Chapter 1

Implementation Plan and Tools

1.1 Phase 1 – Data Collection & Preparation

CreditCard Dataset

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred over a two-day period, where we have 492 frauds out of 284,807 total transactions. The dataset is highly unbalanced: the positive class (frauds) accounts for only 0.172% of all transactions.

It contains only numerical input variables, which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, the original features and more background information about the data cannot be provided. Features `V1, V2, ..., V28` are the principal components obtained with PCA. The only features that have not been transformed are `Time` and `Amount`.

`Time` contains the number of seconds elapsed between each transaction and the first transaction in the dataset. `Amount` is the transaction amount and can be used for example-dependent cost-sensitive learning. The `Class` feature is the response variable, where it takes the value 1 in case of fraud and 0 otherwise.

In addition to the original preprocessing, a synthetic textual representation of each transaction was generated to enable language model fine-tuning. This involved crafting descriptive sentences that encode the transaction amount, time, and selected principal component values (V1 to V8). The goal was to translate the structured numerical data into natural language prompts suitable for transformer-based models.

To address the severe class imbalance inherent in the dataset (fraud cases comprising only 0.172% of all transactions), we applied undersampling on the majority class. Specifically, a balanced subset was created by retaining all fraudulent transactions and randomly sampling a fixed multiple ($5\times$) of nonfraudulent transactions to preserve a representative, yet computationally manageable, dataset for downstream tasks.

Following balancing, the dataset was split into training and testing sets using a standard 80/20 stratified approach to ensure consistency in class distribution. The resulting text-labeled samples were then tokenized using a DistilBERT tokenizer with truncation and padding, setting a maximum token length of 256. This enabled efficient use of pre-trained transformer models for binary classification.

Finally, the tokenized datasets were formatted into PyTorch tensors and prepared for training, including considerations for class imbalance mitigation via weighted loss functions.

1.2 Phase 2 – LLM Model Selection & Fine-Tuning

CreditCard Dataset

In this phase, we explored the application of pre-trained transformer models on fraud detection framed as a natural language processing task. Specifically, we selected `DistilBERT`, a lightweight and efficient variant of BERT, as the backbone model due to its balance of performance and computational cost.

We framed the problem as binary classification (legitimate vs. fraud) using a sequence classification head on top of the DistilBERT encoder. The preprocessed transactions—converted into natural language descriptions—were tokenized using the model's tokenizer with appropriate truncation and padding strategies to ensure uniform input length and batch efficiency.

Training Setup

The model was fine-tuned using the AdamW optimizer with a learning rate of 2×10^{-5}, a linear learning rate scheduler with warmup steps, and a batch size of 16. We trained the model over 4 epochs, using early stopping and model checkpointing based on validation loss to prevent overfitting.

The training loop utilized a weighted cross-entropy loss function to counteract class imbalance, with higher penalties applied to misclassified fraud samples. This weighting encouraged the model to prioritize the minority class during gradient updates without sacrificing performance on the legitimate class.

Both training and validation losses were tracked and plotted per epoch. Training loss consistently decreased, and validation loss followed a similar trend before plateauing—indicating convergence. There were no signs of overfitting within the selected epoch range, confirming the effectiveness of early stopping.

Model Evaluation

After training, the model was evaluated on a held-out test set. The confusion matrix indicated strong performance across both classes, especially in fraud detection: For legitimate transactions, precision reached **0.98** with recall at **0.93** and an F1-score of **0.96**. Fraud detection showed slightly lower precision (**0.75**) but excellent recall (**0.94**). Overall accuracy stood at **0.93**.

These results indicate that the model maintained high precision for legitimate cases while achieving high recall for fraud cases—an essential characteristic in fraud detection where false negatives must be minimized.

Embedding Extraction for RL Integration

Beyond classification, we leveraged the encoder portion of the fine-tuned DistilBERT to extract contextualized text embeddings. By mean-pooling the final hidden states across the token sequence, we obtained dense fixed-size vector representations of each transaction. These embeddings were detached from the computational graph and used as state features in our downstream reinforcement learning (RL) pipeline. This design enabled the RL agents to operate over semantically rich, high-dimensional representations informed by language modeling, thereby enhancing their ability to learn decision policies based on transaction patterns.

1.3 Phase 3 – Environment and Simulator Development

To train reinforcement learning (RL) agents for fraud detection, we designed a custom simulation environment that interfaces seamlessly with precomputed transaction embeddings. The environment was developed using the `Gym` API from OpenAI, allowing us to frame fraud detection as a sequential decision-making problem.

Embedding Preprocessing

Before constructing the environment, we precomputed the contextualized text embeddings of each transaction using the fine-tuned DistilBERT model. Each transaction description was transformed into a fixed-size 768-dimensional vector by applying mean pooling across token embeddings from the final hidden layer. These embeddings captured rich semantic representations of transaction behavior and served as the observation space for the RL agent.

The datasets were split as follows:

- **Training Embeddings:** 2,361 instances with shape (2361, 768)
- **Testing Embeddings:** 591 instances with shape (591, 768)

Label distribution was preserved across splits, maintaining a consistent fraud ratio:

- **Original Dataset:** Fraud = 16.67%

- **Training Split:** Fraud = 16.69%
- **Testing Split:** Fraud = 16.58%

Custom RL Environment

The environment, named `FraudDetectionEnv`, simulates a fraud detection task as a sequential binary classification game. The agent observes one transaction embedding at a time and chooses between two actions:

- `Action 0:` Declare transaction as *not fraud*
- `Action 1:` Declare transaction as *fraud*

Each step corresponds to evaluating one transaction, and the episode spans the full dataset (i.e., each instance is visited once per episode).

Observation Space:

- A 768-dimensional continuous vector, defined by a Box space in Gym.

Action Space:

- A discrete binary space: `Discrete(2)`.

Reward Function:

The environment applies a custom reward scheme to guide the agent's learning behavior, tuned to reflect the asymmetric costs of misclassification:

- **True Positive (TP):** +10.0
- **False Positive (FP):** –5.0
- **False Negative (FN):** –20.0
- **True Negative (TN):** +1.0

This asymmetric reward design incentivizes the agent to detect fraudulent transactions correctly while discouraging both types of errors—especially false negatives, which have the highest cost in real-world fraud detection systems.

Environment Dynamics

During each episode:

- Transactions are presented in a randomly shuffled order to ensure generalization.
- At each step, the agent receives a new embedding, predicts fraud or not fraud, and receives a reward based on the actual label.
- The environment continues until all transactions in the dataset have been processed (i.e., one full episode).

1.4 Phase 4 – Training the RL Agent

CreditCard Dataset

In the final phase of our pipeline, we leveraged reinforcement learning (RL) to train agents capable of classifying financial transactions as either fraudulent or legitimate, based solely on their high-dimensional text-derived embeddings. Two widely adopted RL algorithms were employed: Deep Q-Networks (DQN) and Advantage Actor-Critic (A2C), each offering distinct strengths in learning optimal decision policies.

Model Architecture and Shared Design

Both agents were built upon a shared neural network backbone defined by the `MlpPolicy` from the Stable-Baselines3 library. This architecture consists of two fully connected layers, each with 64 hidden units, tailored for one-dimensional vector observations such as our 768-dimensional transaction embeddings. This design offers sufficient capacity to capture the nuanced patterns needed to separate fraudulent behaviors from legitimate ones.

Deep Q-Network (DQN)

The DQN agent learns to approximate the optimal action-value function through experience replay and target network updates. Its training process emphasizes long-term reward maximization by iteratively refining its estimates of the expected future reward for each possible action. To encourage exploration in the early training stages and promote exploitation of learned policies later, an ε-greedy exploration strategy was adopted. The agent initially explores the environment broadly and gradually transitions to more deterministic actions as its understanding improves.

Experience replay was a core component in stabilizing DQN training, allowing the model to learn from past experiences sampled uniformly from a buffer. To further ensure training stability, gradient magnitudes were clipped, and learning dynamics were regulated through scheduled exploration and learning rate decay.

Advantage Actor-Critic (A2C)

In parallel, we trained an A2C agent—an on-policy actor-critic method that maintains separate but synchronized components: an actor that proposes actions and a critic that evaluates them. A2C offers advantages in settings where efficient, real-time updates from recent interactions can accelerate convergence.

Like DQN, the A2C agent utilized the same multilayer perceptron architecture. However, its training incorporated distinct optimization dynamics, including entropy regularization to promote continuous exploration and a separate loss term to ensure accurate value function approximation. Unlike DQN, A2C does not use experience replay; instead, it performs updates using a sequence of recent transitions, which enables the model to remain closely aligned with the evolving environment.

Exploration in A2C was encouraged through an entropy-based mechanism, allowing the agent to remain uncertain about its action selection in earlier training phases. This approach avoids premature convergence and encourages broader state-space coverage.

Training Infrastructure

Both models were trained using Stable-Baselines3 with GPU acceleration that was employed where available, significantly reducing training time and improving performance scalability.

By comparing these two learning paradigms—off-policy (DQN) and on-policy (A2C)—we sought to understand the efficacy and robustness of each approach in the fraud detection context. This dual-agent setup provided valuable insights into the trade-offs between stability, convergence speed, and classification performance.

Chapter 2

Evaluation & Benchmarking

2.1 Results & Analysis

2.1.1 DQN Model

To assess the effectiveness of the trained DQN agent in classifying transactions, we conducted a comprehensive evaluation involving reward trends, Q-value analysis, and classification performance. The results are presented through both quantitative metrics and qualitative reward dynamics visualizations.

Mean Episode Reward

During training, the agent's episode reward underwent distinct phases:

Initial Rapid Increase: In the early steps, as the exploration rate (ε) decayed from its maximum, the agent began exploiting its learned Q-values. This transition enabled sharp performance gains as it started to make informed decisions.

Moderate Ascent: As training progressed, the learning rate gradually declined, leading to more conservative updates. The reward curve smoothed out, indicating policy refinement and improved stability.

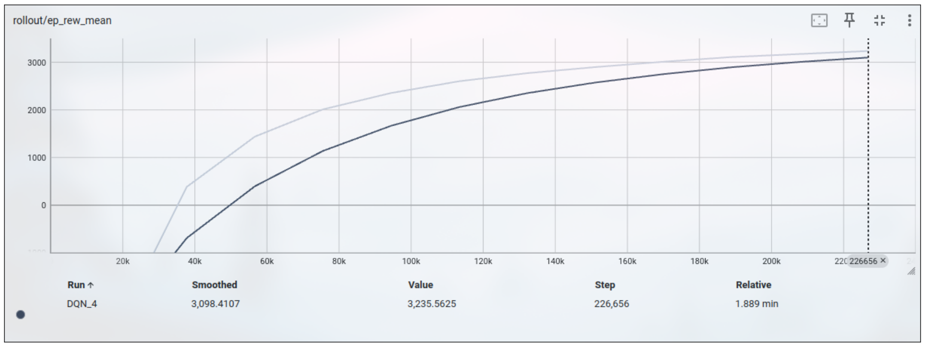
These patterns suggest that the exploration and learning schedules were effective in guiding the agent from exploratory behavior to reliable decision-making.

Training Loss Evolution

The training loss exhibited the following trends:

High Initial Loss: Training started with large temporal-difference (TD) errors due to random replay samples and an untrained Q-network. This was expected and reflected the model's initial lack of knowledge.

Stabilization and Decline: As the agent improved, the loss steadily decreased. Large batch sizes and gradient clipping helped reduce variance and

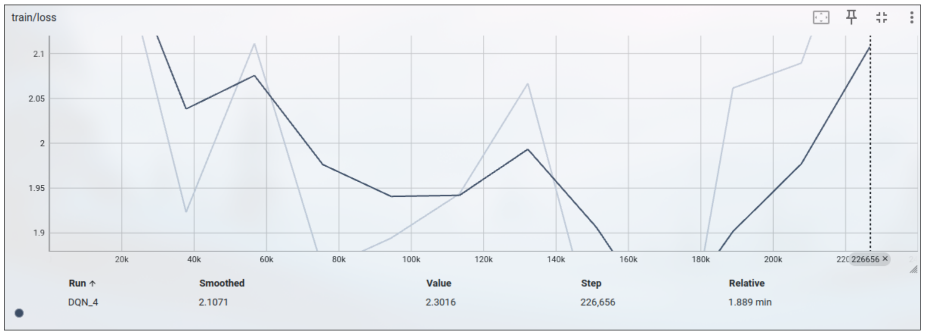


Run ↑	Smoothed	Value	Step	Relative
DQN_4	3,098.4107	3,235.5625	226,656	1.889 min

Figure 2.1: Mean reward per Episode

contributed to stable updates.

Late-Stage Increase: Towards the end of training, the replay buffer became dominated by on-policy transitions as ε approached its minimum. This reduced sample diversity and caused the TD error to slightly rise again, suggesting that early stopping could be beneficial to halt training before overfitting to these transitions.



Run ↑	Smoothed	Value	Step	Relative
DQN_4	2.1071	2.3016	226,656	1.889 min

Figure 2.2: Training Loss

Evaluation Reward Analysis

Evaluation was conducted using a fixed environment to measure the agent's learned policy in terms of reward acquisition. Figure 2.3 summarizes this with two key plots.

Step-Wise Rewards (Left Panel):

- **Positive Rewards:** Frequent and consistent, these suggest successful actions such as correctly identifying fraud or non-fraud.
- **Negative Rewards:** Less frequent but impactful, these likely reflect misclassifications—either false positives or false negatives.

- **Zero Rewards:** Representing neutral outcomes, these may indicate uncertain or ambiguous transaction decisions.

Cumulative Rewards (Right Panel): The cumulative reward increased steadily throughout the evaluation episode, surpassing 1000 by step 600. This upward trend implies that the agent's policy led to consistent positive outcomes, reinforcing the effectiveness of the learned decision strategy.

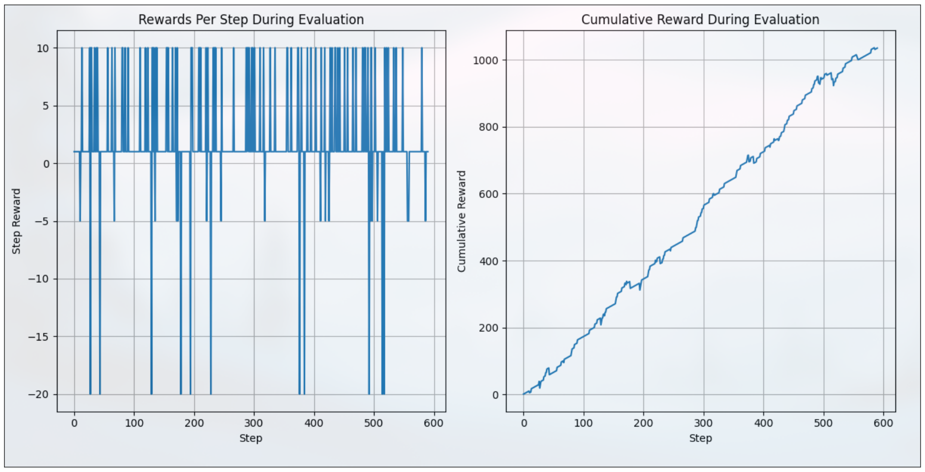


Figure 2.3: Reward Dynamics During Evaluation. Left: Step-wise rewards. Right: Cumulative reward trajectory.

Average Q-Values Per Action

Figure 2.4 presents the average Q-values associated with each action during evaluation.

- Both actions exhibited similarly high average Q-values, indicating high confidence in decision-making regardless of the chosen label.
- This balance implies that the agent has learned effective strategies for both detecting fraud and correctly identifying legitimate transactions.
- The model does not favor one action over the other, a sign of robustness and fairness in decision-making across classes.

Classification Performance Metrics

To quantify the classification ability of the trained agent, we computed standard metrics from its confusion matrix, shown in Figure 2.5.

- **Accuracy:** 95.09% – The agent achieved high correctness across all transactions.

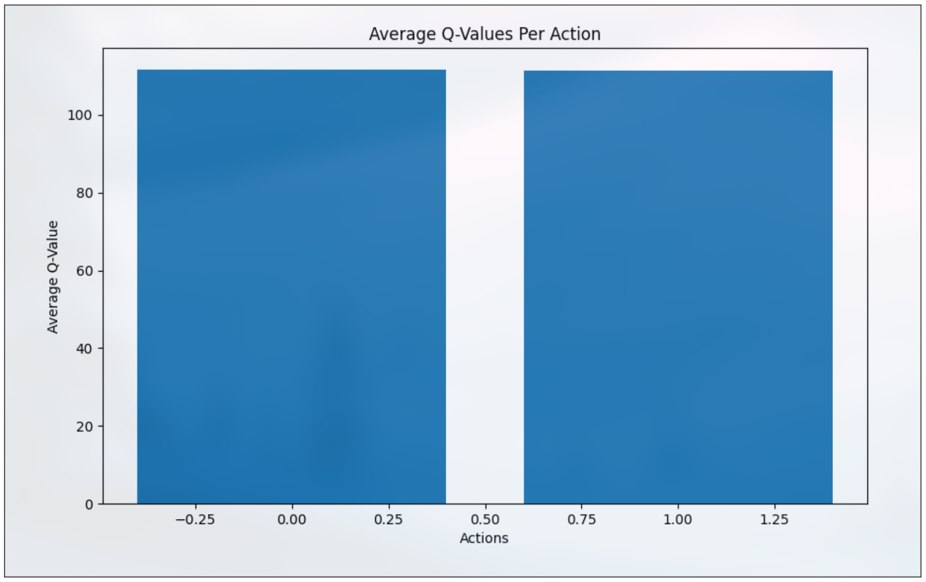


Figure 2.4: Average Q-Values for Actions 0 (non-fraud) and 1 (fraud) during evaluation.

- **Precision (Fraud):** 82.86% – Of predicted frauds, the majority were truly fraudulent.
- **Recall (Fraud):** 88.78% – The agent identified most actual fraudulent transactions.
- **F1 Score (Fraud):** 85.71% – A strong balance between avoiding false alarms and detecting real frauds.

The confusion matrix reflects high effectiveness in both fraud detection (true positives) and minimizing false positives, underscoring the agent's practical utility.

2.1.2 A2C Model

To evaluate the A2C agent's behavior and classification capabilities, we analyzed policy dynamics, cumulative reward trends, and classification outcomes during evaluation episodes. The following subsections detail the agent's performance.

Policy Loss Curve

The policy loss trajectory during training reveals significant insights into the learning dynamics.

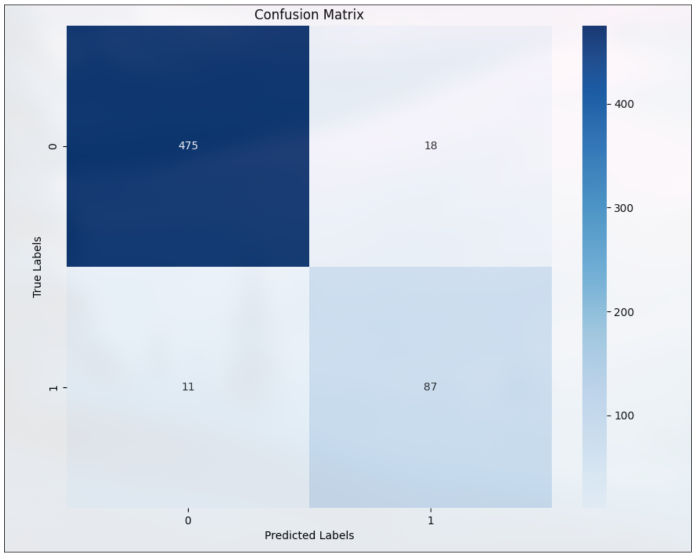
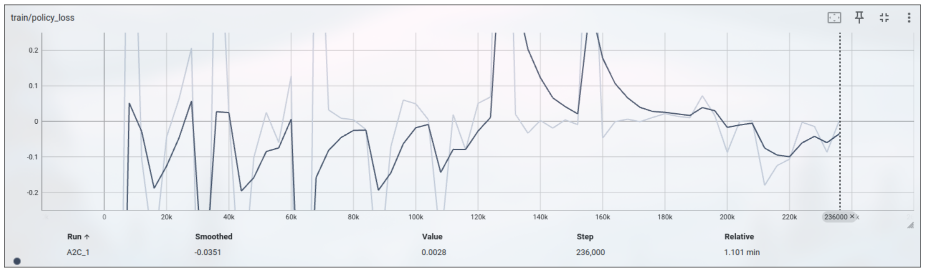


Figure 2.5: Confusion Matrix of DQN Model on Test Set.

- **Oscillatory Pattern:** Sharp spikes and drops occur roughly every 20,000 steps. These likely align with environment resets or model checkpoint events.
- **Stabilization Phase:** After approximately 120,000 steps, the oscillations dampen, and the loss consistently hovers near zero.
- **Final Value:** At the end of training (step 236,000), the smoothed loss reaches approximately -0.035, indicating small corrective updates and a nearly deterministic policy.



Run ↑	Smoothed	Value	Step	Relative
A2C_1	-0.0351	0.0028	236,000	1.101 min

Figure 2.6: Policy Loss Curve for A2C Agent during Training

Action Probabilities and Policy Entropy

Action Probabilities:

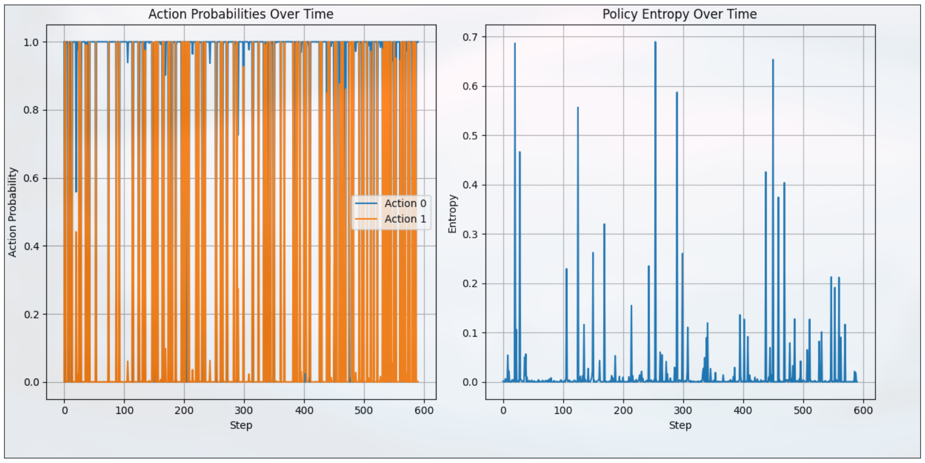


Figure 2.7: A2C Agent Evaluation: Action Probabilities and Policy Entropy

- The action probabilities are almost binary—typically at 0 or 1—across all 600 evaluation steps.
- This indicates a confident and nearly deterministic policy. Dense clusters of "Fraud" actions suggest strong signal detection in embedded state representations.

Policy Entropy:

- Entropy is effectively zero across most of the evaluation.
- Occasional spikes (up to 0.7) appear in ambiguous states, where the agent temporarily reconsiders less certain actions.
- These entropy spikes reveal moments of uncertainty, but the general trend confirms confident decision-making.

4. Confusion Matrix

- **True Negatives (463):** 93.9% of legitimate transactions correctly classified.
- **True Positives (88):** 89.8% of fraudulent transactions accurately detected.
- **False Positives (30):** 6.1% of legitimate transactions incorrectly flagged as fraud.
- **False Negatives (10):** 10.2% of fraudulent transactions missed.

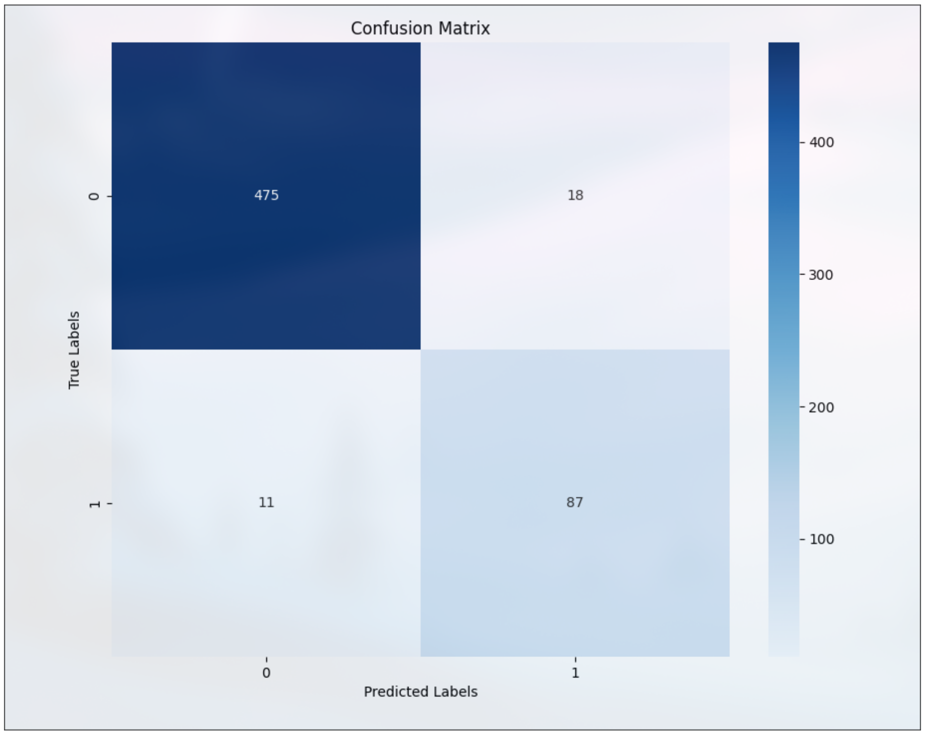


Figure 2.8: Confusion Matrix of A2C Model on Test Set

5. Classification Metrics Summary

- **Accuracy:** 93.23%
- **Precision (Fraud):** 74.58%
- **Recall (Fraud):** 89.80%
- **F1 Score (Fraud):** 81.48%
- **Average Reward per Step:** ∼1.68

Interpretation

The A2C agent successfully learns a high-recall policy capable of detecting nearly 90% of fraudulent transactions, at the cost of a moderate false-positive rate. Its nearly deterministic action probabilities and low entropy reflect strong confidence in its decisions. The cumulative reward trajectory further reinforces the model's robustness, showcasing a stable and effective policy under the fraud detection reward design.

Chapter 3

Challenges, Pitfalls and Resources

3.1 Key Challenges and Mitigation

Fraud detection with LLM+RL poses several overarching challenges:

[leftmargin=*,noitemsep,topsep=0pt]**Extreme Class Imbalance:** Positive (fraud) cases are rare (e.g. 0.17% in Credit Card). Mitigation: cost-sensitive reward shaping, stratified sampling, GAN-based oversampling, and monitoring AUPRC rather than accuracy. **Overfitting & Generalization:** High model capacity (LLM + deep RL) risks memorizing training fraud patterns. Mitigation: regularization (weight decay, dropout), early stopping via validation reward, and cross-validation on held-out temporal splits. **RL Stability & Sparsity:** Sparse fraud rewards can destabilize off-policy learners (e.g. DQN) and lead to catastrophic forgetting. Mitigation: reward shaping (intermediate penalties), prioritized experience replay, entropy regularization, and evaluation callbacks to detect divergence early. **Explainability:** Combined LLM embeddings and RL policies are opaque to analysts. Mitigation: logging LLM attention weights, using post-hoc explainer methods (SHAP/LIME) on embedding+policy inputs, and storing rationale texts from the LLM.

3.1.1 Datasets Considerations

Each dataset brings unique preprocessing and resource challenges:

Credit Card (IEEE-CIS) Numeric PCA features only; no text, but extreme imbalance (0.172%). Preprocessing is lightweight; main risk is over-reliance on engineered components (Time, Amount).

PaySim Mobile Transactions Synthetic, categorical and numeric fields with injected fraud scenarios. Simulator fidelity may not transfer to real-world distributions—requires careful domain adaptation and stress-testing.

SEC Filings (MD&A Texts)

[leftmargin=1em,noitemsep,topsep=0pt]**Huge Document Size:** Average 1.28M characters per report, exceeding typical LLM context windows (2K–32K tokens). **Chunking Pipeline:** We segmented each filing into overlapping chunks (e.g. 2 048-token windows with 50% stride) to preserve continuity. This adds complexity in data loaders and increases storage for intermediate text files. **Encoding Complexity:** Each chunk must be tokenized and embedded separately, then aggregated (e.g. mean, attention-based pooling) to form a single state vector. Designing an efficient batching strategy and aggregator is non-trivial. **Compute Resources:** Fine-tuning LLMs on these chunks demands large GPU memory (¿16 GB), multi-GPU parallelism or model-parallel frameworks, mixed-precision, and gradient checkpointing to avoid OOM. End-to-end backprop through both LLM and RL agent further multiplies resource requirements. **Limitations:** Chunk-level processing loses global document context; embedding aggregation may dilute local signals. High preprocessing and inference latency hinders real-time deployment.

3.2 Additional Resources

For reproducibility and further development, we recommend:

[leftmargin=*,noitemsep,topsep=0pt]**Transformers & Tokenizers:** HuggingFace Transformers, Datasets, Tokenizers, and Accelerate for LLM fine-tuning and chunk management. **Reinforcement Learning:** Stable-Baselines3, RLlib; evaluation callbacks and integration examples in their documentation. **Explainability Tools:** SHAP, LIME for post-hoc analysis of embeddings and policy inputs. **Similarity Search:** FAISS for efficient retrieval of relevant chunks during inference. **Key Papers & Repos:**[leftmargin=1em,noitemsep]

- – Dang et al. (2021) "Reinforcement Learning for Fraud Detection"
  – Purva Singh et al. Gym-based DQN implementation (GitHub)
  – Zhao et al. (2023) GPT-based payment fraud model
- **Compute Platforms:** AWS p3/p4 instances, GCP TPU v3, or on-prem NVIDIA DGX for large-scale LLM+RL training.