(\theta)A_t < clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t \\ (1-\varepsilon) & \text{if } r_t(\theta)A_t < 1-\varepsilon \\ (1+\varepsilon) & \text{if } r_t(\theta)A_t > 1+\varepsilon \end{cases} \tag{3.2}$$

3.2.2.2.2.3 Critic Network – State Value Function V(s) Purpose:

- Take input from CNN Feature Extractor or state vector.

- Evaluate the value of state $V(s)$.

- Support for Advantage Estimation in PPO:

$$A_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+1} \tag{3.3}$$

Common structure of Actor Network:

- Takes input from CNN Feature Extractor or state vector.

- Linear layers: Predicts state value.

- Output is usually a single value $V(s)$.

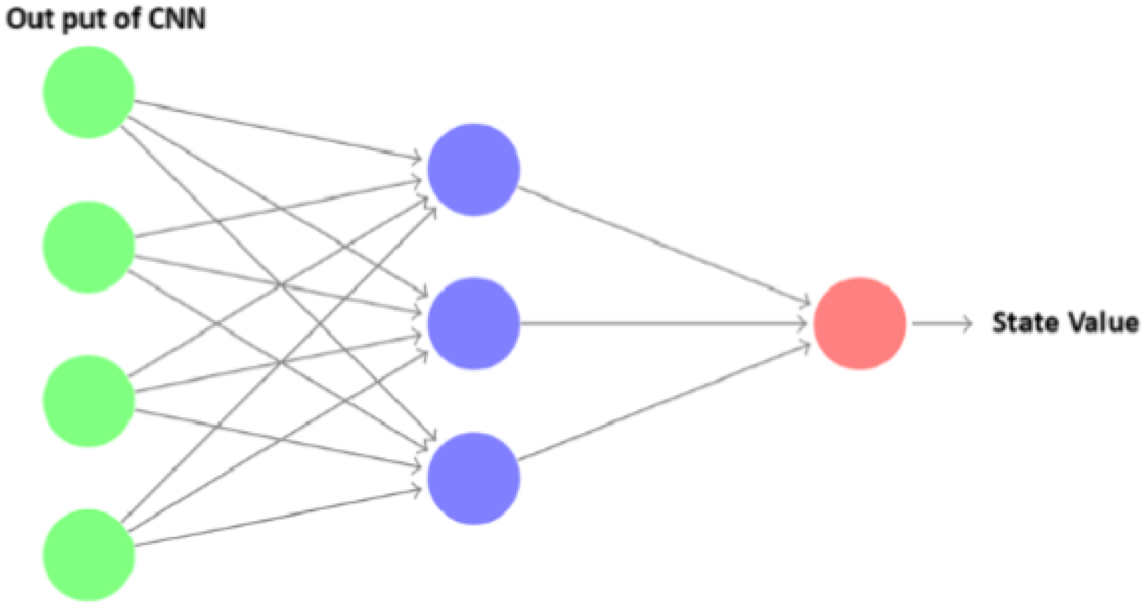


Figure 3.6: Output of CNN reference

- **Use MSE Loss to train Critic:**
  * Critic is trained to estimate the value of the state, with the goal of reducing the error between the estimated value $V(s)$ and the **target value** $V_{target}(s)$:

$$L^{Critic}(\theta) = \frac{1}{N}\sum_{t=0}^{N} (V_\theta(s_t) - V_{target}(s_t))^2 \tag{3.4}$$

- In that:
  - $V_\theta(s_t)$ is the estimated state value from Critic Network.
  - $V_{target}(s)$ is the target value, usually expressed by **Monte Carlo**:

$$V_{target}(s) = R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t} r_T \tag{3.5}$$

- Or using **Generalized Advantage Estimation**: *(mentioned above)*

$$V_{target}(s) = V(s_t) + A_t \tag{3.6}$$

– Gradient of MSE Loss:

- The MSE function has the form:

$$L(\theta) = \frac{1}{2}\left(V_\theta(s_t) - V_{target}(s_t)\right)^2 \tag{3.7}$$

- Derivative with respect to θ:

$$\nabla L(\theta) = (V_\theta(s) - V_{target}(s)) \cdot \nabla_\theta V_\theta(s) \tag{3.8}$$

- Use Gradient Descent to update Critic's weights:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta) \tag{3.9}$$

- Mean:
  - Error between estimated value and actual value $(V_\theta(s) - V_{target}(s))$ is the part that needs to be minimized.
  - Critic Network Gradient $(\nabla_\theta V_\theta(s))$ helps update network weights.
  - If $V_\theta(s)$ is larger than the target value, the gradient will decrease the estimated value.
  - If $V_\theta(s)$ is smaller than the target value, the gradient will increase the estimate.

3.2.2.2.3 The Relationship Between Critic and Actor.

In Proximal Policy Optimization (PPO) and Actor-Critic algorithms, Critic and Actor are closely related to each other, which enhances the learning ability and policy optimization in Reinforcement Learning.

Ingredient	Function
Actor	Choose actions based on probability distribution π_θ
Critic	Estimated state value $V(s)$ to evaluate Actor actions, support Advantage calculation $A(s,a)$, helps update Actor more efficiently.

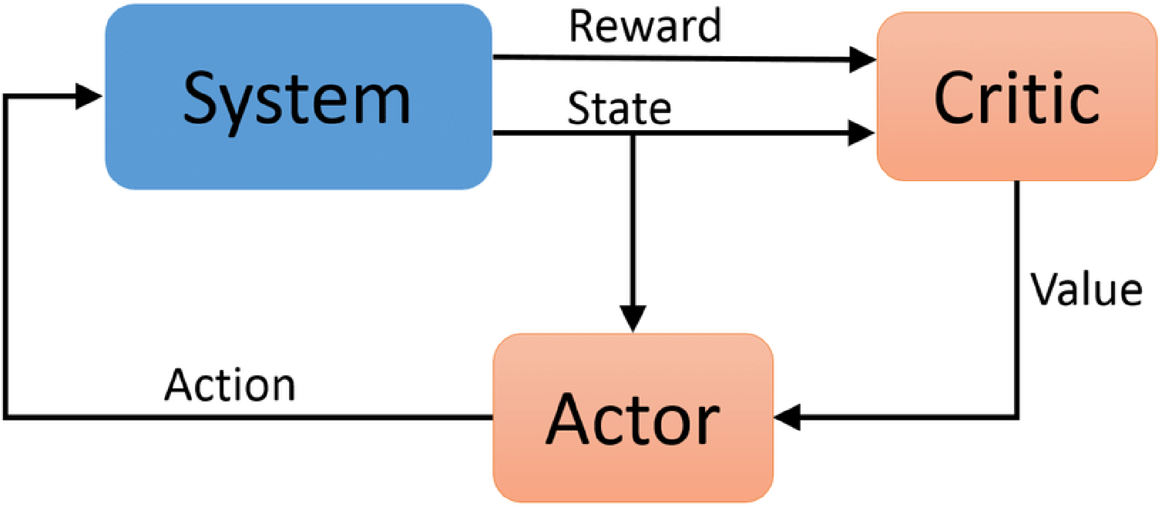


Figure 3.7: The Relationship Between Critic and Actor reference

3.2.2.3 Using Proximal Policy Optimization combined with Heuristics in Cutting Stock problem

3.2.2.3.1 Setting up elements in PPO

3.2.2.3.1.1 Environment initialization (Stock, Product, Valid_Mask)

In the process of building the **CuttingGlassEnv** environment for the glass cutting problem, the three main input data streams — **Stock, Product, and Valid_Mask** — are systematically initialized based on input data from a CSV file and predefined logical rules. Below are the details on how each data stream is initialized:

a. Initializing Stock Data

The Stock data represents the raw glass sheets available for cutting, which are initialized from the CSV file (data_custom.csv) for each specified dataset_id. The initialization process includes the following steps:

- **Reading data from CSV:**
    - ∗ The CSV file is filtered to extract rows with the type "stock" corresponding to a specific batch_id (e.g., dataset=1). Each row in the dataset contains information about the width and height of a glass sheet.
- **Creating the initial matrix:**
    - ∗ For each glass sheet, an initial matrix of size (width, height) is created with all values set to 0, representing empty cells ready for cutting.
- **Padding to a fixed size:**
    - ∗ To standardize the input size for the model, all Stock matrices are padded to a fixed size of (50, 50) using the `pad_stocks` function. The cells outside the original dimensions of the glass sheet are assigned a value of -1 to denote the padded area, while the cells within retain the value 0. This results in a 3D tensor with dimensions (num_stocks, 50, 50), where num_stocks is the number of glass sheets in the dataset.

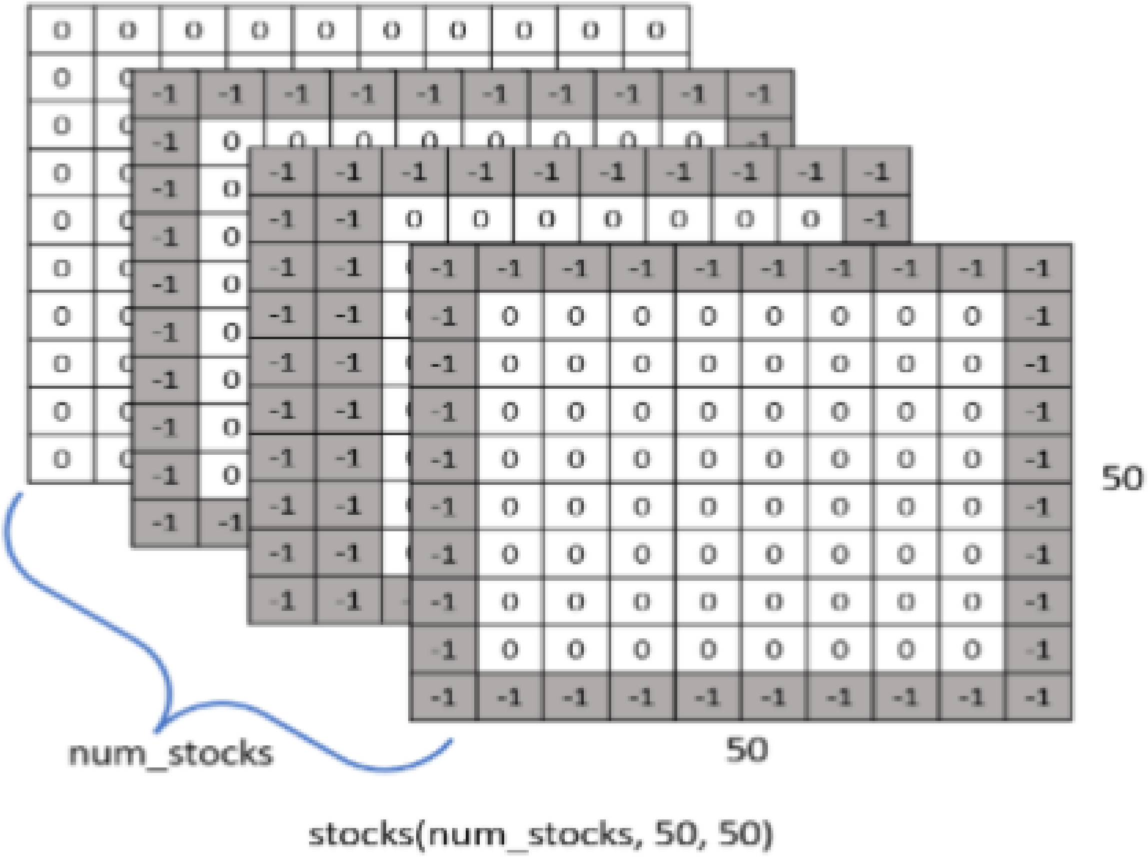


Figure 3.8: Initializing Stock Data

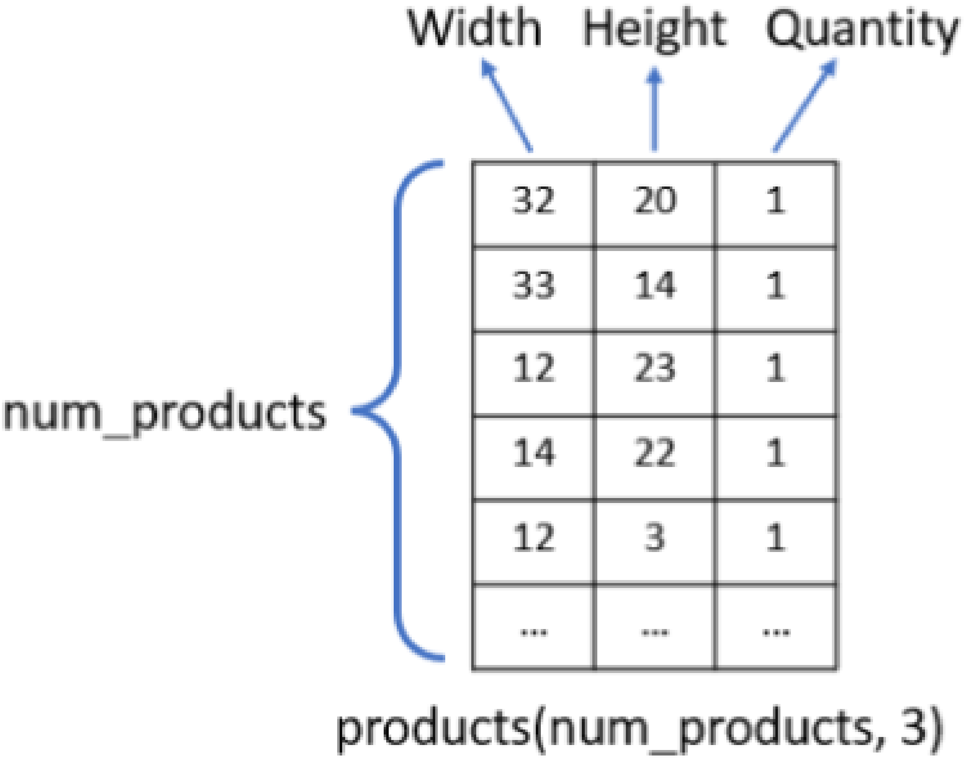
b.Initializing Product Data

The Product data represents the glass pieces to be cut, which are also sourced from the CSV file. The initialization process is as follows:

- **Reading data from CSV:**
  - ∗ Rows with the type "product" are filtered from the CSV file based on the `batch_id`. Each row contains information on the width, height, and the initial quantity of the product.
- **Preparing the information matrix:**
  - ∗ Each product is represented as a vector $[\text{width}, \text{height}, \text{quantity}]$, where the quantity is initially set to 1 (indicating the remaining number of pieces). This results

in a 2D matrix with dimensions $(\text{num\_products}, 3)$, where num_products is the number of different products in the dataset.

- **Dynamic updating:**
  - ∗ During the simulation, the quantity value in the Product matrix is decremented by 1 each time a product is successfully cut, allowing for tracking the remaining number of pieces.



Width	Height	Quantity
32	20	1
33	14	1
12	23	1
14	22	1
12	3	1
...	...	...

Figure 3.9: Initializing Product Data

c. Initializing $Valid_{M}askData$:

The Valid_Mask data is a binary matrix that designates valid positions on each glass sheet where cutting actions can be performed. The initialization and update process is as follows:

- **Initial Setup:**

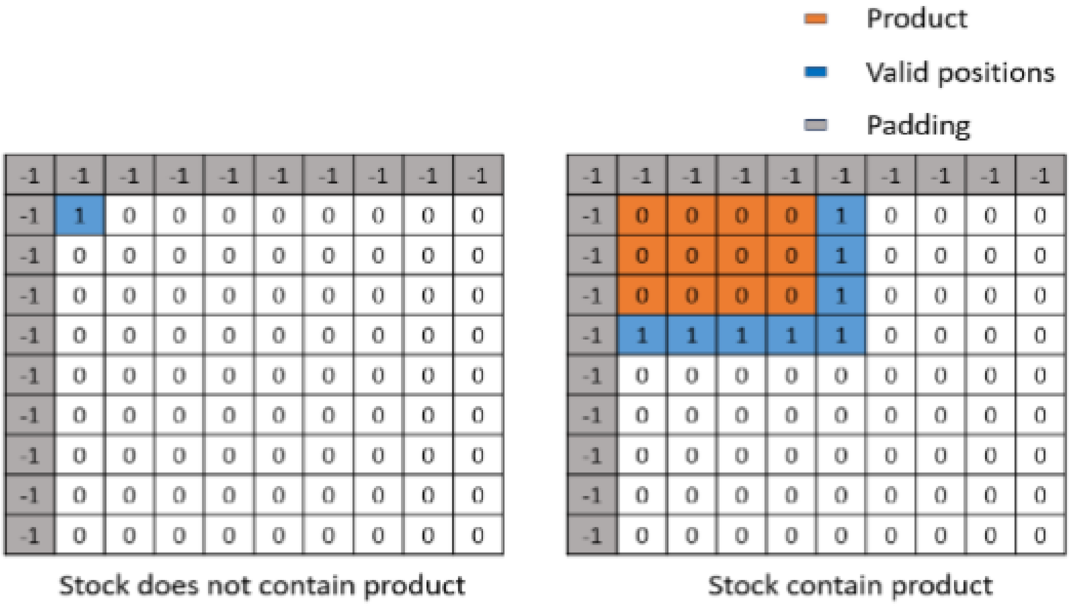
  - The Valid_Mask matrix is initialized with dimensions (`num_stocks`, 50, 50) and all values set to 0. For each glass sheet (stock), if no product has been cut (i.e., all values in the stock are 0 or -1), only the position (0,0) is set to 1 to indicate a default valid position. If products have been cut into the sheet, valid positions are determined using the `get_valid_positions` function.

- **Determining Valid Positions:**

  - The `get_valid_positions` function scans the entire stock matrix to locate cells with values greater than 0 (indicating a cut product). For each such cell, the neighboring cells (up, down, left, right) are checked. If a neighboring cell is empty (0) and falls within the boundaries of the glass sheet, its coordinates are added to the set of valid positions. As a result, the cells marked with 1 in the Valid_Mask represent the positions where the agent can choose to perform the next cut.

- **Dynamic Updating:**

  - After each successful cutting action, the `update_valid_mask` function is called to recalculate the Valid_Mask for the corresponding glass sheet. This process ensures that the Valid_Mask accurately reflects the remaining empty positions after each step, based on the current state of the stock.



-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	0	0	0	0	1	0	0	0	0
-1	0	0	0	0	1	0	0	0	0
-1	0	0	0	0	1	0	0	0	0
-1	1	1	1	1	1	0	0	0	0
-1	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0

Figure 3.10: Stock does not contain product and contain product

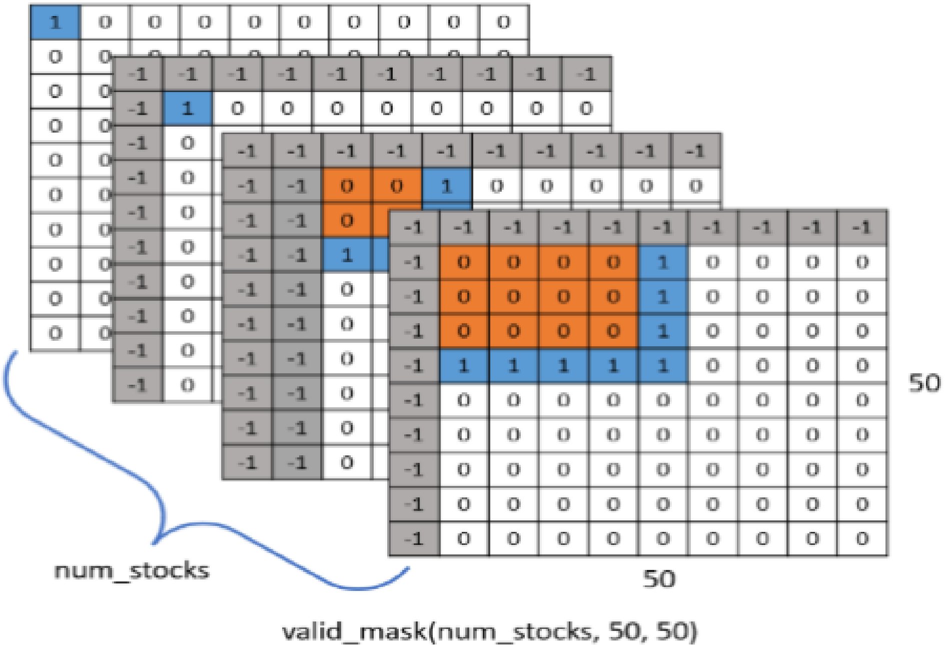


Figure 3.11: Number stock and mark

3.2.2.3.1.2 Neuron Networks initialization (CNN, MLP, Actor, Critic)

The model is built based on the Proximal Policy Optimization (PPO) algorithm to solve the glass cutting problem, using an architecture that combines a Convolutional

Neural Network (CNN) and a Multilayer Perceptron (MLP), along with dedicated Actor and Critic networks for policy and value learning. Below is a detailed description of each component:

a. Convolutional Neural Network(CNN):

The CNN is designed to extract features from 2D input data, specifically the Stock and Valid_Mask matrices. Its architecture includes convolutional layers to recognize spatial patterns in the glass sheets and valid positions. Specifically:

- **Input:** A 4D tensor of shape (`batch_size*num_stocks`, 2, 50, 50), where the first channel contains the Stock information (values ranging from -1 to `num_products`) and the second channel contains the Valid_Mask (binary values 0 or 1). The operation of combining `batch_size` and `num_stocks` merges all the glass sheets in the samples into a single dimension, ensuring the input is suitable for the CNN architecture, where each sample consists of two spatial information channels processed concurrently.
- **Structure:** The CNN consists of one or more convolutional layers (typically using a 3x3 kernel, stride of 1, and appropriate padding) to reduce the spatial dimensions and extract features. This is followed by a pooling layer (often max pooling) or a flattening layer to convert the output into a one-dimensional feature vector.
- **Output:** A feature vector of size (`batch_size`, `feature_dim`), where `feature_dim` is the number of extracted features (designed flexibly, e.g., 64 or 128). This vector represents a summary of the state of all glass sheets in the batch.
- **Purpose:** The CNN helps the model capture spatial relationships between the cells on the glass sheet, such as empty spaces or already cut regions, providing crucial input for subsequent components.

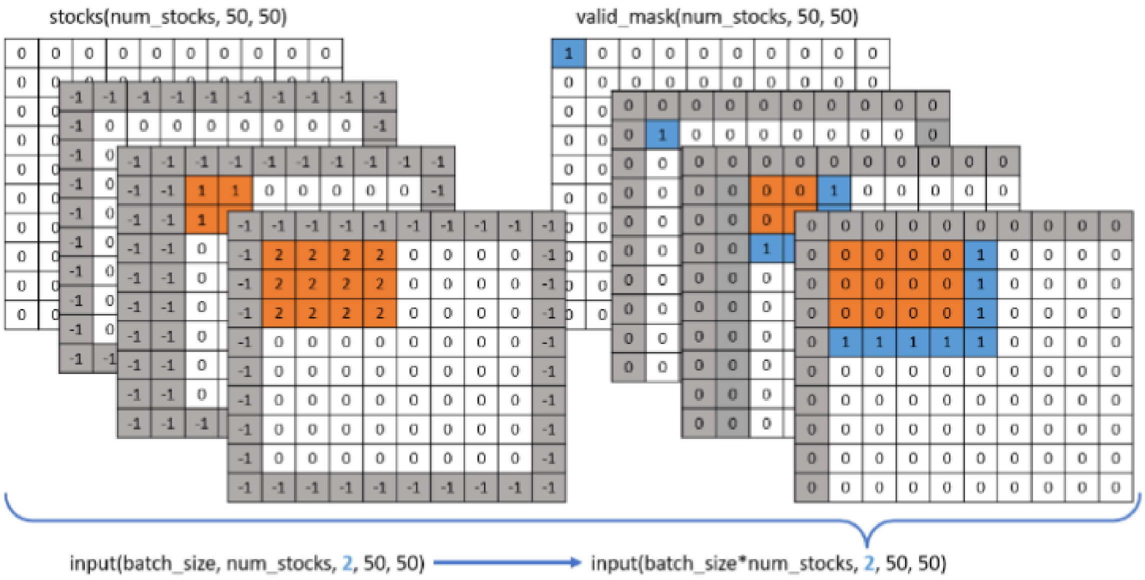

stocks(num_stocks, 50, 50)
valid_mask(num_stocks, 50, 50)
input(batch_size, num_stocks, 2, 50, 50)
input(batch_size*num_stocks, 2, 50, 50)

Figure 3.12: Input data of CNN

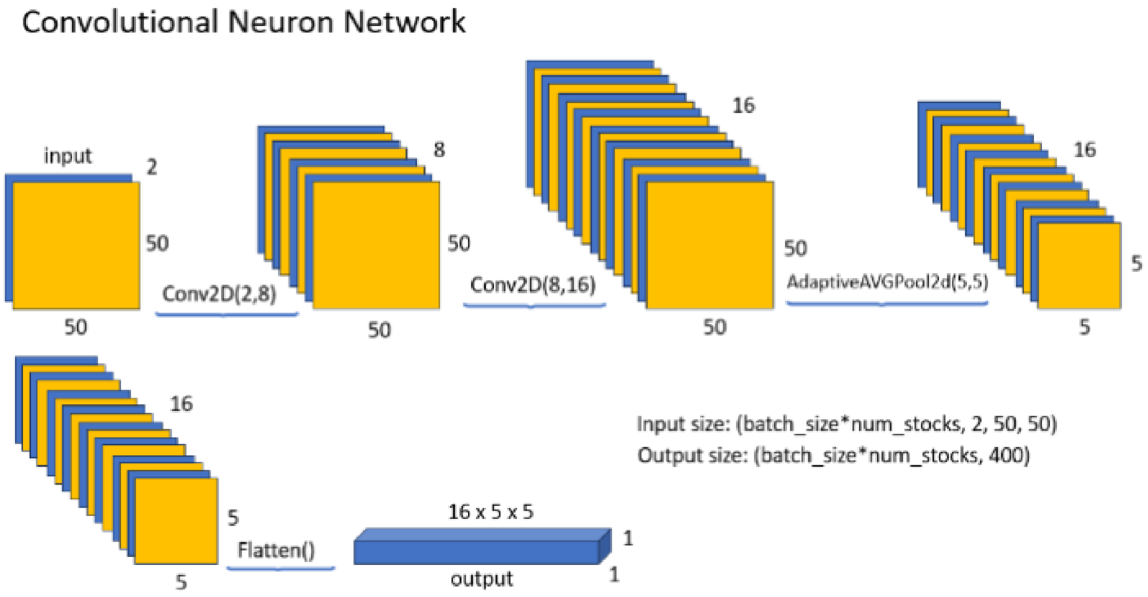

Convolutional Neuron Network
input
2
50
50
Conv2D(2,8)
8
50
50
Conv2D(8,16)
16
50
50
AdaptiveAVGPool2d(5,5)
16
5
5
16
5
5
Flatten()
16 x 5 x 5
1
1
output
Input size: (batch_size*num_stocks, 2, 50, 50)
Output size: (batch_size*num_stocks, 400)

Figure 3.13: Convolutional Neuron Network

b. Multilayer Perceptron (MLP) :

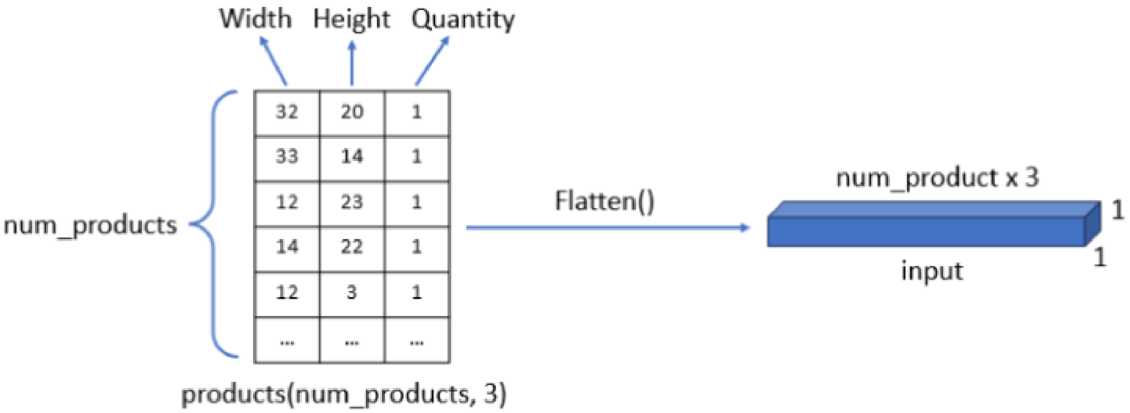
The MLP is used to process the Product data, which is a 2D matrix containing information about the width, height, and remaining quantity of the products. Its architecture includes fully connected layers to map the product information into meaningful features. Specifically:

- **Input:** A 2D tensor of shape (`batch_size`, `num_products` * 3), which is a flattened version of the matrix (`num_products`, 3).

- **Structure:** The MLP consists of one or more hidden layers, typically using the ReLU activation function to introduce non-linearity. The number of neurons in each hidden layer is designed flexibly (e.g., 64 or 128), and the output is compressed into a feature vector.

- **Output:** A feature vector of size (`batch_size`, `hidden_dim`) (for example, 8), representing a summary of the remaining product information.

- **Purpose:** The MLP encodes the product information (such as dimensions and quantity) into a form that the Actor and Critic networks can use to make decisions.



Width	Height	Quantity
32	20	1
33	14	1
12	23	1
14	22	1
12	3	1
...	...	...

Figure 3.14: Flatten

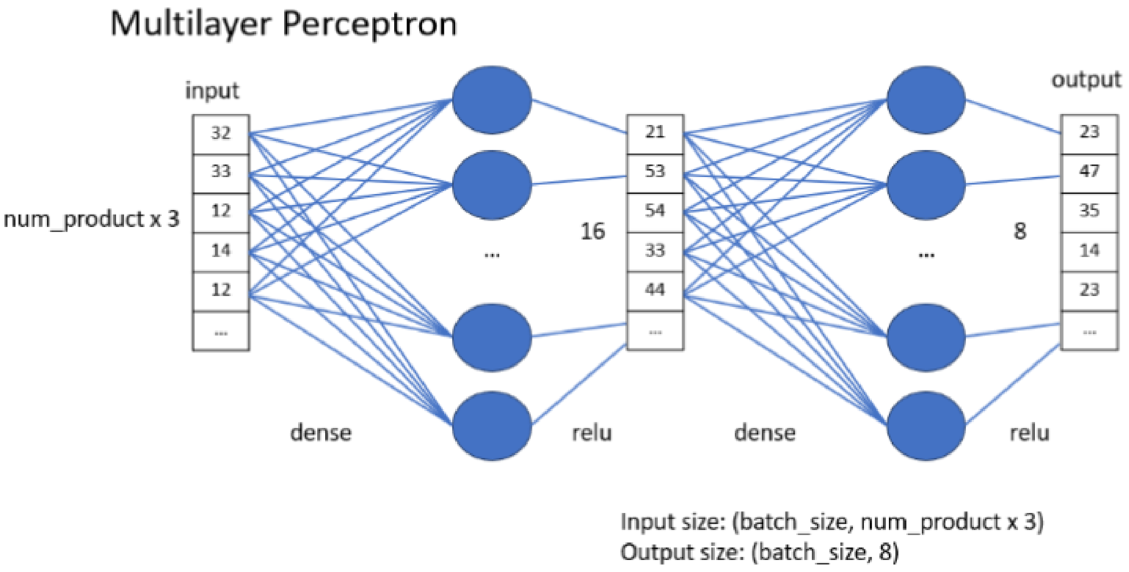


Figure 3.15: Multilayer Perceptron

c. Actor Network:

The Actor network is a component of the PPO model, responsible for learning the policy to select the optimal action based on the current state. This architecture combines the outputs from the CNN and MLP to generate a probability distribution over actions. Specifically:

- **Input:** A feature vector obtained by concatenating the outputs of the CNN (related to Stock and Valid_Mask) and the MLP (related to Product), forming a combined vector of size (`batch_size`, `feature_dim` + `hidden_dim`).
- **Structure:** The Actor network includes one or more hidden layers (typically using ReLU) to process the combined vector, and then branches out into separate outputs:
  - A Categorical distribution for the position index (`pos_logits`) with a shape of (`batch_size`, `num_stocks` * 50 * 50), representing all possible positions on the glass sheets.
  - A Categorical distribution for the rotation decision (`rot_logits`) with a shape of (`batch_size`, 2).

      - A Categorical distribution for the product index (`prod_logits`) with a shape of (`batch_size`, `num_products`).

  - **Output:** A sampled action drawn from these distributions, including (x, y, rotate, product_idx), along with the distributions used to compute the loss during training.

  - **Purpose:** The Actor network learns to optimize the policy by adjusting the probability of selecting each action, ensuring the agent chooses appropriate positions and products to cut.

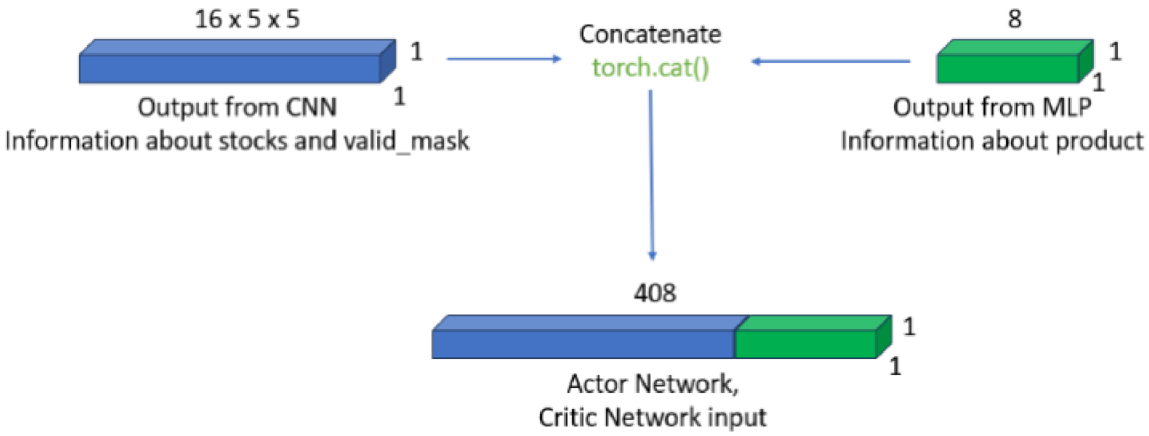


Figure 3.16: Concatenate

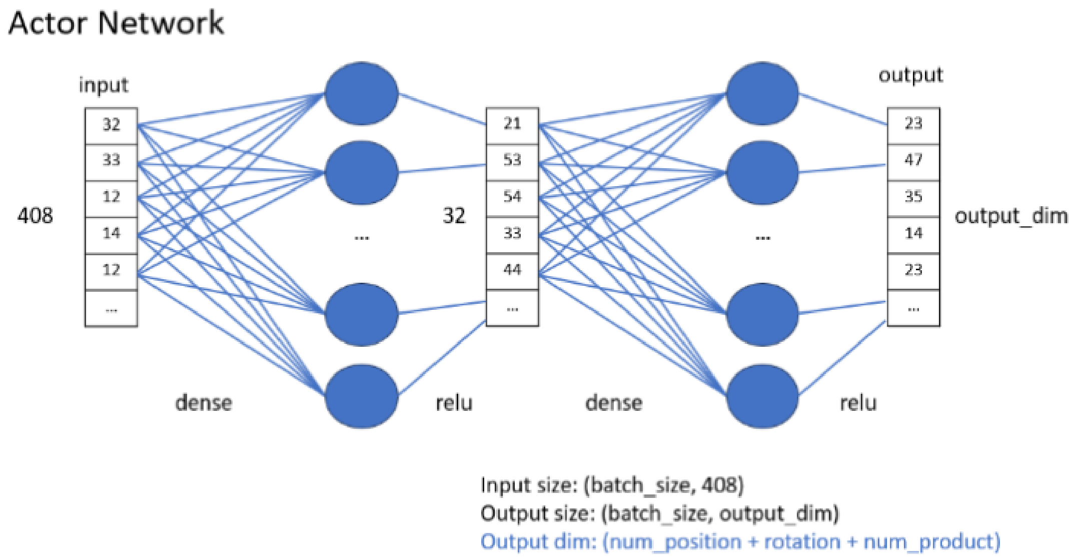

Actor Network
input
408
32
33
12
14
12
...
dense
...
relu
32
21
53
54
33
44
...
dense
...
relu
output
23
47
35
14
23
...
output_dim
Input size: (batch_size, 408)
Output size: (batch_size, output_dim)
Output dim: (num_position + rotation + num_product)

Figure 3.17: Actor Network

d. Critic Network:

The Critic network is the other component of the PPO model, designed to estimate the state value, which helps evaluate the quality of the policy. This architecture uses the same input as the Actor network to predict the expected reward. Specifically:

– **Input:** The combined feature vector from the CNN and MLP, identical to that used by the Actor network, with a shape of (`batch_size`, `feature_dim` + `hidden_dim`).

– **Structure:** The Critic network consists of one or more hidden layers (typically using ReLU), ending with a single output layer with a linear activation function to predict the state value.

– **Output:** A tensor of shape (`batch_size`, 1), representing the estimated value of the current state.

– **Purpose:** The Critic network provides feedback to adjust the Actor's policy by minimizing the discrepancy between the predicted and actual rewards, thus optimizing the PPO loss function.

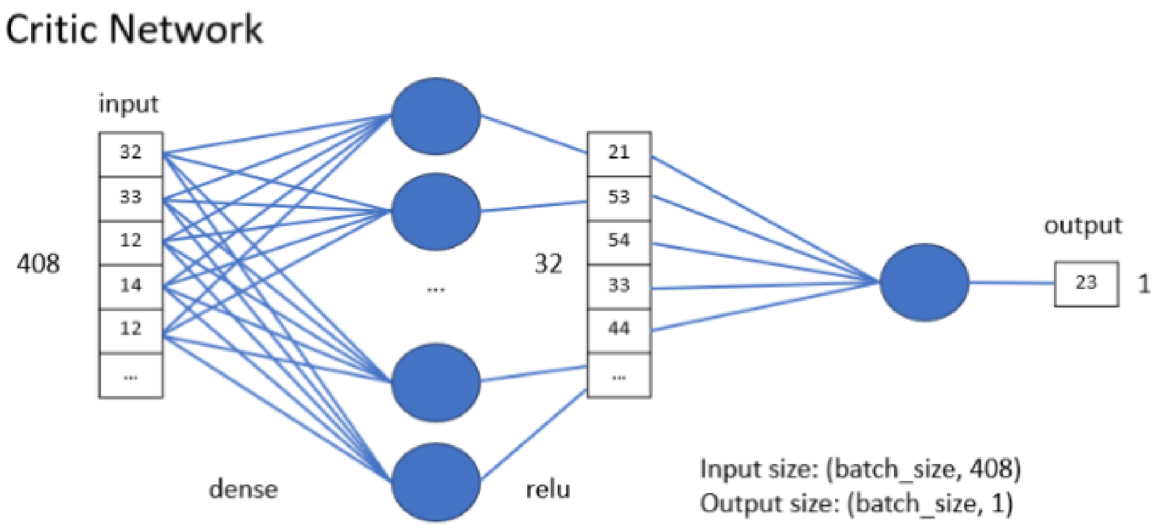


Figure 3.18: Critic Network

3.2.2.3.2 Performance

- After 21000 episodes, our PPO model has not achieved good performance in the glass cutting optimization problem. Key metrics such as Total Trim Loss (too high), Used Stocks (nearly 10), and Avg Used Stock Area (too low) indicate that the model has not learned to use the space efficiently. The model tends to utilize almost all the available glass sheets (10 sheets) without optimizing the area on each sheet, leading to significant waste (high Total Trim Loss).

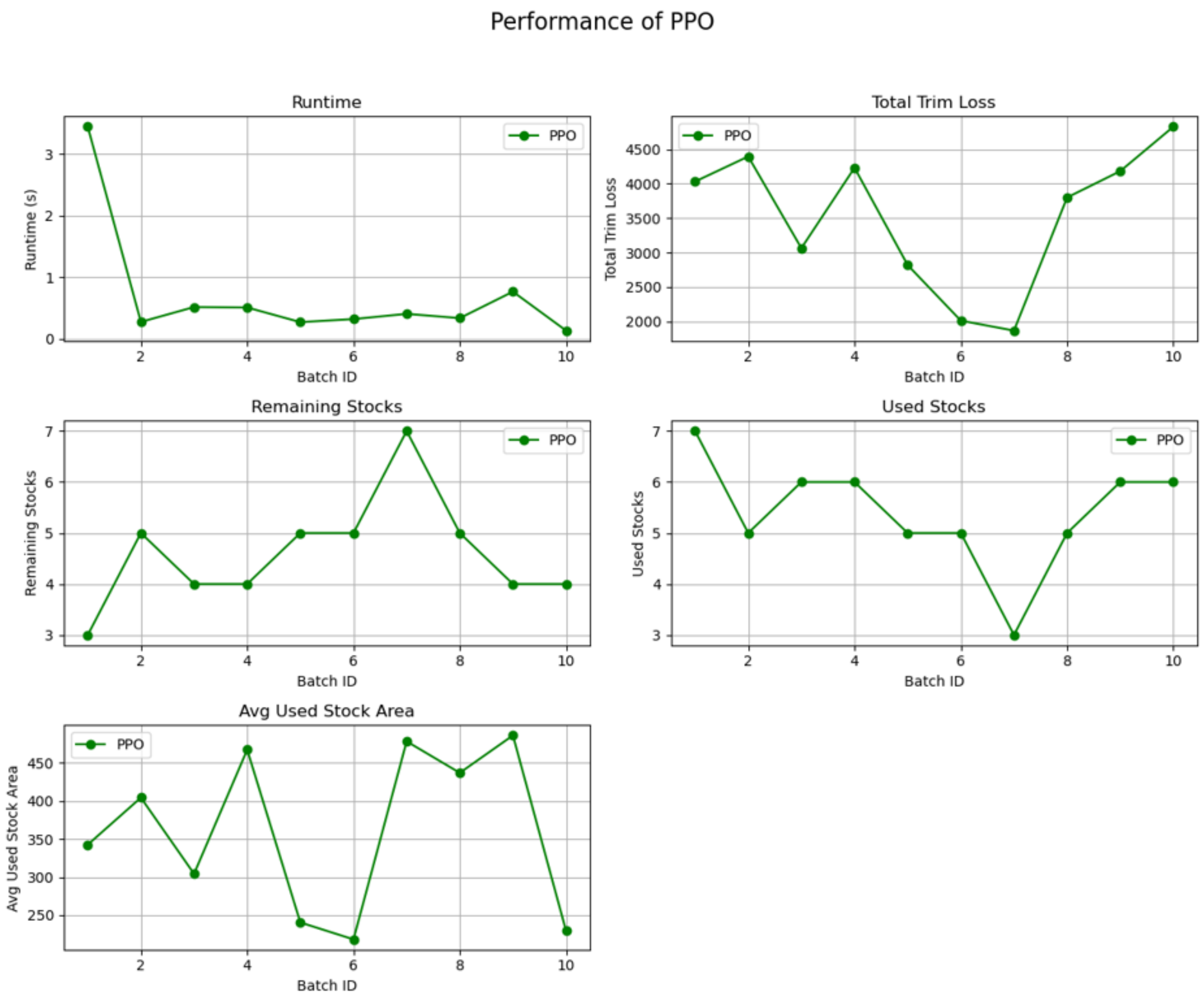


Figure 3.19: Performance of PPO

Example:

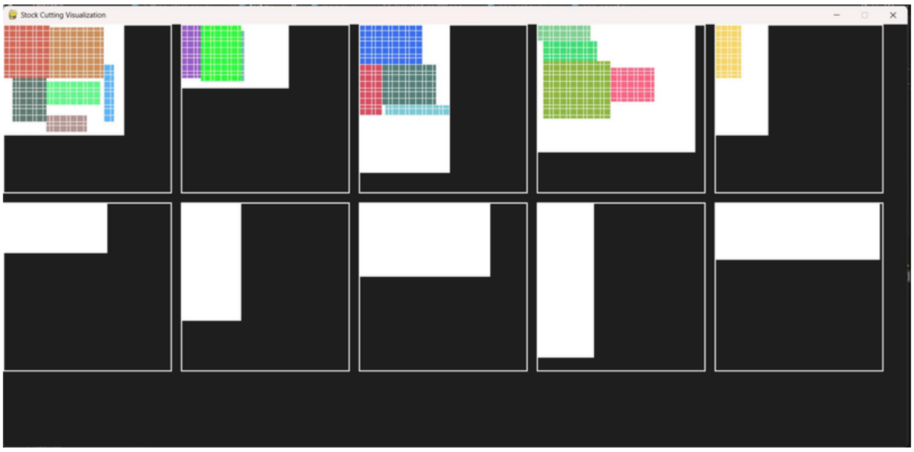


Figure 3.20: Example

Chapter 4

CONCLUSION

This project presents a comparative study between heuristic algorithms and reinforcement learning techniques in addressing the 2D cutting stock problem. Traditional heuristic methods like First Fit and Best Fit offer fast and computationally efficient solutions but often lead to suboptimal material usage. On the other hand, reinforcement learning methods such as PPO and Q-learning have shown potential in dynamically learning and improving cutting strategies over time. However, challenges such as long training times and convergence issues remain. Despite the limitations, the integration of AI techniques in industrial optimization problems provides promising results, demonstrating that reinforcement learning can enhance decision-making in real-world applications. Future work should focus on refining the learning models and incorporating hybrid approaches to further improve performance and scalability.

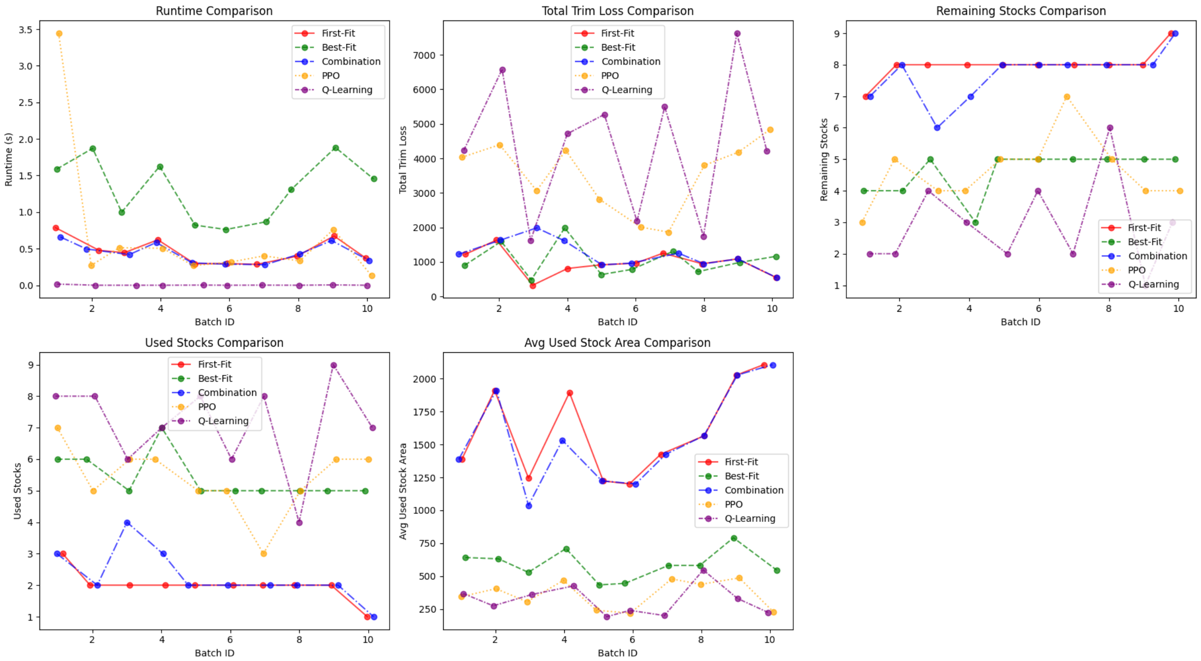


Figure 4.1: Comparison 5 methods

- **First-Fit and Combination** have the best performance in terms of material usage, inventory management, and low trim loss.

- **Best-Fit** operates stably but has a longer runtime.

- **PPO and Q-Learning** tend to be unstable, with **Q-Learning** having the highest trim loss.

- If **speed is the priority**, **Q-Learning** is the best choice. However, if **material optimization is the priority**, **First-Fit or Combination** is the optimal choice.

- This comparison helps evaluate the advantages and disadvantages of each algorithm and supports selecting the appropriate method for the **2D Stock Cutting Problem**.

- Currently, **PPO and Q-Learning** have not yet surpassed traditional algorithms in overall performance.

- However, with **larger datasets, improved algorithms, and more powerful computing resources**, they have the potential to become **the optimal solution for the 2D Stock Cutting problem in the future**.

- The application of **Reinforcement Learning (RL) in production optimization** is an important direction in modern industrial systems.

Chapter 5

DEVELOPMENT DIRECTION

5.1 Potential improvements

To enhance the performance and practical applicability of the proposed AI-based cutting stock solution, future research should consider the following directions:

1. **Model Improvement**: Enhance the reinforcement learning models by optimizing hyperparameters, incorporating advanced techniques such as deep Q-networks (DQN) or Transformer-based architectures.

2. **Hybrid Approaches**: Combine heuristic algorithms with reinforcement learning to leverage the strengths of both methods, allowing for more adaptive and efficient cutting strategies.

3. **Real-World Testing**: Deploy the trained models in real industrial environments and integrate them with computer vision techniques to handle real-time adjustments in the cutting process.

4. **Scalability Enhancement**: Improve the model's ability to handle larger and more complex datasets by using cloud computing or distributed learning frameworks.

5. **Multi-Objective Optimization**: Extend the model to consider additional factors such as production time, energy consumption, and machine efficiency, making it

more applicable to diverse manufacturing scenarios.

By addressing these areas, the project can significantly contribute to the advancement of AI-driven optimization techniques in industrial applications, ensuring greater efficiency, cost savings, and sustainability.

5.2 Source code and data

All materials and everything related to the source code can be found at this github branch of the original project files this report.

The raw data that we use to serves as the foundation for our case-study analysis, along sise all images and materials used for this report can be found at this data.

5.3 References

[1] https://medium.com

[2] https://www.geeksforgeeks.org/a-brief-introduction-to-proximal-policy-optimization/

[3] https://github.com/NgocMinhUniversityProjects/MM241-Assignment

[4] https://viblo.asia/p/reinforcement-learning-q-learning-63vKjO7VZ2R

[5] https://sofdem.github.io [6] https://pure.iiasa.ac.at

[7] https://www.researchgate.net/publication

[8] https://anhvnn.wordpress.com

[9] https://www.sciencedirect.com

[10] https://www.researchgate.net/publication