Universität Zürich[UZH]

UZH Blockchain Center

Graph Inverse Reinforcement Learning for MEV Detection on Ethereum

Master Thesis

Submitted in partial fulfillment of the requirements for the degree of MSc UZH Informatics

Author

Liam Joshua Tessendorf

Glaubtenstrasse 96, 8046 Zurich

Matriculation Number: **20-701-348**

Email: LIAM.TESSENDORF@UZH.CH

Supervisor

Prof. Dr. Claudio J. Tessone

Co-Supervisors

Dr. Taehoon Kim, Krzysztof Gogol

Blockchain & Distributed Ledger Technologies

Department of Informatics

University of Zurich

Date of Submission: June 22, 2025

Executive Summary

This thesis introduces a novel framework for detecting Maximal Extractable Value (MEV) arbitrage transactions on the Ethereum blockchain by integrating Graph Neural Networks (GNNs) with Adversarial Inverse Reinforcement Learning (AIRL). It extends current GNN-based MEV detection models by incorporating AIRL, introducing agentic, utility-based learning that produces discriminative reward functions through interaction with the environment.

Baseline classification using Random Forest (RF) algorithms demonstrated strong predictive performance from simple, tabular transaction-level features, achieving F1-scores above 98%. A GraphSAGE-based classifier was used to incorporate structural context from transaction graphs, but achieved a slightly lower F1-score of 96.1%. However, a hybrid model that combined GraphSAGE embeddings with tabular features improved performance, achieving an F1-score of 99.2%. This improvement was statistically significant (McNemar's test, $p < 0.01$), confirming the complementary value of structural and transactional features.

The central contribution is the integration of AIRL with GNN-based state embeddings to model arbitrage-related transactions dynamically. Separate AIRL models trained on arbitrage and non-arbitrage transactions generated reward functions that were subsequently used for binary classification. The reward function learned via AIRL using the supervised GraphSAGE embeddings yielded the highest classification performance, achieving an F1-score of 99.4%, outperforming all previous methods.

Findings highlight that AIRL, when combined with supervised GNN embeddings, produces reward functions that serve as effective and interpretable classifiers. This dual utility, as both a policy optimization tool and a discriminator, marks a significant step forward in applying reinforcement learning to blockchain analytics.

Key limitations include dataset bias due to Flashbots' MEV-Inspect labels and inherent class imbalance, as well as a narrow focus on arbitrage transactions.

Future research should extend the classification framework to entire Ethereum blocks, investigate other MEV behaviors, and apply AIRL to block building. This would further validate reinforcement learning approaches within blockchain analytics, laying the groundwork for robust, scalable, and interpretable MEV detection methodologies.

Acknowledgments

I would like to express my deepest gratitude to Dr. Taehoon Kim for his exceptional mentorship, time, and commitment throughout this thesis. His insights, critical feedback, and availability for regular discussions were invaluable to both the conceptual development and technical execution of this work. I greatly appreciate the thoughtfulness and rigor he brought to each stage of the project.

I am also thankful to Prof. Dr. Claudio J. Tessone for his supervision and for making this thesis possible as part of the University of Zurich's Blockchain and Distributed Ledger Technologies (BDLT) group.

My thanks also go to Krzysztof Gogol for his co-supervision and administrative support, as well as for his helpful feedback during the final phase of the thesis.

Lastly, I would like to acknowledge the BDLT research group and my peers for providing a collaborative and intellectually stimulating environment during my studies. I am also sincerely grateful to my friends, family, and my partner for their unwavering encouragement and support throughout this academic journey.

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

Acronyms and Abbreviations

AIRL	Adversarial Inverse Reinforcement Learning
BDLT	Blockchain and Distributed Ledger Technologies
DeFi	Decentralized Finance
DEX	Decentralized Exchange
DGI	Deep Graph Infomax
DoS	Denial of Service
GAE	Generalized Advantage Estimation
GAIL	Generative Adversarial Imitation Learning
GAN	Generative Adversarial Network
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GF	Graph Factorization
GINE	Graph Isomorphism Networks with Edge features
GNN	Graph Neural Network
Graph RL	Graph Reinforcement Learning
GraphSAGE	Graph SAmple and aggreGatE
IRL	Inverse Reinforcement Learning
KL	Kullback-Leibler

LE Laplacian Eigenmaps

LINE Large-scale Information Network Embeddings

MDP Markov Decision Process

MEV Maximal Extractable Value

MINE Multi-agent Inverse models of Network Emergence

ML Machine Learning

MPNN Message-Passing Neural Network

PBS Proposer-Builder Separation

PGA Priority Gas Auctions

PoS Proof of Stake

PoW Proof of Work

PPO Proximal Policy Optimization

RF Random Forest

RL Reinforcement Learning

Chapter 1

Introduction

This chapter introduces the central research question of the thesis and motivates the need for new approaches to MEV detection on Ethereum. It begins by discussing the challenges and opportunities in identifying MEV behaviors, with a particular focus on arbitrage transactions. The chapter then summarizes the key contributions of this work and outlines the structure of the thesis.

1.1 Motivation

MEV has emerged as a central challenge in the economics of blockchain systems, particularly within the Ethereum ecosystem. Originating as ”Miner Extractable Value,” the term has since evolved into ”Maximal Extractable Value” following Ethereum’s shift to Proof-of-Stake, signaling a broader set of extraction opportunities now accessible to searchers, block builders, validators, and other participants in the block production process. MEV captures the additional value that can be extracted by influencing the order, inclusion, or exclusion of transactions within a block.

Among the various forms of MEV, arbitrage stands out as the most ubiquitous and reliably profitable. It typically exploits price discrepancies between Decentralized Exchanges (DEXs) through techniques such as simple loop arbitrage and burn-and-mint mechanisms (Park et al., 2024). These patterns, though economically impactful, are often subtle in structure and hard to detect with conventional tools. According to the EigenPhi dashboard (Eigenphi, 2022), between March 17 and April 14, 2025, arbitrageurs collectively extracted approximately $3.24 million in profit and facilitated over $3.49 billion in trading volume, underscoring both the scale and velocity of arbitrage activity on Ethereum.

Despite its economic significance, detecting arbitrage behavior remains a challenging task. Many detection pipelines rely on static heuristics, pattern-matching, and manual labeling based on known ABI-decoded logs, contract event signatures, or predefined token routes (Qin et al., 2022; Piet et al., 2022; Weintraub et al., 2022). These methods typically require access to verified contract ABIs

and full transaction traces, which limits scalability and generalizability. Moreover, evolving MEV strategies and the ever-changing Decentralized Finance (DeFi) ecosystem challenge the robustness of rule-based detection.

Building on prior work that employs GNNs for MEV detection (Park et al., 2024; Yao et al., 2025), this thesis explores the use of graph representation learning and Reinforcement Learning (RL), specifically AIRL with GNN-based state encoders, for detecting arbitrage behavior on Ethereum. Our method trains two AIRL agents independently on arbitrage and non-arbitrage trajectories, learning class-specific reward functions that reflect the underlying decision incentives of each group. For classification, unseen trajectories are evaluated under both learned reward functions and assigned the label of the one yielding the higher calibrated reward. This dual-agent approach is grounded in prior work by Fu et al. (2018), who show that AIRL learns a portable reward signal that generalizes across variations in environment dynamics. Unlike GAIL's discriminator, AIRL's structured reward captures decision quality rather than merely separating expert and agent behavior. We extend this insight to a two-class setting, interpreting reward magnitude as a comparative behavioral score. Inspired in part by Trivedi and Zha (2020), which learns distinct agent payoffs for strategic role inference, our method reuses AIRL rewards as discriminative classifiers over blockchain trajectories, yielding an interpretable, incentive-aligned mechanism for arbitrage detection.

By comparing this reward-driven formulation to traditional classifiers trained on tabular and graph-based features, we aim to answer the following research question:
How effective are Graph Reinforcement Learning techniques at detecting MEV-related transactions within dynamically evolving Ethereum transaction graphs?

To evaluate the effectiveness of this approach, we implement a multi-stage pipeline: local transaction graphs are constructed per account, encoded using GNNs, and passed to AIRL agents trained on arbitrage or non-arbitrage trajectories. These agents learn reward functions that reflect class-specific incentives, which are then calibrated and directly compared to classify new trajectories. This bridges imitation learning and classification, offering a unified, reward-based approach to MEV detection.

1.2 Contributions

This thesis introduces a novel framework for the detection of arbitrage-related MEV transactions on Ethereum, combining graph representation learning with reinforcement learning techniques. The key contributions are as follows:

- **A novel approach for MEV detection using graph reinforcement learning:** We extend prior work on MEV detection, such as Park et al. (2024) and Yao et al. (2025), by introducing a graph-based reinforcement learning framework utilizing AIRL. Our approach models transaction behavior dynamically through environment-agent interaction, enabling reward signals to emerge from imitation learning. This results in classifiers that are rooted in behavioral in-

centives rather than static features, and offers a domain-agnostic alternative to heuristic or ABI-dependent methods.

- **Repurposing AIRL reward functions for detection:** To the best of our knowledge, this is the first work to train separate AIRL agents for different transaction classes and calibrate their reward scores as stand-alone discriminative signals. We show that the resulting reward networks achieve high performance on downstream binary classification tasks, outperforming both the supervised GNN baseline and hybrid models, despite not being explicitly trained for this purpose.
- **Demonstrating the feasibility of Graph Reinforcement Learning for Ethereum transaction modeling:** Our results provide the first empirical evidence that GNNs, used as state encoders within a reinforcement learning loop, can capture meaningful patterns in Ethereum transaction graphs. This establishes a new methodological baseline for applying graph reinforcement learning to blockchain environments, where interactions are sparse, high-dimensional, and non-Euclidean.

Taken together, these contributions advance the state of MEV detection by introducing a learning-based alternative. They demonstrate that inverse reinforcement learning can simultaneously optimize agent behavior and generate discriminative reward signals for classification.

1.3 Thesis Outline

The structure of this thesis is organized to first build a conceptual foundation before progressively introducing and evaluating increasingly complex models for MEV detection, leading to a novel inverse reinforcement learning approach.

Chapter 2 reviews the background and related work relevant to this study. It begins with an overview of MEV, its historical development, ecosystem dynamics, and the taxonomy of known MEV strategies such as arbitrage, sandwich attacks, and liquidations. The chapter then discusses existing detection methodologies, contrasting rule-based systems dependent on ABI access with recent graph-based approaches such as the work by Park et al. (2024) and Yao et al. (2025). Following this, key principles in graph representation learning, including shallow and deep embedding techniques, are introduced. Finally, this chapter concludes with a discussion of RL paradigms—both forward and inverse—with a special focus on graph reinforcement learning.

In Chapter 3 the data collection process is described. It outlines the Ethereum transaction and account balance corpus that was created, as well as the MEV-labeled datasets provided by Flashbots. It details how this data was parsed, filtered, and labeled to enable both supervised learning and reinforcement learning experiments.

Chapter 4 introduces the methodology. It begins with the development of Random Forest baseline models trained on tabular features at the transaction, account, and graph levels. These baselines are followed by experiments with GNNs, in particular GraphSAGE, applied to per-transaction graph representations. A hybrid model that combines GNN embeddings with tabular features via a Random Forest classifier is also evaluated. The second half of the chapter presents the design and implementation of an AIRL framework tailored to the MEV setting. It includes dynamic graph construction, state embedding via GNNs, a custom environment modeling agent interactions with historical blockchain data, and the AIRL architecture for policy and reward learning.

Chapter 5 presents the empirical results. It reports classification performance for all baseline and GNN-based models and evaluates the effectiveness of the AIRL-trained policies. The chapter also introduces the results of a novel technique that repurposes AIRL reward signals as classifiers. Specifically, two separate AIRL agents are trained, one on arbitrage and one on non-arbitrage transactions, producing calibrated reward functions that are then compared to classify unseen trajectories based on which function assigns the higher reward.

Chapter 6 provides a critical discussion of the findings. It compares model performance across baselines, GNNs, and AIRL-based systems, and reflects on the strengths and limitations of graph-based RL in the MEV domain. The chapter also presents key implications for graph reinforcement learning, inverse reinforcement learning, and MEV detection, and identifies limitations related to dataset quality, scope, and action space complexity. Finally, it outlines directions for future work, including full-block classification, broader MEV strategy coverage, and AIRL-based block building.

Finally, Chapter 7 concludes with the key findings of the thesis, emphasizing the viability of AIRL-based classifiers and GNN-based state representations for MEV detection. It reflects on the broader relevance of these results and outlines directions for future work.

Chapter 2

Background and Related Work

This chapter reviews the foundations and prior work relevant to this thesis. It begins with an overview of MEV in Ethereum, followed by methods for its detection. The chapter then introduces key concepts in graph representation learning and reinforcement learning, concluding with recent advances in graph-based RL and inverse RL that inform the approach developed in this work.

2.1 Maximal Extractable Value

MEV refers to the value that can be extracted by privileged actors in the block production pipeline, such as miners, validators, searchers, or builders, through the strategic ordering, insertion, or suppression of transactions within a block (Daian et al., 2020; Gramlich et al., 2024). While originally studied in the context of Ethereum, MEV is a general phenomenon observed across smart contract platforms, including Solana, Polygon, and Cosmos. It emerges wherever transaction ordering influences state transitions, especially in financial applications like DEXs, lending markets, and auction mechanisms.

2.1.1 Maximal Extractable Value on Ethereum

MEV has become one of the most studied and economically significant phenomena on Ethereum. The term was originally introduced by Daian et al. (2020) as *Miner Extractable Value*, referring to the profits miners could gain by arbitrarily including, excluding, or reordering transactions within a block. Following Ethereum's transition to Proof of Stake (PoS), the term evolved to *Maximal Extractable Value* to reflect that such privileges extend beyond miners to validators, sequencers, and other parties involved in block production.

According to data from the Flashbots Explorer (Flashbots, 2025c), over $675 million in MEV was extracted on Ethereum between 2020 and 2022, through strategies such as arbitrage, sandwich at-

tacks, and liquidations. Sandwich attacks accounted for more than 700,000 transactions, extracting over $170 million, primarily on protocols such as Uniswap, Sushiswap, and Bancor (Gramlich et al., 2024).

The transparent nature of Ethereum's mempool and the programmability of smart contracts create an environment where actors can observe and react to pending transactions, executing strategies with minimal risk. While some MEV strategies like arbitrage may enhance market efficiency, others, such as front-running or suppression attacks, can degrade user experience, inflate gas prices, and introduce systemic risks (Daian et al., 2020).

To address these challenges, the Ethereum community has developed infrastructure such as Flashbots (Flashbots, 2025d), which provides off-chain MEV auctions and private transaction submission to reduce harmful extraction behaviors. However, MEV continues to evolve rapidly, and Ethereum remains a central case study for its analysis and mitigation.

2.1.2 A Brief History of MEV on Ethereum

In Ethereum's early years under Proof of Work (PoW), MEV was primarily captured by miners or bots by submitting transactions with high gas fees to outbid competitors for favorable placement within blocks. This process, known as Priority Gas Auctions (PGA), created competitive bidding wars where searchers raced to have their MEV-exploiting transactions mined first (Gosselin, 2020). Miners had full control over transaction ordering and could either extract MEV themselves or include the highest-paying bundles. As MEV opportunities grew, especially in decentralized exchanges and lending protocols, projects such as Flashbots emerged to create more transparent and permissionless access to MEV through private transaction bundles.

The release of "Flashbots: Frontrunning the MEV Crisis" (Gosselin, 2020) as well as the publication of "Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges" by Daian et al. (2020) highlighted the existential risks MEV posed to Ethereum's consensus layer. These included time-bandit attacks, centralization of transaction routing, and inefficient blockspace usage. Flashbots introduced two key infrastructure components: *MEV-Inspect*, to quantify and visualize MEV extraction, and *MEV-Geth*, a sealed-bid blockspace auction prototype for miners.

MEV-Geth allowed for permissionless participation by searchers who submitted bundles to miners through a dedicated RPC interface. This reduced reliance on the public pool and mitigated front-running. Although MEV-Geth was only a proof of concept, it paved the way for future Flashbots infrastructure, and by April 2021, mining pools accounting for more than 84% of the Ethereum hashrate had adopted MEV-Geth (Flashbots, 2025d).

In 2022, Buterin (2022) introduced *Proposer-Builder Separation (PBS)*, a concept to decouple the roles of block proposal and block construction, thereby reducing the power and complexity required by validators and enabling a more competitive, decentralized block-building market. As outlined by

Buterin (2022), PBS proposes a two-tier system in which specialized actors called *builders* construct execution payloads and bid to have their blocks included, while *proposers* (validators) simply select the most profitable bid without visibility into the execution payload contents before selection. This design introduces pre-confirmation privacy to mitigate MEV stealing and reduce validator centralization.

The ultimate goal is to implement PBS in-protocol, removing the need for trusted intermediaries and enabling native support for this separation within the Ethereum consensus mechanism. However, due to the complexity of fully integrating PBS at the protocol level, an interim off-chain solution called *MEV-Boost* was deployed. Flashbots introduced MEV-Boost as a production-ready realization of PBS outside of the core Ethereum protocol (Smedley, 2022) following Ethereum's transition to PoS (the Merge) in September 2022. MEV-Boost enables validators to outsource block building to an open marketplace of builders, coordinated through *relays* that act as intermediaries. This architecture introduces a dual-auction dynamic: searchers compete for inclusion within builder blocks, and builders compete to have their blocks selected by validators (see Section 2.1.3 for more details).

MEV-Boost preserves the PBS design principles of minimizing validator complexity and decentralizing block production. Although it improves censorship resistance relative to monolithic builders, the use of relays and lack of full protocol integration still introduce centralization risks and trusted assumptions.

2.1.3 The MEV Ecosystem

The MEV supply chain has evolved into a complex ecosystem composed of multiple specialized roles:

- **Searchers** are specialized actors who monitor the blockchain state and mempool to detect MEV opportunities such as arbitrage, liquidation, and sandwich attacks. Rather than submitting transactions to the public mempool, searchers use the Flashbots private transaction pool to ensure pre-trade privacy. They construct *bundles* (ordered sets of transactions) and submit them directly to builders, often conditioning their bids using direct ETH transfers instead of gas fees to avoid paying for failed inclusion. Searchers range from DeFi bot operators and MEV extractors to users and dApps requiring frontrun protection or custom execution semantics (Flashbots, 2025e).

- **Builders** are specialized actors responsible for constructing full execution payloads (blocks) by aggregating and ordering public transactions, private transactions, and searcher-submitted bundles. They run sophisticated algorithms and simulations to maximize block profitability, often using strategies like First-Price Auctions or First-Come-First-Served. Builders bid for validator blockspace via relays, proposing the most valuable block they can assemble. Payments to validators are made through an in-block transaction, with the builder initially setting them-

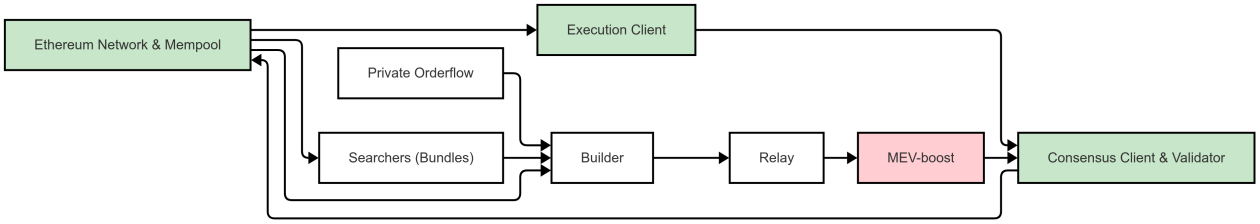


Figure 2.1 – Visualization of the MEV supply chain in Ethereum, highlighting the interaction between searchers, builders, relays, and validators under the MEV-Boost framework. Adapted from Flashbots documentation (Flashbots, 2022).

selves as the fee recipient and later transferring ETH to the validator's fee recipient address. Block value is typically determined via block-level scoring, which measures the net balance increase of the validator's address over a single block, providing a simple and effective way to assess builder payments and block profitability (Flashbots, 2025a).

- **Relays** act as intermediaries between builders and validators, forming a doubly-trusted layer in the MEV supply chain. Builders trust relays to fairly aggregate and forward their payloads without censorship or manipulation, while validators trust relays to verify block validity, simulate transactions, and ensure data availability. Relays receive full blocks from one or many builders, verify their correctness (e.g., fee payments, gas limits, and transaction ordering), and submit only the highest-bidding valid block to the validator. To protect against frontrunning, the block contents remain encrypted until signed by the validator. Additionally, relays are often optimized for Denial of Service (DoS) resistance and are responsible for enforcing rate limits and maintaining the integrity of block proposals (Flashbots, 2025f).

- **Validators** (formerly miners in PoW) are block proposers selected pseudorandomly per slot using the Ethereum beacon chain's RANDAO mechanism. Their role is to propose a block for the current slot, either by locally constructing it or by selecting one built by external builders. With *MEV-Boost*, validators are relieved of block-building duties and instead receive multiple blinded block proposals (execution payload headers) from connected relays. They sign and propose the most profitable one without knowing the full transaction contents, preserving preconfirmation privacy. Validators are rewarded for including high-value blocks and for acting honestly according to Ethereum's consensus specifications (e.g., Bellatrix's Honest Validator rules) (Flashbots, 2025b).

This separation of roles enhances decentralization and mitigates the censorship risks previously associated with single-actor block production. However, as noted by Smedley (2022), it introduces new strategic considerations for searchers, including builder selection, bid optimization, and bundle placement. Figure 2.1 illustrates the architecture of this ecosystem, showing how MEV flows from searchers to validators through builders and relays.

2.1.4 Types of MEV

Based on the consolidated literature (Gramlich et al., 2024; Park et al., 2024), MEV strategies can be categorized into six primary types, each reflecting distinct economic behaviors and implications for blockchain security and user welfare.

- **Arbitrage:** Arbitrage is the most prevalent and consistently profitable form of MEV. According to the EigenPhi dashboard (Eigenphi, 2022), arbitrageurs generated approximately $3.24 million in profits and facilitated over $3.49 billion in trading volume between March 17 and April 14, 2025 (30 days). Arbitrage typically exploits temporary price discrepancies between DEXs, often by back-running large trades or oracle updates that momentarily distort pricing. While arbitrage contributes to market efficiency by aligning prices across venues, it is also a highly competitive space dominated by latency-sensitive bots.
- **Liquidations:** In lending protocols, if a borrower's collateral falls below a threshold, liquidators can repay a portion of the debt and seize collateral at a discount. This creates MEV opportunities, especially when the liquidation is initiated via front- or back-running. Liquidations are crucial for maintaining the health of DeFi protocols.
- **Front-running:** This strategy involves observing a profitable transaction in the mempool and inserting a new transaction ahead of it, typically through higher gas fees or private relays. Front-running includes transaction imitation (copying a profitable action with minor edits) and is common in arbitrage, liquidation, and smart contract exploit contexts. The original transaction may fail or become economically irrelevant as a result.
- **Back-running:** Back-running follows a profitable transaction to exploit the state change it causes. Common scenarios include reacting to an AMM price shift or a lending protocol's oracle update. Unlike front-running, it does not harm the original transaction but benefits from its successful execution.
- **Sandwich Attacks:** A sandwich attack surrounds a victim's trade with a front-run and back-run transaction by the attacker. The front-run increases slippage, the victim's trade is executed at a worse rate, and the attacker back-runs to lock in profit. Sandwiching is prevalent in AMMs and exploits user-specified slippage tolerances. It is among the most discussed and measured MEV types due to its high transaction count and economic impact.
- **Suppression:** Suppression attacks (also called clogging or block stuffing) prevent a competitor's transaction from being included by flooding the block with high-fee or spam transactions. This is seen in lottery-based smart contracts and high-stakes contests. Though rare, suppression has historically been used in attacks lasting dozens of blocks with high cumulative gas costs.

While arbitrage and liquidation are often protocol-aligned behaviors, front-running, sandwich attacks, and suppression impose direct harm on users or the network, necessitating ongoing detection and mitigation efforts.

2.1.5 MEV Detection

While MEV activity has been widely studied, the practical detection of MEV remains an evolving challenge. Qin et al. (2022) introduced a systematic taxonomy of MEV, classifying transactions into arbitrage, sandwich attacks, and liquidations, and proposed heuristics based on token transfer graphs and DEX event logs. Their detection method required ABI access and full historical traces from an Ethereum archive node. Piet et al. (2022) developed a graph-based algorithm to detect arbitrage, frontrunning, and backrunning in historical data, identifying over $6 million in MEV profits across 12 days, 87.6% of which were extracted via private transactions, and demonstrating that miners claimed the majority of this value. Weintraub et al. (2022) further contributed a public detection pipeline tailored for private pools, including Flashbots, and emphasized the distinction between public and private MEV opportunities, highlighting the opacity introduced by off-chain infrastructure like relays and bundles.

These detection approaches typically follow a two-stage process: first, decoding transactions into token transfer graphs using known contract ABIs, and second, applying rule-based heuristics to identify MEV behaviors such as arbitrage, sandwich attacks, or liquidations. For example, sandwich detection often involves identifying transaction patterns like attacker-victim-attacker swaps within DEXs, while arbitrage detection seeks profit-generating cycles across pools. Liquidations, in contrast, are usually identifiable through dedicated smart contract events.

Despite their contributions, these methods suffer from three major limitations, as previously shown by Park et al. (2024):

- **Dependency:** Reliance on centralized ABI sources such as Etherscan poses a bottleneck. Contracts not verified on Etherscan render detection infeasible, and decompilation tools often produce incomplete ABIs.
- **Maintenance:** The rapid proliferation of DeFi protocols (for example, more than 1,150 on Ethereum according to DeFiLlama (DefiLlama, 2025)) increases the burden of maintaining up-to-date ABI databases and heuristics tailored to new platforms.
- **Low Recall:** Heuristic methods miss non-standard patterns, including MEV actions involving staking contracts, vault protocols, index tokens, or synthetic assets, especially when events are not explicitly emitted.

Recent developments in MEV detection have focused on GNN-based frameworks designed to overcome the scalability, generalizability, and dependency limitations of heuristic-based approaches.

Mecon (Yao et al., 2025) constructs address-centric behavior graphs from labeled transaction data, sourced via Flashbots through the Dune platform. Each graph encodes transactional relationships and behaviors between addresses, capturing both statistical and structural features. These include metrics like interaction frequency and Pearson correlations between address behaviors. Mecon applies a two-layer Graph Convolutional Network (GCN) (Kipf and Welling, 2017) to these graphs for classification and achieves a high F1 score of 94.33%, significantly outperforming baselines such as XGBoost (82.99%).

ArbiNet (Park et al., 2024) introduces a similar but ABI-free detection pipeline using token transfer graphs as input. Each transaction is converted into a graph where nodes are addresses, and edges represent ERC20 transfers. A set of 14 node features—covering profits, token activity, and address metadata—is extracted and fed into GCN, GraphSAGE (Hamilton et al., 2017), or Graph Attention Networks (GATs) (Veličković et al., 2018) layers. The models are trained to classify arbitrage transactions without relying on event logs or contract ABIs. Evaluation over Ethereum blocks 15,540,000 to 15,585,000 shows that ArbiNet achieves an F1-score of 98.54%, demonstrating robustness against ABI incompleteness and novel MEV strategies.

Together, these models illustrate a shift toward scalable, learning-based MEV detection systems that capture complex behavioral and structural patterns across diverse DeFi protocols.

2.2 Graph Representation Learning

Graph-structured data arises naturally in many domains where entities interact in complex, relational ways. Traditional machine learning models, which assume independent and identically distributed (i.i.d.) inputs, fail to capture such dependencies. Graph Representation Learning provides a framework to embed nodes, edges, or entire graphs into low-dimensional vector spaces while preserving structural and feature-based information. This section introduces key graph learning paradigms and modeling techniques, laying the foundation for the graph classification approach used in this thesis.

2.2.1 Graphs and Graph-Structured Data

Graphs are a fundamental data structure used to represent complex systems by modeling relationships between entities. Formally, a graph is defined as $G = (V, E)$, where V is a set of nodes (also called vertices), and E is a set of edges representing interactions between pairs of nodes. An edge between nodes $u, v \in V$ is denoted as $(u, v) \in E$.

Graphs can be:

- **Undirected**, where edges are bidirectional and $(u, v) \in E$ implies $(v, u) \in E$.
- **Directed**, where edges have a direction from u to v.

- **Weighted**, where edges have associated real-valued weights.

Graphs are often represented by an adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$ where $A[u, v] = 1$ if $(u, v) \in E$ and 0 otherwise. In weighted graphs, $A[u, v]$ holds the corresponding edge weight.

Graphs may also contain:

- **Node features**, stored in a matrix $X \in \mathbb{R}^{|V|\times m}$ where each row contains a feature vector for a node.
- **Multi-relational edges**, denoted (u, τ, v) with edge type $\tau \in \mathcal{R}$, captured by a tensor $A \in \mathbb{R}^{|V|\times|\mathcal{R}|\times|V|}$.
- **Heterogeneous node types**, where $V = V_1 \cup V_2 \cup \cdots \cup V_k$ with disjoint subsets representing different node categories.

This generalized graph structure is versatile and supports applications in social networks, biology, chemistry, and more (Hamilton, 2022).

2.2.2 Machine Learning on Graphs

Machine learning (ML) on graphs encompasses a variety of tasks that differ from traditional ML due to the relational and non-i.i.d. structure of graph data. Graph-based tasks are often categorized based on the granularity of prediction, whether at the node, edge, or graph level. We outline the main paradigms of graph learning, as follows Hamilton (2022):

- **Node classification:** The goal of node classification is to predict labels for nodes in a graph given a subset of labeled nodes. This task appears frequently in social network analysis (e.g., bot detection), biology (e.g., protein function prediction), and document classification. Unlike standard supervised learning, node classification violates the i.i.d. assumption because nodes are interdependent through the graph structure. Effective methods thus exploit *homophily* (e.g., friends tend to share attributes) or *structural similarity* (e.g., nodes with similar roles in different neighborhoods).
- **Relation prediction:** Also known as link prediction or graph completion, this task involves inferring missing edges between nodes. For example, predicting protein-protein interactions or recommending new connections in a social network. Depending on the graph, models must handle varying degrees of relational complexity and inductive generalization, particularly in multi-relational or dynamic graphs.
- **Community detection or clustering:** These are unsupervised tasks that aim to group nodes or subgraphs based on connectivity patterns. Community detection identifies tightly-knit clusters (e.g., research groups in a co-authorship graph), while clustering extends this idea across entire datasets. These tasks are central to network science and fraud detection.

- **Graph classification:** Here, each input is a whole graph (e.g., molecular structure, program syntax tree), and the goal is to predict a graph-level label or property such as toxicity or malware presence. This setup closely resembles classical supervised learning, with the additional challenge of learning meaningful graph-level representations. Graph regression and classification tasks are especially relevant in chemistry, cybersecurity, and finance.

In this work, we focus on the task of *graph classification*, where the goal is to classify entire Ethereum transactions, represented as graphs created from transaction logs, based on whether they involve arbitrage behavior.

2.2.3 Shallow Embedding Methods

Shallow embedding approaches learn a fixed vector representation for each node in a graph by optimizing an unsupervised objective over the graph structure. These methods do not parameterize neighborhood aggregation functions like GNNs, but instead treat embeddings as lookup-table parameters that are learned directly. They are conceptually simpler and often efficient for static graphs, but limited in generalization capacity. Common shallow methods include matrix factorization and random walk-based techniques (Hamilton, 2022).

Matrix Factorization Embeddings

Early approaches to graph representation learning framed the problem as matrix factorization, where the goal is to approximate a node-node similarity matrix S using low-dimensional embeddings. These methods aim to preserve global graph structure by minimizing the reconstruction error of S, often viewed as a generalized adjacency matrix.

- **Laplacian Eigenmaps (LE):** Belkin and Niyogi (2001) define similarity based on local proximity in the graph and use an L2-based decoder $\text{dec}(z_u, z_v) = \|z_u - z_v\|_2^2$. The loss function penalizes embeddings of similar nodes that are far apart. When S is the graph Laplacian, minimizing this loss recovers the bottom d eigenvectors of the Laplacian, equivalent to spectral clustering. This method is foundational but not scalable to large graphs.

- **Inner-Product Methods:** More recent work in factorization-based models generally uses an inner-product decoder $\text{dec}(z_u, z_v) = z_u^\top z_v$, aiming to approximate a similarity matrix S. Methods such as Graph Factorization (GF) (Ahmed et al., 2013), GraRep (Cao et al., 2015), and HOPE (Ou et al., 2016) fall into this category. They differ primarily in how they define S: GF uses the adjacency matrix, GraRep incorporates higher-order proximities via powers of A, and HOPE supports asymmetric similarity measures. All optimize a reconstruction loss $\|ZZ^\top - S\|_F^2$, interpreting graph embedding as a low-rank approximation.

All these methods fall under the umbrella of shallow factorization-based models, where embeddings $Z \in \mathbb{R}^{|V| \times d}$ are learned such that $ZZ^\top \approx S$, minimizing the Frobenius norm $\|ZZ^\top - S\|_F^2$. Despite their interpretability and theoretical foundation, these methods struggle with scalability and generalization to unseen nodes or graphs (Hamilton, 2022).

Random Walk Embeddings

A significant advancement in shallow methods came with the introduction of random walk-based embeddings. These methods generate sequences of nodes via truncated random walks and apply the Skip-Gram model (originally from natural language processing) to learn embeddings that capture co-occurrence in these walks.

- **DeepWalk** (Perozzi et al., 2014) performs truncated random walks on the graph to generate sequences of nodes, treating these sequences similarly to sentences in natural language. It then applies the Skip-Gram model to learn node embeddings that capture the co-occurrence patterns within these walks.
- **node2vec** (Grover and Leskovec, 2016) extends DeepWalk by introducing a biased random walk procedure with parameters that control the trade-off between breadth-first and depth-first search strategies, allowing the embeddings to capture both community structures and structural roles.
- **Large-scale information network embeddings (LINE)** (Tang et al., 2015) aims to preserve both first-order (direct connections) and second-order (shared neighbors) proximities in the embedding space. It defines separate objectives for each and combines them to learn embeddings that reflect both local and global network structures.

These methods are efficient and scalable to large graphs. However, they are typically transductive, meaning they cannot naturally generalize to unseen nodes or graphs without retraining. Additionally, they often do not incorporate node or edge attributes directly into the embedding process.

Limitations of Shallow Embeddings

While shallow embedding and matrix factorization methods have advanced the field, they have notable limitations:

- **Transductive Nature:** These methods typically cannot handle new nodes or graphs without retraining.
- **Limited Incorporation of Attributes:** Node and edge features are often not directly integrated into the embedding process.

- **Fixed Embedding Size:** They produce embeddings of a fixed size, which may not capture the varying complexities of different parts of the graph.

These limitations have led to the development of deep learning approaches, particularly GNNs, which address these challenges by learning functions that can generalize across different graphs and naturally incorporate node and edge attributes. Section 2.2.4 will delve into the principles and architectures of GNNs.

2.2.4 Graph Neural Networks

Graph Neural Networks, originally proposed by Merkwirth and Lengauer (2005) and Scarselli et al. (2009), represent a class of neural architectures designed to operate directly on graph-structured data. Unlike shallow methods that decouple feature extraction from model learning, GNNs unify the two in an end-to-end framework. GNNs fundamentally operate within the *Message-Passing Neural Network (MPNN)* framework (Gilmer et al., 2017). It structures the computation in three phases: message aggregation, node state update, and a readout phase. This framework has become a unifying abstraction for a wide range of GNN architectures (Hamilton, 2022).

Formally, the general MPNN update rule for a node u at layer $k+1$ is defined as:

$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})\right) \tag{2.1}$$

$$= \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right), \tag{2.2}$$

where `UPDATE` and `AGGREGATE` represent differentiable functions (such as neural networks) and $m_{N(u)}$ denotes the "message" collected from the neighborhood $N(u)$ of node u in the graph (Hamilton, 2022). Superscripts help differentiate the embeddings and functions across various iterations of the message passing process. A concrete instantiation of the general GNN framework in Equation (2.1), inspired by early models (Merkwirth and Lengauer, 2005; Scarselli et al., 2009), computes the node update at layer k as a weighted combination of the node's previous embedding and the aggregated embeddings of its neighbors, followed by a non-linear activation.

$$\mathbf{h}_u^{(k)} = \sigma\left(W_{\text{self}}^{(k)}\mathbf{h}_u^{(k-1)} + W_{\text{neigh}}^{(k)}\sum_{v\in\mathcal{N}(u)}\mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}\right), \tag{2.3}$$

where $\mathbf{h}_u^{(k)}$ is the embedding of node u at layer k, $W_{\text{self}}^{(k)}$ and $W_{\text{neigh}}^{(k)}$ are learnable weight matrices, $\mathbf{b}^{(k)}$ is a bias term, $\mathcal{N}(u)$ denotes the neighborhood of node u, and σ is a non-linear activation function such as ReLU or tanh.

A popular baseline GNN, the Graph Convolutional Network, aggregates features by normalizing

neighbor embeddings symmetrically to stabilize training and improve convergence:

$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}^{(k)} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{\mathbf{h}_v^{(k-1)}}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}\right). \tag{2.4}$$

GraphSAGE (SAmple and aggreGatE) introduced inductive capabilities by learning functions that aggregate features from a sampled neighborhood, enabling generalization to unseen nodes. It supports various aggregation strategies such as mean, LSTM-based, or pooling.

Graph Attention Networks improve flexibility by assigning learned attention weights to neighbors during aggregation. This allows the model to focus on more relevant parts of a neighborhood, addressing cases where uniform weighting is suboptimal.

Despite their success, GNNs face challenges such as over-smoothing. Recent work addresses this with techniques such as jumping knowledge (Xu et al., 2018).

Overall, GNNs offer a powerful framework for graph-based learning, enabling representation learning directly from raw relational structures, and have been widely adopted in domains such as social networks, recommendation systems, and, as in our work, blockchain transaction analysis.

2.3 Reinforcement Learning

Reinforcement Learning provides a flexible framework for decision-making under uncertainty, enabling agents to learn optimal behaviors through interaction with an environment. This section introduces foundational concepts from standard (forward) RL and its extension, Inverse Reinforcement Learning (IRL), which focuses on learning reward functions from expert demonstrations. We conclude with Graph Reinforcement Learning (Graph RL), a recent paradigm that applies RL to graph-structured domains. Together, these methods provide the theoretical basis for modeling and analyzing strategic behavior in blockchain transaction graphs, which is central to the objectives of this thesis.

2.3.1 (Forward) Reinforcement Learning

RL is a framework for sequential decision-making in which an agent learns to make decisions through interaction with an environment to maximize cumulative reward over time (Sutton and Barto, 2018). In contrast to supervised learning, where labeled input-output pairs guide learning directly, RL agents must discover effective behavior through trial-and-error and delayed feedback.

Markov Decision Processes

Most RL problems are formalized as a Markov Decision Process (MDP) (Sutton and Barto, 2018), described by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$:

- $\mathcal{S}$: a set of environment states called the state space,
- $\mathcal{A}$: a set of actions called the action space,
- $P(s' \mid s, a)$: the probability of transitioning to state s' after taking action a in state s,
- $r(s, a)$: the immediate (expected) reward received for taking action a in state s,
- $\gamma \in [0, 1]$: a discount factor that prioritizes immediate over future rewards.

The agent's behavior is defined by a stochastic policy $\pi(a \mid s)$, which represents the probability of taking action a in state s. The goal is to find an optimal policy π^* that maximizes the expected cumulative discounted reward, also called the return:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r(s_{t+k+1}, a_{t+k+1}). \tag{2.5}$$

Value Functions and Advantage Estimation

To evaluate and improve policies, RL algorithms often rely on value functions. The state-value function $V^\pi(s)$ is the expected return starting from state s under policy π:

$$V^\pi(s) = \mathbb{E}_\pi \left[G_t \mid s_t = s \right]. \tag{2.6}$$

The action-value function $Q^\pi(s, a)$ extends this to state-action pairs:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid s_t = s, a_t = a \right]. \tag{2.7}$$

The advantage function measures the relative value of an action compared to the state value:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s). \tag{2.8}$$

Advantage estimation is central to many modern policy optimization algorithms, as it provides a low-variance estimate for improving policies (Sutton and Barto, 2018).

Policy Gradient Methods

Rather than learning value functions explicitly, policy gradient methods directly optimize the policy by maximizing the expected return. The policy gradient theorem provides a foundation for this:

$$\nabla J(\theta) \propto \sum_{s} \mu(s) \sum_{a} q_\pi(s,a) \nabla \pi(a \mid s, \theta), \tag{2.9}$$

where θ are the parameters of the stochastic policy $\pi(a \mid s, \theta)$ and $\mu(s)$ denotes the (on-policy) state distribution induced by π. This formulation, known as the policy gradient theorem (Sutton and Barto, 2018), underpins actor-critic methods by enabling policy improvement through gradient ascent using estimated value functions.

Actor-Critic Architectures and Generalized Advantage Estimation

Actor-critic methods combine the strengths of value-based and policy-based approaches. The actor updates the policy using gradients computed from estimated advantages, while the critic learns a value function to support those estimates.

One powerful technique used in this setting is *Generalized Advantage Estimation* (GAE), which provides a tunable bias-variance tradeoff in advantage estimates:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \tag{2.10}$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ is the temporal difference error, and λ is a smoothing parameter (Schulman, Moritz, et al., 2016).

A widely used practical algorithm that builds on the actor-critic framework is *Proximal Policy Optimization (PPO)* (Schulman, Wolski, et al., 2017). PPO introduces a clipped surrogate objective to constrain the policy update, striking a balance between learning efficiency and stability. Instead of taking large steps that might degrade performance, PPO penalizes updates that move the new policy too far from the old one. The objective is defined as:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right], \tag{2.11}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new and old policies, $\hat{A}_t$ is an estimate of the advantage function, and ϵ is a hyperparameter controlling the update range. The full PPO algorithm is shown in Algorithm 1.

Due to its robustness and sample efficiency, PPO is commonly used in imitation learning and inverse reinforcement learning settings. In this thesis, we use PPO as the policy optimization method within our Adversarial Inverse Reinforcement Learning framework to iteratively improve the agent's policy

Algorithm 1 PPO, Actor-Critic Style (Schulman, Wolski, et al., 2017)

for iteration $= 1, 2, \ldots$ **do**
  for actor $= 1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L w.r.t. θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

based on learned reward signals (see Section 2.3.2).

On-policy vs. Off-policy Learning

A final distinction in RL is between *on-policy* and *off-policy* methods. On-policy methods update the policy using data collected from the current version of the policy itself. Off-policy methods, by contrast, use a replay buffer to learn from past experiences, often collected under different policies. This distinction has practical implications for sample efficiency and training stability (Sutton and Barto, 2018).

The concepts above—MDPs, value functions, advantage estimation, and policy gradients—form the theoretical basis of most modern RL algorithms. They are particularly relevant when using policy optimization frameworks in environments modeled as decision processes, including the blockchain transaction graphs explored in this thesis.

2.3.2 Inverse Reinforcement Learning.

While standard reinforcement learning focuses on learning a policy to maximize cumulative rewards in a given environment, *IRL* inverts this paradigm: instead of learning a policy from a known reward, the goal is to recover the reward function that an expert is implicitly optimizing, based on observed behavior (Russell, 1998; Ng and Russell, 2000). IRL provides a principled framework for learning from demonstrations in settings where designing a reward function is difficult or ill-defined. IRL offers interpretability and generalizability, especially in applications that require transferring behavior across environments or inferring agent intent (Finn, Levine, et al., 2016; Finn, Yu, et al., 2017).

Maximum Entropy IRL

The foundational work by Abbeel and Ng (2004) introduced IRL in the context of *apprenticeship learning*, formalizing it as the task of recovering a reward function under which the expert's policy is (near-)optimal. The proposed algorithm optimizes a reward function such that the learned policy

matches the feature expectations of the expert, using linear function approximation and iterative projection methods. Given and $MDP \setminus R$, and a linear reward function $R(s) = w^\top \phi(s)$, where $\phi(s)$ is a feature vector, the expected return under a policy π becomes:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi\right] = w^\top \mu(\pi), \tag{2.12}$$

where $\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \pi\right] \in \mathbb{R}^k$ is the feature expectation. The objective is to find a policy π such that:

$$\|w^T \mu(\pi) - w^T \mu_E\|_2 \leq \epsilon, \tag{2.13}$$

ensuring near-optimality for all reward functions w with bounded norm $\|w\|_1 \leq 1$.

To disambiguate among multiple reward functions consistent with expert behavior, Ziebart et al. (2008) introduced the *principle of maximum entropy* (Jaynes, 1957) to the IRL framework. This approach prefers stochastic policies that match expert feature expectations while remaining as uncertain (high entropy) as possible. It defines a distribution over trajectories such that higher-reward trajectories are exponentially more likely:

$$P(\tau \mid w) = \frac{1}{Z(w)} \exp\left(w^\top \phi(\tau)\right), \tag{2.14}$$

where $\phi(\tau) = \sum_t \phi(s_t) \in \mathbb{R}^k$ is the summed feature vector along trajectory τ, and $Z(w)$ is the partition function ensuring normalization.

The model enforces that the expected features under the learned distribution match those of the expert:

$$\sum_{\tau} P(\tau \mid w)\phi(\tau) = \mu_E. \tag{2.15}$$

The log-likelihood gradient used to optimize the reward weights w is given by:

$$\nabla_w \mathcal{L} = \mu_E - \sum_{\tau} P(\tau \mid w)\phi(\tau), \tag{2.16}$$

which drives learning by aligning the expected features of the induced distribution with the empirical expert features.

Wulfmeier et al. (2015) extended maximum entropy IRL to non-linear reward functions by using deep neural networks. This enables learning complex, high-dimensional reward functions directly from raw input features such as images. The method preserves the probabilistic formulation of maximum entropy IRL, while leveraging the representation power of deep learning.

Adverserial IRL

Inspired by Generative Adversarial Networks (GANs), Ho and Ermon (2016) proposed *Generative Adversarial Imitation Learning (GAIL)*, which bypasses reward recovery altogether. GAIL trains a generator (policy) to produce trajectories indistinguishable from expert demonstrations, using a discriminator to guide the learning signal. This results in a learned policy that imitates expert behavior without requiring an explicit reward function.

Fu et al. (2018) extends the framework of imitation learning by proposing *Adversarial Inverse Reinforcement Learning*, a method that unifies the maximum entropy IRL framework with adversarial training. Unlike GAIL, which learns a policy without recovering a reward function, AIRL is explicitly designed to learn reward functions that generalize across environments with different dynamics.

AIRL formulates a discriminator $D_{\theta,\phi}(s, a, s')$ that distinguishes expert from agent-generated state-action-next-state triplets. The reward signal is derived from the discriminator's output via the shaped reward:

$$r_{\theta,\phi}(s, a, s') = \log D_{\theta,\phi}(s, a, s') - \log(1 - D_{\theta,\phi}(s, a, s')). \tag{2.17}$$

The discriminator is defined in terms of a logit function $f_{\theta,\phi}$, structured to reflect the advantage function:

$$f_{\theta,\phi}(s, a, s') = g_\theta(s, a) + \gamma h_\phi(s') - h_\phi(s), \tag{2.18}$$

where g_θ serves as the reward approximation and h_ϕ is the shaping term. By enforcing that the reward g_θ depends only on state, AIRL disentangles the reward from the environment dynamics, making it transferable across different MDPs. The full algorithm is detailed in Algorithm 2.

Algorithm 2 Adversarial Inverse Reinforcement Learning (Fu et al., 2018)

1: **Input:** Expert trajectories τ_i^E
2: Initialize policy π and discriminator $D_{\theta,\phi}$
3: **for** step $t \in \{1, \dots, N\}$ **do**
4: Collect trajectories $\tau_i = (s_0, a_0, \dots, s_T, a_T)$ by executing π
5: Train $D_{\theta,\phi}$ via binary logistic regression to classify expert data τ_i^E from samples τ_i
6: Update reward: $r_{\theta,\phi}(s, a, s') \leftarrow \log D_{\theta,\phi}(s, a, s') - \log(1 - D_{\theta,\phi}(s, a, s'))$
7: Update π with respect to $r_{\theta,\phi}$ using any policy optimization method
8: **end for**

In the optimal case, the logit function $f_{\theta,\phi}$ recovers the true advantage function:

$$f^*(s, a, s') = Q^*(s, a) - V^*(s), \tag{2.19}$$

such that $g^*(s)$ approximates the true state-based reward and $h^*(s)$ the value function.

AIRL retains the stability and expressivity of adversarial training while recovering a usable reward function. It has been shown to match GAIL in terms of imitation performance in the training envi-

ronment, and significantly outperform it in transfer settings where the test-time dynamics differ from those of the expert. In this thesis, AIRL forms the foundation for reward learning from Ethereum transaction graphs, enabling us to detect MEV-related transactions.

As summarized in Arora and Doshi (2021), IRL methods vary in their assumptions (e.g., known dynamics, stochasticity), scalability, and ability to generalize. While early approaches focused on tabular MDPs, modern methods scale to complex settings trough deep learning and adversarial training. Key challenges remain in sample efficiency, interpretability of learned rewards, and robustness to imperfect demonstrations.

2.3.3 Graph Reinforcement Learning

Graph reinforcement learning is an emerging paradigm that unifies reinforcement learning and graph-structured decision problems, particularly those involving combinatorial optimization. In this framework, problems defined over discrete graph structures are cast as Markov Decision Processes, enabling the use of trial-and-error learning to derive efficient solution strategies (Darvariu et al., 2024).

Graph RL problems fall into two main categories, as defined by Darvariu et al. (2024):

- **Structure Optimization**, where the goal is to construct or modify a graph $G \in \mathcal{G}$ such that an objective function $F(G)$ is maximized. Applications include network design, molecular generation, and causal discovery.
- **Process Optimization**, where the graph structure is fixed, and the task is to learn control policies that optimize a dynamic process on the graph. Examples include network routing, network games, and search and navigation tasks.

In both settings, RL provides a flexible and general-purpose method to learn policies that outperform traditional heuristics, especially in non-canonical graph problems where no satisfactory exact or approximate algorithms are known. The RL agent interacts with an environment defined by the graph, receives feedback via a reward signal, and adjusts its policy to maximize long-term returns.

Graph RL methods often leverage GNNs as function approximators for value functions or policies as well as for state representation. This integration allows for parameter sharing across nodes and generalization to unseen graph structures, crucial for scaling to large graphs and diverse instances.

Beyond traditional applications, Graph RL has also been used to model strategic behavior in network formation. For instance, Trivedi and Zha (2020) introduce MINE (Multi-agent Inverse models of Network Emergence), a framework that casts network emergence as a Markov game among self-interested agents. By jointly learning reward functions and policies from observed graphs, MINE offers interpretable models of network dynamics and provides a compelling demonstration of multi-agent inverse reinforcement learning applied to evolving graph environments. This framework also

serves as inspiration for the model developed in this thesis, where learned reward functions are used to classify Ethereum transaction subgraphs as arbitrage or non-arbitrage events.

Despite its promise, Graph RL faces ongoing challenges such as reward design, sample inefficiency, and scalability to large or dynamic graphs. Addressing these challenges remains an active area of research, with potential to unlock new capabilities in fields ranging from computational biology to decentralized systems (Darvariu et al., 2024).

Chapter 3

Data Collection

The data collection process for this study combined on-chain and off-chain data sources to create a comprehensive dataset suitable for both supervised learning and reinforcement learning tasks. The primary focus was on arbitrage transactions, leveraging their diverse structure and availability of ground-truth labels. Data was sourced from Flashbots' MEV dataset, widely recognized in the Ethereum ecosystem, and enriched with on-chain information such as transaction receipts and account balances. This section outlines the sources and methods used to collect and process the data, which formed the foundation for constructing features, generating graph-based representations, and designing state spaces for subsequent RF, GNN, and AIRL models.

3.1 Flashbots' MEV Data

The dataset used in this study is sourced from the publicly available Flashbots MEV data repository[1]. Flashbots has published MEV data from September 2022 (post-Merge) to June 2023, which consists of .csv files containing arbitrage, liquidation, sandwiched swaps, and sandwich transactions. This data was collected by Flashbots using the now-deprecated *MEV-inspect* tool mentioned in Section 2.1.2 and serves as the ground truth for training models. This dataset is well-suited for our purposes, as it is widely recognized in the ecosystem. In particular, the MEV analytics platform EigenPhi uses Flashbots labels as a benchmark to evaluate their algorithms (Eigenphi, 2022). In turn, Etherscan, the most prominent Ethereum block explorer, integrates EigenPhi data to tag MEV transactions on its platform. This endorsement chain highlights the credibility and industry acceptance of the Flashbots dataset as a reliable source of ground truth for MEV-related research.

In this study, we focus on atomic arbitrage transactions, arbitrage opportunities fully executed within a single Ethereum transaction, such that all component trades either succeed collectively or the entire transaction is reverted (Daian et al., 2020). While a sandwich attack spans multiple coordi-

[1] https://flashbots-data.s3.us-east-2.amazonaws.com/index.html

Column Name	Description
id	Unique identifier for the arbitrage transaction.
created_at	Timestamp indicating when the data was recorded.
account_address	Address of the entity executing the arbitrage.
profit_token_address	Address of the token in which the profit was received.
block_number	The Ethereum block in which the transaction occurred.
transaction_hash	Unique hash identifying the transaction.
start_amount	Initial amount used in the arbitrage trade.
end_amount	Final amount after executing the arbitrage.
profit_amount	Profit earned from the arbitrage trade.
error	Indicator of any transaction failure (e.g., Reverted).
protocols	Set of protocols involved in the arbitrage execution.

Table 3.1 – Columns available in the Flashbots' arbitrage transaction dataset.

nated transactions, its pattern is highly stereotyped and easier to catch with a rule set. Conversely, an arbitrage opportunity is usually executed inside a single transaction, yet the internal token-flow patterns are far more diverse, so existing heuristics show low recall and require constant ABI maintenance. Arbitrage constitutes a substantial share of total MEV activity, with some studies estimating it, alongside sandwich attacks, to account for up to 99% of all MEV transactions. Moreover, arbitrage detection is where existing heuristic-based approaches suffer the most from low recall and maintenance issues due to their reliance on contract ABIs and hard-coded patterns (Park et al., 2024). Compared to liquidations, which are rarer and relatively easy to detect, arbitrage provides a more impactful and technically demanding target for MEV analysis. This makes it a tractable yet meaningful choice for initial analysis and model development. The dataset sourced from Flashbots provides various attributes for each arbitrage transaction, which are described in Table 3.1.

Although this dataset served as the foundation for identifying arbitrage transactions, the features in this thesis were extracted from transaction receipts and account balances rather than the original Flashbots data.

3.2 Ethereum Data

Ethereum data was collected using web3.py[2] and Alchemy[3], based on the arbitrage transactions identified by Flashbots in Q2 of 2023. For each transaction, asynchronous requests were used to efficiently fetch transaction receipts and account balances, allowing for scalable retrieval of high-volume data.

To complement the arbitrage-labeled transactions, a set of non-arbitrage transactions was collected for each block containing an arbitrage event. Specifically, 5% of the block size was randomly sam-

[2] https://web3py.readthedocs.io/en/stable/
[3] https://www.alchemy.com/

Field	Description
`blockHash`	Hash of the block containing the transaction
`blockNumber`	Block number containing the transaction
`contractAddress`	Created contract address (if contract creation)
`cumulativeGasUsed`	Total gas used in the block up to this transaction
`from`	Sender address
`gasUsed`	Gas used by this transaction alone
`logs`	List of emitted event logs, including topics and decoded data. Each log corresponds to an event, e.g., ERC20 `Transfer`, and contains the emitting contract address, topics, and data.
`logsBloom`	Bloom filter for the logs
`status`	1 if successful, 0 if reverted
`to`	Recipient address (or null for contract creation)
`transactionHash`	Hash of the transaction
`transactionIndex`	Index of the transaction within the block

Table 3.2 – Structure of Transaction Receipt returned by `web3.eth.get_transaction_receipt()`.

pled per arbitrage transaction. This strategy ensured that the dataset remained sufficiently balanced for downstream machine learning tasks.

3.2.1 Transaction Receipts

Transaction receipts were collected using the `get_transaction_receipt(transaction_hash)` method, which returns the corresponding receipt. This method returns an `AttributeDict` containing metadata such as the transaction's block number and hash, gas usage, sender and receiver addresses, emitted logs, and execution status. The full return structure can be found in Table 3.2. In total, transaction receipts were collected for 222,374 transactions spanning blocks 16,950,601 to 17,517,773 (a total of 567,172 blocks). Of these, 86,510 blocks were sampled specifically because they contained arbitrage transactions. These receipts are essential for extracting execution-level features that reflect the structural behavior of the transaction, such as the number of log events and ERC20 transaction amounts.

3.2.2 ETH Balances

For each sender address corresponding to a collected transaction receipt, the ETH balance was queried at the specific block number of the transaction using web3.py's `get_balance(account, block_identifier)` method. In total, balances were retrieved for 78,738 unique addresses. These balances serve as additional node features in the transaction graphs used for the AIRL models.

Chapter 4

Methodology

This chapter presents the experimental design and evaluation framework used to assess the effectiveness of various models for MEV detection. It begins with Random Forest baselines trained on tabular and graph-derived features, proceeds to GNN-based classifiers for graph-structured data, and culminates in a novel application of AIRL for both imitation learning and reward-based classification. The experiments are designed to compare model performance across architectures and demonstrate how learned reward functions can generalize strategic behavior in Ethereum transactions. Implementation details and code access are provided in Appendix A.1.

4.1 Baseline Models: Random Forest on Tabular Features

To establish a robust performance baseline for detecting MEV-related transactions and identifying MEV bots, this study first utilizes traditional machine learning techniques. Specifically, Random Forest classifiers are trained on structured, tabular features extracted from Ethereum transaction data. These baseline models leverage transaction-level, account-level, and graph-based features to evaluate the predictive power of various types of information, providing a foundation against which more sophisticated methods like GNNs and reinforcement learning approaches can later be compared.

4.1.1 Data Preprocessing

To construct a dataset suitable for machine learning models, three distinct feature sets were created, as detailed in the following.

Transaction-Level Features

This dataset contains features derived directly from transaction receipts. The following key features were extracted:

- **Block Information:** Block number.
- **Gas Usage:** Gas used and cumulative gas used.
- **Transaction Index:** Position of the transaction within the block.
- **Effective Gas Price:** The gas price paid.
- **Transaction Status:** Indicates whether the transaction was successful or reverted.
- **Transaction Fee:** Computed as gas used multiplied by effective gas price.
- **Number of logs:** How many log entries the transaction generated.
- **Address Information:** Sender and recipient addresses.
- **Transaction Signatures:** Dummy variables for the top 10 transaction signatures used in arbitrage transactions and the top 10 signatures used in normal transactions.

Account-Level Features

This dataset aggregates features per account across multiple transactions, allowing for an analysis of account behavior. The extracted features include:

- **Gas Usage:** Mean gas used.
- **Cumulative Gas Used:** Mean cumulative gas used.
- **Effective Gas Price:** Max and mean effective gas price.
- **Fee:** Mean fee.
- **Transaction Index:** Minimum, maximum, and mean transaction index values.
- **Status:** Mean status.
- **Number of transactions:** How many transactions the account emitted.
- **Number of logs:** Minimum, maximum, and mean number of transaction logs.
- **Transaction Signatures:** Mean of occurrences of the top 10 transaction signatures in arbitrage and normal transactions.

Graph-Based Transaction Features

In addition to standard transaction-level features, several graph-based features were extracted by constructing directed graphs both at the per-block level and across all transactions spanning the entire block range. A directed graph $G = (V, E)$ was built using sender and receiver address pairs extracted from transactions, where each edge $(u, v) \in E$ represents a transfer from account u to account v.

The following procedure was used to generate graph features:

- The graph G was constructed using NetworkX's (Hagberg et al., 2008) DiGraph representation, with edges defined by the from and to fields of each transaction.
- Node-level features were computed for both the sender and receiver:
  - in-degree and out-degree of each node,
  - clustering coefficient, calculated on an undirected version of G.
- For each transaction edge (u, v), the number of common neighbors between u and v was also computed to capture shared connectivity patterns.
- All graph-derived features were joined back to the original transaction-level data.

This process enriched each transaction with structural information from the local graph topology, enabling the downstream model to exploit higher-order patterns such as hubs, bridges, and dense clusters in the Ethereum transaction graph. The enriched dataset was then used to train the classification models.

4.1.2 Random Forest Classifier

The purpose of this section is to describe the baseline machine learning models used to classify MEV-related transactions and accounts. Four separate models were trained using Random Forest classifiers on distinct feature sets (as described in detail in Section 4.1.1):

1. **Transaction Features:** Extracted from transaction receipts.
2. **Node Features:** Aggregated account-based features used for classifying whether an account is an MEV bot.
3. **Transaction + Per Block Graph Features:** Includes network-based attributes derived from per-block transaction graphs.
4. **Transaction + full Transactions Graph Features:** Includes network-based attributes derived from one transaction graph constructed over all transactions.

Random Forest classifiers were chosen for their robustness, interpretability, and effectiveness in structured data (Biau and Scornet, 2016).

Training Pipeline

To train the models, the following machine learning pipeline was used:

1. **Model Training**:
   - A `ColumnTransformer` was used to preprocess the dataset, in which numerical features were standardized using `StandardScaler`.
   - A pipeline was constructed with preprocessing and model training.
   - `RandomizedSearchCV` was applied to optimize hyperparameters:
     - Number of trees: {100, 200}
     - Maximum depth: {10, 20, 30}
     - Minimum samples per split: {2, 5}
     - Minimum samples per leaf: {1, 2}
     - Maximum features per split: {"sqrt", None}
2. **Cross-Validation**:
   - Stratified K-Fold Cross Validation (K=3) was used to ensure balanced training splits.
3. **Experiment Tracking**:
   - MLflow[1] was used to track experiment parameters and performance.
   - Metrics such as accuracy, precision, recall, and F1-score were logged.

All training and evaluation steps were implemented using the scikit-learn library (Pedregosa et al., 2011).

4.2 Graph Neural Networks

To capture complex relational patterns within Ethereum transaction data, GNNs were employed. Unlike traditional machine learning methods, GNNs explicitly utilize the structural information inherent in graph-based data representations, enabling effective modeling of relationships between transaction entities. Specifically, GraphSAGE was selected due to its proven capability in similar settings (e.g., in the work by Park et al. (2024)). A conceptual illustration of a transaction graph, derived from

[1] https://mlflow.org

Figure 4.1 – Transaction graph of an arbitrage transaction.

transaction receipts, is shown in Figure 4.1. The following sections detail the graph data preprocessing steps and describe the training and evaluation of the GraphSAGE classifier used to classify arbitrage transactions.

4.2.1 Data Preprocessing

Let $\mathcal{T}$ denote the set of Ethereum transaction receipts under consideration. For every $r \in \mathcal{T}$ we construct a *directed, attributed graph*

$$G_r = \big(V_r,\, E_r,\, X_r,\, Z_r\big),$$

where the components are obtained as follows:

1. **Log filtering and edge formation:** Each receipt r is inspected for ERC20 transfer events (identified by the event signature). For every such event with sender u and receiver v we add the ordered pair (u, v) to E_r; the participating addresses are collected into the node set V_r.

2. **Edge attributes:** Each edge $e = (u, v) \in E_r$ is assigned an attribute vector $z_e \in \mathbb{R}^2$ containing *(i)* an integer-encoded transaction signature and *(ii)* the transferred token value. Collecting all edge attributes yields $Z_r = \{z_e\}_{e \in E_r}$.

3. **Node features:** The node feature matrix $X_r \in \mathbb{R}^{|V_r| \times 1}$ stores

$$X_r[u] \;=\; \deg^{\text{out}}_{G_r}(u) + \deg^{\text{in}}_{G_r}(u),$$

   i.e. the total degree of node u within G_r.

4. **Graph encoding and labeling:** The quadruple $\big(V_r, E_r, X_r, Z_r\big)$ is serialized into a PyTorch Geometric (Fey and Lenssen, 2019) `Data` object, guaranteeing compatibility with subsequent GNN training. Each graph receives a label $y_r \in \{0, 1\}$, where $y_r = 1$ denotes an *arbitrage* transaction and $y_r = 0$ a *non-arbitrage* transaction.

4.2.2 GraphSAGE Classifier

To classify arbitrage transactions using graph-structured data, a GraphSAGE model was trained. The training process involved the following steps:

1. **Data Preparation:**
    - Graph-structured transaction data was prepared, ensuring proper encoding of node and edge attributes, as discussed in Section 4.2.1.
    - The data was split into train and test sets by using PyTorch's (Ansel et al., 2024) `random_split` method with a ratio of 80-20.
    - For efficient training, PyTorch's `DataLoader` was used with a batch size of 512 for the training set and 128 for the test set.
2. **Model Architecture:**
    - The GraphSAGE model implemented consists of four convolutional layers with 256 hidden channels each.
    - Node embeddings obtained from convolutional layers were processed using a global mean pooling operation, summarizing node embeddings into graph-level representations.
    - The graph-level embeddings were passed through two fully connected layers: the first linear layer reduced dimensionality to 128 channels, followed by a dropout layer (probability=0.5), and the final linear layer mapped embeddings to the number of output classes.
3. **Training Procedure:**
    - The model was trained using the Adam optimizer with a learning rate of 0.0001.
    - Cross-entropy loss was used as the optimization criterion.
    - Training was conducted over 20 epochs, with performance monitored for convergence. Convergence was defined as test loss fluctuations within ± 0.001 over the last 5 epochs.
4. **Evaluation and Embedding Extraction:**
    - Model performance was evaluated using a held-out test set (80% train, 20% test split), and graph embeddings were extracted for further downstream tasks.

Several GNN architectures, including models incorporating edge attributes such as GATv2Conv (Brody et al., 2022) and GINEConv (Hu et al., 2020), were also explored. However, initial experiments indicated that these approaches resulted in weaker performance compared to the chosen GraphSAGE architecture. Detailed comparative analyses are provided in Section 6.2.

4.3 Hybrid Model: Random Forest with GNN Embeddings

To leverage the strengths of both traditional features and graph embeddings, a hybrid classification model was developed. This model combined embeddings generated by the GraphSAGE model with transaction and edge-based features, utilizing a Random Forest classifier for final prediction.

The following procedure outlines the hybrid model approach:

1. GNN embeddings of dimension 128 were extracted from the previously trained GraphSAGE model for each transaction.
2. These embeddings were merged with the transaction and features, including gas usage, cumulative gas used, effective gas price, transaction fee, and the number of logs associated with each transaction.
3. The combined dataset was then split into training (80%) and testing (20%) sets.
4. The same model pipeline specified in Section 4.1.2 was employed, utilizing a Random Forest classifier with hyperparameter tuning conducted via randomized search cross-validation.
5. Features related to transaction size and cost were standardized before training to ensure effective model learning.
6. The model's performance was evaluated using accuracy, precision, recall, and F1-score metrics.

4.4 Statistical Significance Test

A statistical significance test was conducted to evaluate whether the performance difference between the hybrid model (Section 4.3) and the baseline model trained on transaction features (Section 4.1) was statistically meaningful. Specifically, a contingency table was created from predictions of both models, and McNemar's test (McNemar, 1947) was applied. This test is suitable for comparing two related classification models on the same dataset. Additionally, both models were evaluated with different random seeds to ensure robustness and consistency in the observed performance differences.

4.5 Adversarial Inverse Reinforcement Learning

Inverse reinforcement learning aims to infer the reward functions that explain observed expert behaviors. In this work, we employ AIRL to recover reward functions and policies underlying arbitrage and non-arbitrage transaction behaviors on the Ethereum blockchain. We adopt AIRL because it jointly recovers a potential-based reward and an optimal policy in a single loop, scales to large graph-structured state spaces, and yields a reward signal that we later reuse for transaction-level classification, outperforming Behavioral Cloning, classic Maximum Entropy Inverse RL, and GAIL in both tractability and interpretability (Fu et al., 2018; Ho and Ermon, 2016). To accomplish this, we

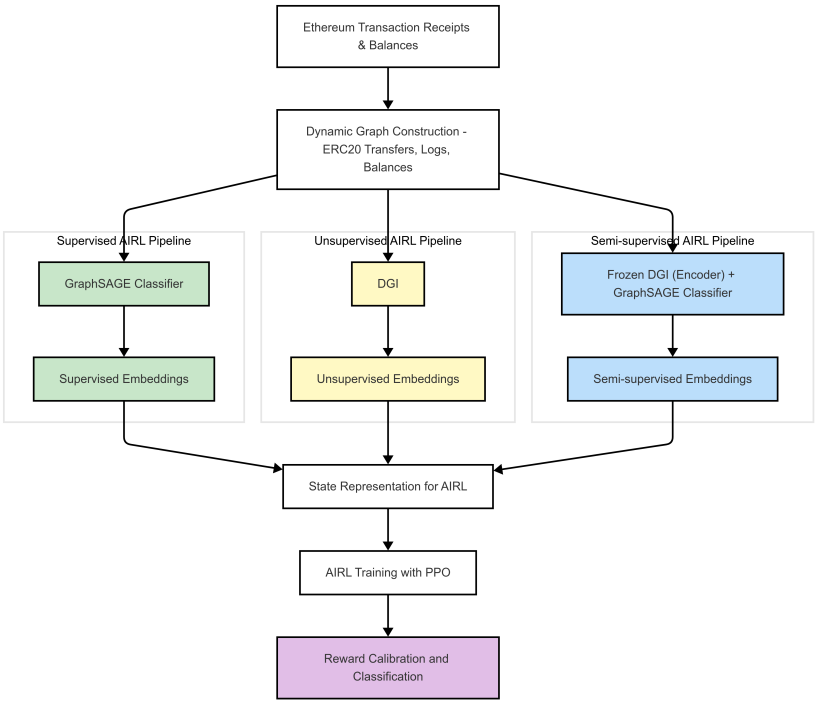


Figure 4.2 – AIRL training pipeline.

design a pipeline that integrates graph-based state representations, a custom environment modeling transaction dynamics, and adversarial learning techniques. An overview of the full AIRL pipeline is presented in Figure 4.2. This section details the key components of our AIRL framework, including data preprocessing, graph representation learning, environment and policy design, reward function recovery, and the use of calibrated rewards for classification.

4.5.1 Dynamic Graph Construction

To capture both the temporal and structural evolution of on-chain behavior, we construct dynamic graphs per address using a sliding window over the most recent transactions. This approach ensures that each agent's action, particularly gas payments, affects the next observed state, preserving the causal structure required by reinforcement learning. Unlike static graphs, which blur recency and dilute short-term effects, our snapshots maintain locality and provide meaningful, temporally grounded inputs for AIRL. We now describe the graph construction process in detail.

Let $\mathcal{A}$ denote the set of sender addresses that appear at least twice in the transaction corpus $\mathcal{T}$. For each $a \in \mathcal{A}$ we order the corresponding receipts chronologically, obtaining a sequence

$$T_a \;=\; \big(\tau_{a,1}, \tau_{a,2}, \ldots, \tau_{a,n_a}\big), \qquad n_a \geq 2,$$

where ordering is induced by `blockNumber` and, secondarily, by `transactionIndex`.

1. **Sliding-window snapshots:** Fix a window size $w = 10$. For each index $t = 1, \ldots, n_a$ define

$$W_{a,t} \;=\; \{\tau_{a,\mathsf{max}(1,t-w+1)}, \ldots, \tau_{a,t}\},$$

so that $|W_{a,t}| = \min(w, t)$. The first snapshot, therefore, contains a single receipt; the window then grows until it reaches length w and subsequently slides forward one receipt at a time, always containing the most recent w transactions.

2. **Log parsing and edge accumulation:** Each receipt $\tau \in W_{a,t}$ is inspected for ERC20 transfer events. For every event with sender u and receiver v the directed edge (u, v) is added to $E_{a,t}$; all involved addresses are placed in $V_{a,t}$. Because the window grows until $t = w$ and then slides, the snapshot sequence is $\langle G_{a,1}, G_{a,2}, \ldots, G_{a,n_a} \rangle$, where early graphs may contain fewer than w receipts and later graphs always aggregate the latest w receipts. Successive snapshots, therefore, share edges: repeated transfers between the same address pair accumulate naturally over time, yielding an evolving dynamic graph for each sender.

3. **Node features:** Let $\mathcal{T}_{a,t}$ be the set of contract addresses (ERC20 tokens) whose balances are non-zero for at least one node in snapshot $G_{a,t}$; this set always contains the placeholder address θ^{ETH} that represents native ETH. Each snapshot is equipped with a feature matrix

$$X_{a,t} \in \mathbb{R}^{|V_{a,t}| \times (1+|\mathcal{T}_{a,t}|)},$$

where the row for node $v \in V_{a,t}$ is

$$X_{a,t}[v] \;=\; \Big(d_v,\; \underbrace{\mathsf{log}\big(1 + |b_v^{\theta_1}|\big)\,\mathsf{sgn}\big(b_v^{\theta_1}\big),\; \ldots,\; \mathsf{log}\big(1 + |b_v^{\theta_{|\mathcal{T}_{a,t}|}}|\big)\,\mathsf{sgn}\big(b_v^{\theta_{|\mathcal{T}_{a,t}|}}\big)}_{\text{one column per } \theta_j \in \mathcal{T}_{a,t}}\Big).$$

Here $d_v = \mathsf{deg}^{\mathsf{out}}_{G_{a,t}}(v) + \mathsf{deg}^{\mathsf{in}}_{G_{a,t}}(v)$ is the total degree of v, and b_v^θ denotes the net balance (aggregated over the current window $W_{a,t}$) of token θ held by v; absent tokens are encoded by $b_v^\theta = 0$. Thus, every feature vector comprises the node's connectivity (degree) followed by signed-log-scaled balances for each token observed in the snapshot, yielding a variable but well-defined feature dimension $1 + |\mathcal{T}_{a,t}|$.

4. **Encoding and data split:** Each quadruple $(V_{a,t}, E_{a,t}, X_{a,t}, y_{a,t})$ is serialized as a PyTorch Geometric `Data` object. The collection of all snapshots is partitioned *by account* into disjoint training and test sets using an 80/20 split, ensuring that all graphs derived from a given address reside exclusively in either split, thus preventing information leakage.

5. **Downstream usage:** The resulting graph sequence constitutes the input corpus for pretraining GNN encoders. During AIRL training, an individual snapshot $G_{a,t}$ provides the state representation s_t supplied to the policy, reward, and discriminator networks.

4.5.2 State Representation using Graph Neural Networks

To represent the state s_t in the AIRL framework, three distinct GNN models were developed to encode the structural and transactional features of Ethereum accounts into low-dimensional embeddings.

GraphSAGE Classifier (Supervised) Embeddings

The first approach involved training a supervised GraphSAGE model using the dynamic state graphs described in Section 4.5.1. The following procedure was used:

1. **Data Preprocessing:**
   - The graph data was preprocessed by padding features to ensure uniform input dimensions and splitting the dataset into training and testing subsets using an 80/20 ratio.
   - Pytorch's `DataLoader` was used with a batch size of 256 for both the training set and the test set.
2. **Model Architecture:**
   - *Encoder:* A four-layer GraphSAGE network with 256 hidden units per layer was implemented, each activated with a ReLU function.
   - *Classification Head:* Node embeddings from the encoder were aggregated into a graph-level representation using global mean pooling, followed by two fully connected linear layers. Dropout regularization (probability $p = 0.5$) was applied between the two linear layers.
3. **Training Objective:** The model was trained for 20 epochs using the Adam optimizer with a learning rate of 0.001, employing cross-entropy loss to classify arbitrage versus non-arbitrage transactions.
4. **Embedding Extraction:** After training, embeddings were generated for each transaction graph and saved.

These supervised embeddings were tailored to capture patterns relevant to the arbitrage classification task and later served as state representations for the AIRL environment.

Deep Graph Infomax (Unsupervised) Embeddings

The second approach uses an unsupervised method to derive more general-purpose graph embeddings:

1. **Data Preprocessing:** The same dynamic state graphs were utilized as in the supervised GraphSAGE approach, including feature padding to ensure consistent input dimensions.

2. **Model Architecture:** The Deep Graph Infomax (DGI) model consists of three primary components:

   - *Encoder:* A four-layer GraphSAGE network with 128 hidden units per layer was used, each activated with a ReLU function.
   - *Corruption Function:* To provide negative examples necessary for contrastive learning, the corruption function shuffles node features within each graph. Formally, given a graph with node feature matrix X, a corrupted version $\tilde{X}$ is generated by randomly permuting the rows of X: $\tilde{X} = X[\text{perm}(1:N), :]$ where $\text{perm}(1:N)$ denotes a random permutation of node indices.
   - *Summary Function:* A global summary vector is generated via global mean pooling:

     $$s = \frac{1}{N} \sum_{i=1}^{N} h_i^{(L)},$$

     where $h_i^{(L)}$ denotes the embedding of node i after the final encoder layer, and N is the total number of nodes.

3. **Training Objective:** The DGI objective maximizes the mutual information between the node embeddings and the global summary vector by distinguishing positive (original graph) from negative (corrupted graph) samples. The model was trained for 20 epochs using the Adam optimizer and a learning rate of 0.001.

4. **Embedding Extraction:** Once trained, the encoder generated embeddings for each transaction graph, which were saved for later downstream tasks.

These unsupervised embeddings were designed to capture the inherent structural features of the graphs, independent of any particular classification task.

Frozen Deep Graph Infomax + GraphSAGE Classifier (Semi-Supervised) Embeddings

The third approach utilized embeddings obtained from the unsupervised DGI encoder, further enhanced by adding a supervised classification head. This setup combined unsupervised structural embeddings with task-specific supervised refinement. The following steps summarize the procedure:

1. **Data Preprocessing:** The identical dynamic state graphs and feature padding procedures from the previous two approaches were retained.

2. **Model Architecture:**
   - *Encoder:* A four-layer GraphSAGE encoder, pretrained via DGI, with 128 hidden units per layer. All encoder parameters remained frozen during training (no gradient updates).
   - *Classification Head:* Node embeddings from the encoder were aggregated into a graph-level representation using global mean pooling, followed by two fully connected linear layers (128-dimensional input to a 128-dimensional hidden space, activated via ReLU, then mapped to two output classes). Dropout regularization (probability $p = 0.5$) was applied between the two linear layers.
3. **Training Objective:** The model was trained for 20 epochs using the Adam optimizer with a learning rate of 0.001. Only the parameters of the supervised classification head were optimized, employing cross-entropy loss to classify arbitrage versus non-arbitrage transactions.
4. **Embedding Extraction:** After training, embeddings were generated for each transaction graph and saved for later downstream tasks.

These embeddings leverage both the general-purpose structural insights from the unsupervised DGI training and the supervised refinements from the downstream classification task.

4.5.3 AIRL Classifier

To recover reward functions and policies underlying observed arbitrage behaviors, we implemented an AIRL pipeline. A key component of this approach is the environment in which the AIRL agent interacts. This environment provides state observations, simulates agent actions, and computes rewards.

Markov Decision Process Formulation

The transaction graph environment was modeled as an MDP, defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where each component was designed specifically for the Ethereum transaction setting:

- **State Space $\mathcal{S}$:** The state s_t is represented by GNN-based transaction graph embeddings, which encode both structural and transactional properties of the Ethereum transaction graphs. Three embedding variants were employed:
  1. Supervised embeddings from a GraphSAGE classifier.
  2. Unsupervised embeddings from a DGI model.
  3. Semi-Supervised embeddings from a frozen DGI + GraphSAGE classifier.

- **Action Space $\mathcal{A}$:** At each timestep, the agent selects an action $a_t \in \{0, 1\}$, corresponding to conservative (low gas price) or aggressive (high gas price) bidding strategies. This simplified binary action space was chosen to focus the learning process on reward inference and classification, rather than attempting to model the full range of decisions a real MEV searcher must make, such as fine-grained fee adjustment, transaction ordering, bundle construction, and multi-contract interactions.
- **Transition Dynamics P:** The environment transitions are deterministic, depending solely on the agent's chosen actions. If the selected action diverges from historical transaction data, the environment dynamically updates the account's ETH balance and reconstructs the transaction graph. The updated graph embedding, computed by the respective GNN, becomes the next state.
- **Reward Function $r(s, a)$:** Rewards are assigned based on alignment with expert behavior. The agent receives a reward of 1 if its chosen action matches the actual gas price decision in historical data; otherwise, the reward is 0. *Note:* this reward is not used for policy updates during AIRL training, as AIRL returns its own shaped reward. It is retained solely for ex-post evaluation to assess whether the learned policy aligns with expert decisions.
- **Discount Factor γ:** Discount factors of $\gamma = 0.99$ and $\gamma = 0.9$ were used for arbitrage and non-arbitrage transactions, respectively.

Episodes correspond to transaction trajectories for individual Ethereum accounts, ordered chronologically. Upon completing a trajectory, the environment resets, and a new account is selected at random.

Implementation Details

Custom Gymnasium Environment. The environment was implemented as a custom Gymnasium (Towers et al., 2024) environment. Let $\mathcal{A}$ denote the set of sender addresses that appear at least twice in the transaction corpus $\mathcal{T}$. For each $a \in \mathcal{A}$ we order the corresponding receipts chronologically, obtaining a sequence

$$T_a = (\tau_{a,1}, \tau_{a,2}, \ldots, \tau_{a,n_a}), \qquad n_a \geq 2,$$

where ordering is induced by `blockNumber` and, secondarily, by `transactionIndex`. The environment is then defined by the following components:

- **Initialization:**
  - *Dataset.* $D_a \subset \mathcal{T}$ for a fixed class label $y \in \{0, 1\}$.

  - *Pre-trained encoder.* A GNN f_θ : Graphs $\to \mathbb{R}^d$ loaded from an MLflow artifact identified by `model_uri`. If the artifact is a `DeepGraphInfomax` wrapper, only its encoder is retained.
  - *Observation space.* Observation space $\mathcal{S} = \mathbb{R}^d$.
  - *Action space.* Action space $\mathcal{U} = \{0, 1\}$.
  - *Sliding-window length.* A constant $w = 10$.

- **Episode reset:** An account $a \sim \text{Unif}(\mathcal{A})$ is selected. Internal indices are reset to $t = 1$ and the state $s_1 = f_\theta(G_{a,1})$ (see below) is returned to the agent.

- **State construction:**

  - *Window.* $W_{a,t} = \{\tau_{a,\max(1,t-w+1)}, \ldots, \tau_{a,t}\}$.
  - *Graph.* ERC20 transfer events in $W_{a,t}$ induce the directed multigraph $G_{a,t} = (V_{a,t}, E_{a,t})$.
  - *Embedding.* The observable state is $s_{a,t} = f_\theta(G_{a,t}) \in \mathbb{R}^d$.

- **Action semantics:** The agent selects $u_t \in \mathcal{U}$ where

$$u_t = 1 \implies \text{high-priority bid } \big(p_t^{\text{med}} + \sigma_t\big), \qquad u_t = 0 \implies \text{low-priority bid } \big(p_t^{\text{med}} - \sigma_t\big).$$

  Here p_t^{med} is the *median effective gas price of all sampled transactions in the block that contains* $\tau_{a,t}$, and σ_t is the corresponding standard deviation.

- **Ground-truth label (evaluation only):** Every receipt is annotated with the historical gas-bid decision $e_t \in \mathcal{U}$. During AIRL training, no external reward is supplied—the algorithm recovers its reward function adversarially. The label e_t is retained exclusively for post-hoc evaluation, where we define $r_t = \mathbf{1}\{u_t = e_t\}$ to measure the match between the learned policy and expert behavior.

- **Balance dynamics:** Let b_t be the ETH balance prior to $\tau_{a,t}$ and g_t the recorded gas usage. The update rule is

$$b_{t+1} = b_t - g_t \begin{cases} p_t^{\text{eff}} & u_t = e_t, \\ p_t^{\text{med}} + \sigma_t & u_t = 1 \neq e_t, \\ p_t^{\text{med}} - \sigma_t & u_t = 0 \neq e_t, \end{cases}$$

  where p_t^{med} and σ_t are the median and s.d. of gas prices in the block and p_t^{eff} is the historical effective gas price.

- **Termination:** An episode terminates when $t = n_a$.

- **Data split:** All graphs derived from the same address reside in the same partition; we use an $80{:}10{:}10$ train-validation-test split by address to avoid data leakage.

This specification mirrors the implementation of `TransactionGraphEnvV2` in our codebase.

Policy, reward, and discriminator networks. AIRL instantiates three parametric functions—policy π_θ, shaped-reward r_ϕ, and discriminator D_ψ—that are optimized in alternating phases.

1. **Policy network π_θ.** We represent the control policy as a mapping $\pi_\theta : \mathcal{S} \longrightarrow \Delta(\mathcal{U})$, where $\mathcal{S} \subset \mathbb{R}^d$ is the GNN state-embedding space and $\mathcal{U} = \{0, 1\}$ the action set. The architecture is the `MlpPolicy` implementation of Stable Baselines3 (Raffin et al., 2021), trained with PPO. This yields a stochastic, entropy-regularized bidding strategy over conservative and aggressive gas prices.

2. **Shaped-reward network r_ϕ.** Following Fu et al. (2018), the learned reward decomposes into a state-action term and a potential function,

$$r_\phi(s, a, s') = f_\phi(s, a) - \gamma\, h_\phi(s') + h_\phi(s),$$

   where both f_ϕ and h_ϕ are multilayer perceptrons. We implement this structure with `BasicShapedRewardNet` from the imitation library (Gleave et al., 2022). The potential h_ϕ removes environment-specific shaping, making r_ϕ transferable to downstream tasks such as transaction-level MEV classification.

3. **Discriminator D_ψ.** Consistent with the AIRL formulation in Fu et al. (2018), the discriminator evaluates, for every state-action-state (s, a, s'),

$$D_\psi(s, a,' s) = \frac{\exp r_\psi(s, a, s')}{\exp r_\psi(s, a, s') + \pi_\theta(a \mid s)}, \qquad \operatorname{logit}\big(D_\psi(s, a)\big) = r_\psi(s, a) - \log \pi_\theta(a \mid s).$$

   High logits correspond to expert behavior, and low logits to generator behavior. The parameters ψ are obtained by maximizing the binary cross-entropy loss

$$\mathcal{L}_{\text{disc}} = \mathbb{E}_{(s,a)\sim\text{expert}}\big[\log D_\psi(s, a)\big] + \mathbb{E}_{(s,a)\sim\pi_\theta}\big[\log\big(1 - D_\psi(s, a)\big)\big].$$

   Training alternates between discriminator updates and PPO policy updates until the generated trajectories are indistinguishable from the expert demonstrations.

Several hyperparameters specifically for PPO and AIRL, including the learning rate, batch size, number of epochs per update, and discount factor γ, were tuned to ensure stable training (see Appendix B.1 for full details).

Reward Calibration and Classification

While the AIRL framework learns reward functions that capture expert behavior, the raw reward outputs generated by the two separate reward networks are not inherently calibrated for direct comparison across classes. To enable the use of learned rewards for downstream classification tasks,

we applied an affine calibration procedure to the reward signals produced by each AIRL reward network.

For both reward networks trained on a specific class (arbitrage or non-arbitrage), we computed the reward outputs on a held-out validation set. The outputs were then transformed using the following affine function:

$$f(x) = \alpha x + \beta$$

The parameters α and β were determined as:

$$\alpha = \frac{\sigma_{\text{target}}}{\sigma_x}, \quad \beta = \mu_{\text{target}} - \alpha \cdot \mu_x$$

where μ_x and σ_x denote the empirical mean and standard deviation of the reward outputs for the validation set, and μ_{target} and σ_{target} were set to 0 and 1 respectively, enforcing zero mean and unit variance. This transformation ensured that the calibrated reward outputs for both classes were directly comparable in scale and location.

After calibration, we evaluated the performance of the learned reward functions by applying both calibrated reward networks to a separate test set consisting of unseen transaction trajectories (arbitrage and non-arbitrage). Each test trajectory was input into both reward networks. For each trajectory, the reward outputs were computed as:

$$r_1 = \texttt{reward\_net1.predict}(s_i, a_i, s_i', d_i), \quad r_0 = \texttt{reward\_net0.predict}(s_i, a_i, s_i', d_i)$$

where r_1 and r_0 are the raw reward outputs from the arbitrage and non-arbitrage networks, respectively, and (s_i, a_i, s_i', d_i) denotes the state, action, next state, and done flag for both classes $i \in \{0, 1\}$. The outputs were then calibrated using the parameters α and β derived from the validation set:

$$\hat{r}_1 = \alpha_1 \cdot r_1 + \beta_1, \quad \hat{r}_0 = \alpha_0 \cdot r_0 + \beta_0$$

After computing the calibrated reward scores for each trajectory, we used them to perform binary classification between arbitrage and non-arbitrage behavior. Let $\mathcal{T}_0 = \{\tau_1^{(0)}, \ldots, \tau_{N_0}^{(0)}\}$ denote the set of test trajectories labeled as non-arbitrage, and $\mathcal{T}_1 = \{\tau_1^{(1)}, \ldots, \tau_{N_1}^{(1)}\}$ the set of arbitrage trajectories. For each trajectory τ, we obtained two scalar scores $\hat{r}_0(\tau)$ and $\hat{r}_1(\tau)$ from the respective calibrated AIRL reward networks. The predicted label was then determined by selecting the class whose reward network assigned the higher calibrated score:

$$\hat{y}(\tau) = \arg\max_{k \in \{0,1\}} \hat{r}_k(\tau).$$

Here, $\hat{y}(\tau) = 1$ indicates that the trajectory is predicted as arbitrage, and $\hat{y}(\tau) = 0$ as non-arbitrage.

This prediction procedure was applied to all trajectories in $\mathcal{T}_0 \cup \mathcal{T}_1$, and the resulting prediction vector was evaluated against the corresponding ground-truth labels $[0, \ldots, 0, 1, \ldots, 1]$ (with N_0 zeros followed by N_1 ones) to compute standard classification metrics including accuracy, precision, recall, and F1-score.

Chapter 5

Results

This chapter presents the empirical results of the models developed in this thesis. We begin by establishing strong baselines using Random Forest classifiers trained on transaction- and account-level features. We then evaluate the impact of incorporating structural graph information through GraphSAGE and hybrid models that combine graph embeddings with tabular features. Finally, we analyze the performance of the AIRL pipeline, assessing its training dynamics, the quality of the learned reward functions, and their utility for downstream classification tasks. The results collectively demonstrate the effectiveness of combining graph-based state representations with reinforcement learning techniques for detecting MEV-related arbitrage behavior on Ethereum.

5.1 Baseline Models: Random Forest

Before exploring advanced graph-based methods and inverse reinforcement learning, baseline models were trained to assess the predictive power of transaction-level and account-level features. These models provide a reference point to evaluate the impact of incorporating graph-based features. The key goal of these baselines is to determine whether integrating graph information enhances classification performance.

5.1.1 Account Classification Performance

The account classification model aimed to predict whether an account was an arbitrage bot based on aggregated account-level features. The classification results for this model are summarized in Table 5.1.

The account classification model achieved a test F1-score of 90.3% and a high accuracy of 99.6%, demonstrating strong performance in distinguishing MEV participants from regular accounts.

Metric	Value
Test Accuracy	0.996
Test Precision	0.965
Test Recall	0.848
Test F1-score	0.903

Table 5.1 – Performance of Account Classification Model

Metric	Model 1	Model 2	Model 3
Test Accuracy	0.984	0.985	0.991
Test Precision	0.978	0.978	0.986
Test Recall	0.996	0.997	0.999
Test F1-score	0.987	0.987	0.993

Table 5.2 – Performance comparison of different transaction classification models.

5.1.2 Transaction Classification Performance

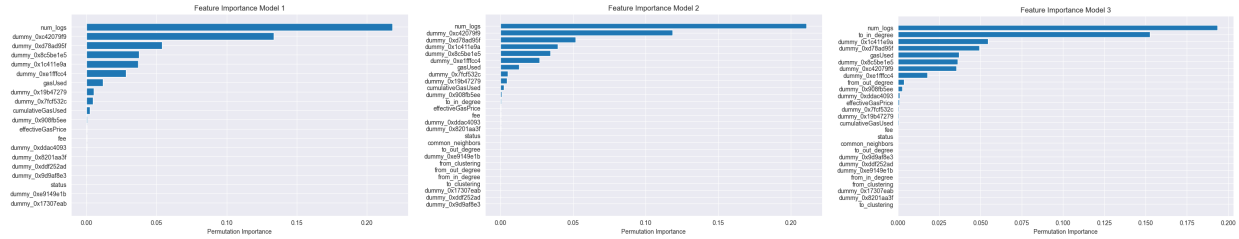
In contrast to account classification, transaction classification focuses on predicting whether a given transaction was an arbitrage transaction. Different variations of transaction-based models were evaluated, including:

- **Model 1** - Transaction features without graph-based attributes: This model relied solely on transaction features created from the transaction receipts.
- **Model 2** - Transaction features with per-block graphs: Graphs were constructed at the block level, and edge-specific features were extracted.
- **Model 3** - Transaction features with a full-graph structure: A single large graph was built from all transactions, and graph-based features were computed.

The classification results for transaction prediction across these different graph construction approaches are summarized in Table 5.2.

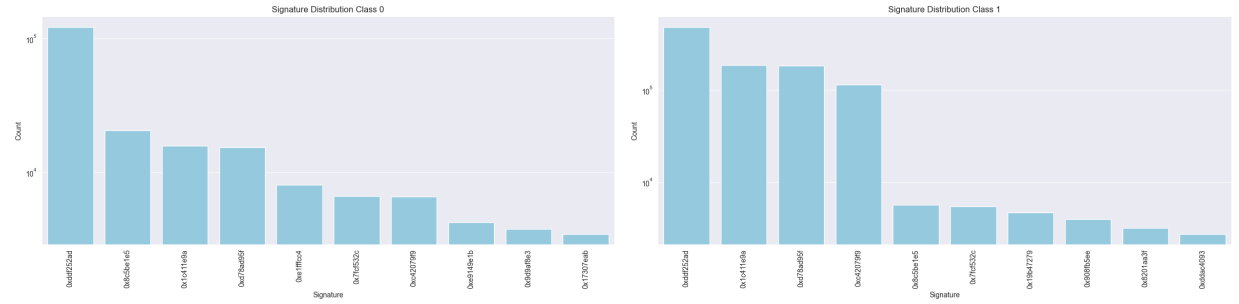
The best performance was obtained using graph-based features from the full transaction network, achieving a test F1-score of 99.3% and accuracy of 99.1%. The per-block graph-based model followed closely with a test F1-score of 98.7%. Lastly, the model without graph features performed similarly, with a test F1-score of 98.7%.

To uncover which raw transaction attributes and on-chain events drive model decisions, we first computed permutation feature importances for each RF variant and then inspected the underlying function-signature distributions. As shown in Figure 5.1, the number of emitted logs (`num_logs`) is unanimously the top predictor, followed closely by the swap-related signature dummies (`dummy_0xd78ad95f9` for Uniswap V2 and `dummy_0xc42079f9` for Uniswap V3). This aligns with the signature frequencies in Figure 5.2: although the ERC20 `Transfer` event (`0xddf252ad…`) is the most common call over-



(a) Model 1 (b) Model 2 (c) Model 3

Figure 5.1 – Permutation feature importances for the three transaction classification models.



(a) Class 0 (b) Class 1

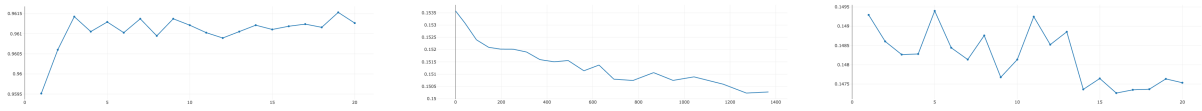
Figure 5.2 – Top 10 most frequent transaction function signatures in each class.

all—especially in non-arbitrage transactions—swap calls appear far more often in arbitrage transactions. A third signature, `0x1c411e9a…` (the Uniswap V2 `Sync` event), also ranks highly in importance and occurs disproportionately in class 1.

5.2 GraphSAGE Classifier

The GraphSAGE model achieved strong classification performance, demonstrating its ability to effectively leverage structural graph features. Specifically, the model achieved a test accuracy of 96.13%, with precision and recall values of 96.21% and 96.13%, respectively. Additionally, the model obtained an F1-score of 96.1%, underscoring its balanced ability to accurately identify arbitrage transactions from graph-structured transaction data. The close alignment of training and test losses (≈ 0.150) suggests that the model generalizes well without significant overfitting. Figure 5.3 illustrates the learning dynamics of the GraphSAGE classifier over 20 training epochs. The steady decline in training loss and the stable trend in test loss indicate consistent convergence. Moreover, the upward trajectory of test accuracy supports the quantitative metrics reported above.

Other GNN architectures were also explored, including GATs and Graph Isomorphism Networks with Edge features (GINE). These models, however, demonstrated considerably weaker performance. Specifically, the GAT model achieved an accuracy of 60.4%, while the GINE model performed simi-



(a) Test Accuracy (b) Train Loss (c) Test Loss

Figure 5.3 – Training loss, test loss, and test accuracy curves for the GraphSAGE classifier over 20 epochs.

Metric	Hybrid Model	Baseline Model
Test Accuracy	0.989	0.984
Test Precision	0.983	0.978
Test Recall	0.999	0.996
Test F1-score	0.991	0.987

Table 5.3 – Comparison of hybrid model performance with baseline model using no graph features.

larly below expectations.

5.3 Hybrid Model

The hybrid model, which integrated graph embeddings from the GraphSAGE model with traditional edge and transaction features, was evaluated against the most basic baseline model, which relied solely on transaction-level features without any graph-based attributes. With a F1-score of 99.1%, the hybrid model was able to outperform the baseline model by a small margin. Table 5.3 and Figure 5.4 summarize this comparison.

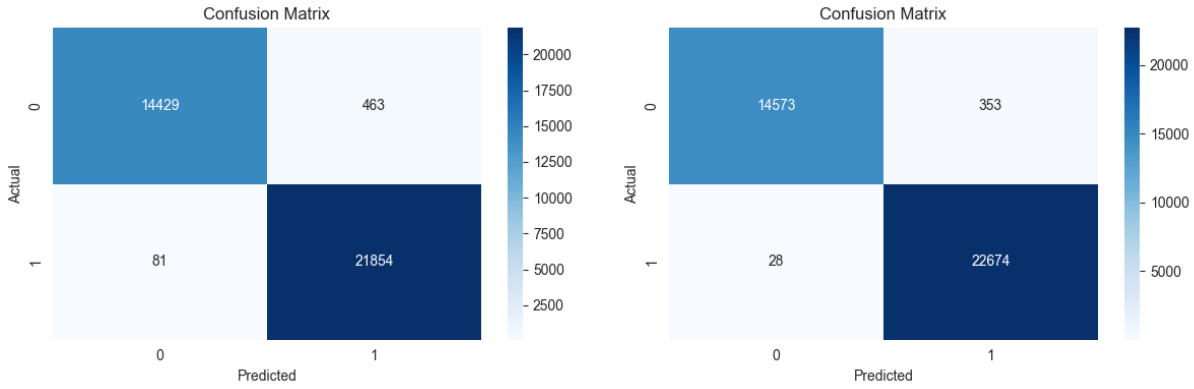
5.4 Statistical Significance Test

To assess whether the performance improvements observed in the hybrid model were statistically significant compared to the baseline model, McNemar's test was conducted. The contingency table derived from the predictions of both models can be seen in Table 5.4.

The McNemar test confirmed a statistically significant difference ($p < 0.01$), supporting the robust-

Baseline	**Hybrid**	**Count**
Correct	Correct	35,406
Correct	Incorrect	53
Incorrect	Correct	220
Incorrect	Incorrect	336

Table 5.4 – Contingency Table from a McNemar Test.



(a) Baseline RF Model (Model 1) (b) Hybrid Model

Figure 5.4 – Confusion matrices comparing the classification performance of the baseline RF model and the hybrid model.

ness of the hybrid model's superior predictive performance. Additionally, repeated experiments using different random seeds showed consistent results, further validating the reliability of the hybrid model's improvements.

5.5 Adversarial Inverse Reinforcement Learning

AIRL was applied to learn reward functions that explain observed Ethereum transaction behaviors. Using expert demonstrations labeled as either arbitrage (class 1) or non-arbitrage (class 0), separate AIRL agents were trained for each class across three different state embedding strategies: supervised (GraphSAGE classifier), unsupervised (DGI), and semi-supervised (frozen DGI encoder with GraphSAGE classifier head). This section evaluates AIRL from three perspectives: (1) the stability and quality of training dynamics, (2) the improvement in agent behavior (policy) through reward-guided learning, and (3) the utility of learned rewards for downstream classification tasks.

5.5.1 Training Dynamics

Table 5.5 summarizes AIRL training outcomes across the three embedding strategies: supervised (GraphSAGE), unsupervised (DGI), and semi-supervised (frozen DGI with GraphSAGE classifier head), trained separately on arbitrage (class 1) and non-arbitrage (class 0) expert trajectories. A table containing all metrics can be found in Appendix B.2.

To control for differences in trajectory length across arbitrage and non-arbitrage classes, we report both the raw episode reward and the step-normalized reward. This decouples performance from episode duration and highlights the relative learning efficiency of each policy. Arbitrage policies consistently achieve higher normalized returns (≈ 0.80–0.82), reflecting more structured, multi-step

Metric	Class 0 (Non-Arb)			Class 1 (Arb)		
	Sup.	Unsup.	Semi	Sup.	Unsup.	Semi
Episode reward	5.17	5.11	5.05	23.43	23.85	23.78
Episode length	8.47	8.47	8.47	29.17	29.17	29.17
Normalized reward (per step)	0.61	0.60	0.60	0.80	0.82	0.82
Discriminator accuracy	0.64	0.66	0.64	0.66	0.68	0.69
Discriminator loss	0.162	0.154	0.155	0.124	0.115	0.111
Policy loss†	0.004	0.004	0.003	2.31	1.86	2.23
Value loss	0.89	1.43	0.74	59.45	91.36	82.04
Approx. KL	0.0066	0.0032	0.0047	0.00073	0.00012	0.00015
Explained variance	0.48	0.41	0.32	0.14	0.14	0.21

Table 5.5 – AIRL training diagnostics. †Absolute value of the policy-gradient term is reported.

behavior. Non-arbitrage returns are notably lower (≈ 0.60–0.61), consistent with the noisier and more heterogeneous nature of the underlying trajectories.

For class 1 (arbitrage), all three embeddings support stable training. The unsupervised model achieved the highest normalized reward (0.82), slightly outperforming both supervised and semi-supervised runs. Interestingly, the unsupervised configuration yielded the lowest policy loss (1.86), while the semi-supervised variant achieved the best explained variance (0.21), albeit still low in absolute terms. Discriminator performance was moderate across all runs (accuracy ≈ 0.66–0.69), indicating a healthy balance between expressiveness and generalization capacity. This level of accuracy avoids overfitting to expert data while still providing informative gradients for reward learning.

Class 0 (non-arbitrage) results remain more fragile. While the normalized rewards are roughly equal across all embeddings (≈ 0.60–0.61), training dynamics reveal more variance. The semi-supervised model achieved the lowest policy and value losses (0.003 and 0.74, respectively), with the supervised run close behind. Explained variance, however, was relatively low across all models (0.32–0.48)—but notably higher than for class 1—indicating unreliable value prediction in this heterogeneous regime. Discriminator accuracy hovered near 0.64–0.66, a level that likely avoids the pathologies of overconfidence but also reflects the difficulty of separating expert and generated behavior in this more variable class.

5.5.2 Policy Improvement

To assess whether policy behavior improves under the influence of learned rewards, we evaluated learner policies *before* and *after* AIRL training on a held-out test set of trajectories, measuring the proportion of actions that matched those taken by the expert concerning gas fee choice. Specifically, the evaluation reward function assigns a score of 1 if the agent selects the same gas strategy (i.e., aggressive vs. conservative) as the expert, and 0 otherwise. Note that this reward is not the same as the internal reward function learned by AIRL, nor is it the ground-truth reward function; rather,

Embedding	Class 0 (Non-Arb)		Class 1 (Arb)	
	Before	After	Before	After
Supervised	0.451 ± 0.379	0.535 ± 0.378	0.493 ± 0.416	0.843 ± 0.263
Unsupervised	0.582 ± 0.360	0.572 ± 0.380	0.653 ± 0.378	0.833 ± 0.276
Semi-Supervised	0.434 ± 0.396	0.576 ± 0.376	0.644 ± 0.388	0.835 ± 0.268

Table 5.6 – Mean reward per episode before and after AIRL training for learner policies.

Metric	Supervised	Unsupervised	Semi-Supervised
Test Accuracy	0.991	0.852	0.964
Test Precision	0.996	0.930	0.966
Test Recall	0.993	0.879	0.989
Test F1-score	0.994	0.904	0.977

Table 5.7 – Classification performance using AIRL-learned reward functions.

it serves as a proxy to evaluate whether the agent has learned to mimic expert gas-fee decisions. In this sense, the evaluation focuses on action-level imitation—specifically, whether the agent selects the same gas strategy as the expert—rather than assessing alignment with the learned reward landscape. The results can be found in Table 5.6.

All embedding strategies exhibit meaningful post-training gains in at least one class, suggesting that policy behavior becomes better aligned with expert decisions under the guidance of learned AIRL rewards. In the arbitrage setting (class 1), improvements are substantial and consistent across embeddings: all models achieve a post-training average reward above 0.83, indicating strong alignment with expert gas fee choices.

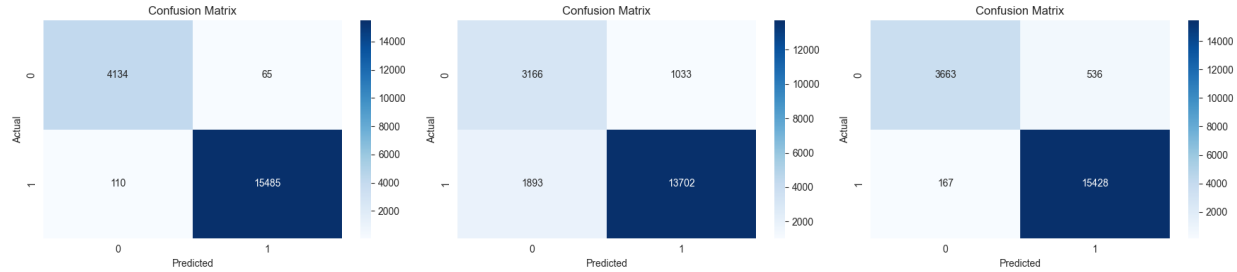
For non-arbitrage (class 0), results are more mixed. The semi-supervised and supervised models improve noticeably, with the former showing the largest gain (from 0.434 to 0.576). The unsupervised model exhibits a small decrease. Despite the relative modesty of some gains, it is notable that all embeddings converge toward higher policy-expert agreement in at least one regime.

5.5.3 Reward-Based Classification Performance

To evaluate the discriminative capacity of the reward functions learned via AIRL, each trained reward network was used to score previously unseen transaction trajectories, as described in Section 4.5.3. For each test sample, the trajectory was evaluated under both the class 0 and class 1 AIRL reward networks. The class corresponding to the higher reward was chosen as the predicted label.

Table 5.7 summarizes the classification results. The supervised AIRL setup achieved the strongest performance, with an accuracy of 99.1%, a precision of 99.6%, and an F1-score of 99.4%. The semi-supervised configuration also performed well, achieving an F1-score of 97.7%, while the unsupervised model lagged behind with a notably lower F1-score of 90.4%.

Figure 5.5 displays the confusion matrices for all three variants. The supervised model (left) ex-



(a) Supervised (b) Unsupervised (b) Semi-supervised

Figure 5.5 – Confusion matrix for reward-based classification using AIRL-trained models.

hibits excellent class separation with minimal misclassification. The semi-supervised model (right) achieves comparably strong performance with slightly higher false positives. The unsupervised model (center) underperforms, misclassifying 1,033 out of 4,199 negative examples and 1,893 out of 15,595 positive examples.

Chapter 6

Discussion

This chapter critically examines the empirical results presented in Chapter 5. It aims to contextualize the performance of the proposed models within the broader landscape of MEV detection, graph representation learning, and inverse reinforcement learning. We first evaluate the baseline Random Forest classifiers to establish a strong point of comparison for graph-based methods. Next, we analyze the effectiveness of GraphSAGE and hybrid classifiers in capturing structural transaction patterns. Finally, we provide a detailed assessment of the AIRL pipeline, examining its training dynamics, representation quality, and capacity to yield interpretable and discriminative reward functions. Finally, we highlight implications, limitations, and directions for future work.

6.1 Baseline Models: Random Forest

The RF classifiers trained on transaction- and account-level features yielded strong results, particularly considering that graph representation learning was not employed at this stage. Both the account classification and transaction classification models performed at a level that rivals or even surpasses models found in prior work that relied on GNNs, e.g. Park et al. (2024) and Yao et al. (2025). This demonstrates the significant predictive power embedded in basic transaction receipts and account summaries.

In account classification, the RF classifier achieved an accuracy of 99.6% and an F1-score of 90.3%, indicating that it could robustly differentiate arbitrage bots from non-arbitrage accounts based on aggregated behavioral features alone. This is noteworthy since only simple statistical aggregates and dummy-encoded transaction signature frequencies were used as inputs.

Transaction classification showed equally compelling results. All three variants of the RF transaction classifier, including the model trained exclusively on raw transaction-level data (Model 1), achieved F1-scores of 98.4% or more. The full-graph variant (Model 3) achieved the best overall performance, with a test accuracy of 99.1% and an F1-score of 99.3%. However, the margin of

improvement introduced by adding graph-based features was relatively modest. This suggests that much of the discriminative signal required to detect arbitrage transactions is already embedded in the raw transaction receipts.

To further investigate model behavior, we computed permutation feature importance for all three transaction classification models (see Figure 5.1). This analysis provides insight into which input features were most influential in the classifier's predictions. Across all models, the number of emitted logs (`num_logs`) consistently emerged as the most important feature, followed by a small set of highly discriminative transaction signature dummies (e.g., `dummy_0xc42079f9`, `dummy_0xd78ad95f`). This finding is particularly relevant for arbitrage detection, where bots often exhibit unique calling patterns or emit specific event signatures tied to arbitrage smart contracts. For the model trained on the full transaction graph (Model 3), `to_in_degree` emerged as the second most important feature. While this aligns with expectations—frequently targeted addresses may signal arbitrage or MEV activity—it reflects a fundamental issue: Model 3 is affected by data leakage. Since the graph is constructed using all transactions, including those occurring after the transaction being classified, the model effectively has access to future information. This compromises the validity of its performance, as it can learn patterns that would not be available in a real-time setting. Furthermore, only in Model 3 did additional graph-derived metrics like `from_out_degree` begin to show relevance, though they remained secondary to core transaction-level features. Despite the data leakage, it's still interesting to observe how these graph metrics influence the model's behavior, as they highlight structural properties of the transaction network that could be leveraged in a properly time-aware setting.

As noted in the Chapter 5 (see Figure 5.2), the distribution of function signatures differs markedly between arbitrage and non-arbitrage transactions. In both classes, the ERC20 Transfer event (`0xddf252ad…`) remains the single most frequent signature, but it constitutes a larger share of calls in non-arbitrage flows. By contrast, swap operations (`0xd78ad95f…` for Uniswap V2 and `0xc42079f9…` for Uniswap V3) are significantly more prevalent in the arbitrage class, reflecting searchers' reliance on cross-DEX price differences. Finally, the Sync event (`0x1c411e9a…`), which pools emit to update internal reserves, appears disproportionately often among arbitrage transactions, consistent with bots deliberately triggering state updates before executing profitable swaps.

Overall, the baseline results demonstrate that structured transaction metadata alone offers a strong foundation for MEV detection. While graph-based techniques have the potential to capture structural nuances and generalize better to unseen patterns, their added value must be evaluated relative to this already high-performing baseline. These results set a strong benchmark that the subsequent GNN and AIRL models must aim to improve upon.

6.2 Graph-Based Models: GraphSAGE and Hybrid Classifier

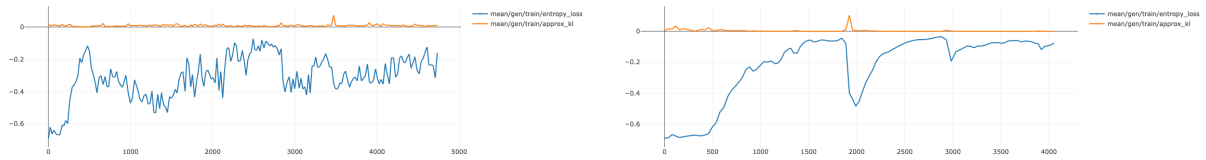
The GraphSAGE classifier demonstrated strong performance, highlighting the value of local transaction graph structure in detecting arbitrage-related activity. However, it did not outperform our robust baseline model. Trained on dynamic graphs constructed per transaction, the model reached a test accuracy of 96.13%, along with a precision of 96.21% and a recall of 96.13%. This yielded an F1-score of 96.1%, indicating a balanced capacity to correctly identify arbitrage transactions. The training and test losses remained closely aligned (both around 0.150), showing that the model generalized well without overfitting. As shown in Figure 5.3, training dynamics were stable across epochs, with overall improvements in accuracy and a general downward trend in loss, despite occasional fluctuations.

While alternative GNN architectures such as GAT and GINE were also evaluated, they performed markedly worse. GAT struggled with convergence and achieved an accuracy of only 60.4%, while GINE failed to outperform GraphSAGE despite its ability to incorporate edge features. This supports prior findings that GraphSAGE's neighborhood aggregation is particularly well-suited for inductive tasks that leverage node feature information (Hamilton et al., 2017).

To assess whether combining structural graph representations with traditional tabular features could offer further gains, a hybrid model was developed. This model concatenated GraphSAGE embeddings with transaction-level features (e.g., gas usage, event logs, function signatures) and used the combined feature vector as input to an RF classifier.

As Table 5.3 shows, the hybrid model outperformed the baseline model across all metrics, achieving a test F1-score of 99.1% and accuracy of 98.9%. Compared to the baseline's F1-score of 98.7%, this improvement may appear modest, but it proved statistically significant. McNemar's test, conducted over a contingency table of model prediction outcomes, yielded a p-value < 0.01. The hybrid model correctly classified 220 additional examples that the baseline missed, confirming the added value of structural features captured via GNN embeddings. Figure 5.4 compares the confusion matrices of the baseline RF model (left) and the hybrid model (right). The hybrid model demonstrates clear improvements in both recall and precision. Specifically, it reduces false positives from 472 to 357 and false negatives from 84 to just 32. This translates to more accurate identification of both arbitrage and non-arbitrage transactions.

In summary, while the baseline already achieved strong performance using only transaction receipts, integrating graph-based representations enabled further improvement. The GraphSAGE model independently offered solid classification performance, but its full potential was unlocked when used as part of a hybrid model. This underscores the complementarity between tabular and structural transaction data in arbitrage detection.



(a) Class 0 (b) Class 1

Figure 6.1 – Policy-entropy loss (blue) together with the approximate Kullback-Leibler divergence between successive policies (orange) for the supervised GraphSAGE encoder for both class 0 and class 1.

6.3 Adversarial Inverse Reinforcement Learning

This section provides a critical analysis of the AIRL results presented in Tables 5.5, 5.6, and 5.7, focusing on convergence dynamics, the role of graph-based state representations, class-specific learning differences, behavioral alignment with external signals, and the classification utility of the learned reward functions.

6.3.1 Training Convergence and Performance

Figure 6.1 plots the policy-entropy loss (blue) together with the approximate Kullback-Leibler (KL) divergence between successive policies (orange) for the supervised GraphSAGE encoder for both class 0 and class 1. In both cases, the entropy loss rises from approximately -0.70 to -0.05 within the first one to two thousand updates, after which it stabilizes in the arbitrage case, but continues to exhibit higher variance in the non-arbitrage setting. In the arbitrage case, entropy quickly plateaus with minimal variance, suggesting convergence to a near-deterministic policy. By contrast, entropy in the non-arbitrage setting continues to fluctuate, indicating uncertainty in the action distribution. This discrepancy is consistent with the structural differences between the two classes. Arbitrage trajectories tend to follow a well-defined pattern that presents consistent and exploitable state-action associations. Non-arbitrage transactions, by contrast, stem from a much more diverse set of behaviors, including user wallet activity, exchange withdrawals or deposits, NFT minting, and so on. This heterogeneity leads to greater state variability and inconsistent returns, thereby delaying policy convergence. Although KL divergence collapses to near-zero in both cases, signifying that PPO's trust region is satisfied and that policy updates are minimal, its higher variance in class 0 reinforces the observation that policy stabilization remains incomplete. Taken together with the absence of a clear plateau in class 0 entropy loss, these signals suggest that non-arbitrage training had not fully converged within the allocated budget and might have benefited from additional training iterations.

Across embeddings, convergence required between 2×10^5 and 4×10^5 environment steps, corresponding to roughly 1,000 PPO updates for arbitrage and 4,000 updates for non-arbitrage. Through-

out this window, the clipped surrogate objective maintained the KL divergence mostly below 10^{-2}, and discriminator accuracy stabilized between 0.64 and 0.69. Despite these positive training signals, the critic's explained variance remained low: 0.14–0.21 for arbitrage and 0.32–0.48 for non-arbitrage. This indicates that the value function struggled to accurately predict returns in both settings. The particularly low scores for arbitrage likely stem from the sparse and binary reward structure combined with short, repetitive trajectories that provide a limited learning signal. In contrast, the higher variance in the non-arbitrage setting, although still suboptimal, suggests that the critic could partially exploit the longer and more varied trajectories, which contain more diverse transitions and potentially richer feedback for value estimation.

6.3.2 State Representation: Architecture and Effects

The environment state is defined as a GNN embedding of the ten most recent transactions initiated by the current account. This sliding-window approach captures temporally local interaction patterns while maintaining tractable state dimensionality. Three encoder configurations were compared:

1. **Supervised:** GraphSAGE trained to classify arbitrage versus non-arbitrage using snapshot labels.
2. **Unsupervised:** DGI trained to maximize mutual information between local and global graph representations.
3. **Semi-supervised:** Frozen DGI encoder with a learned GraphSAGE classification head.

AIRL Training Metrics

For class 1, the supervised GraphSAGE encoder yielded the lowest value loss (59.45), indicating stable learning dynamics and efficient credit assignment under the sparse binary reward. Meanwhile, the semi-supervised encoder produced a marginally higher explained variance (0.21 vs. 0.14), suggesting better alignment between predicted and actual returns despite slightly worse loss metrics. Step-normalized returns were consistent across all embedding strategies, indicating that once the agent has learned to imitate expert trajectories associated with arbitrage, differences in latent structure mainly affect sample efficiency rather than final performance.

In the more heterogeneous class 0 setting, the unsupervised encoder incurred the highest value loss (1.43) but achieved competitive explained variance (0.41) and average reward (0.60)—compared to 0.48 and 0.32 explained variance in the supervised and semi-supervised models—implying that although value estimation was noisier, the richer embeddings may have captured useful structural patterns in the more diverse transaction trajectories. This highlights a potential trade-off between representational capacity and value function stability in environments with high behavioral diversity.

Reward-Net Classification

Supervised embeddings yielded an F1-score of 99.4% when reward scores were used directly for binary classification. Semi-supervised embeddings reached 97.7%, while unsupervised embeddings lagged at 90.4%. The pronounced performance difference underscores the value of task-aligned supervision when reward functions are repurposed as discriminative tools.

6.3.3 Class-Conditioned Learning Behavior

Arbitrage trajectories exhibit lower structural entropy and are dominated by a limited number of recurring patterns, such as simple loop arbitrage, burn-and-mint mechanisms, or set-token decompositions (Park et al., 2024), when compared to non-arbitrage trajectories. These behaviors typically conform to modular subgraph structures, such as short cycles involving high-liquidity tokens or predefined paths through protocol-specific functions, that facilitate consistent exploitation of price differences across decentralized exchanges. This structural regularity makes them easier to capture for both the AIRL discriminator and policy, resulting in more stable loss curves and faster convergence. By contrast, non-arbitrage trajectories contain a broader spectrum of transaction types (by design), including exchange deposits or withdrawals, staking payouts, NFT interactions, and generic user activity. This increased structural and behavioral heterogeneity complicates trajectory modeling, which is reflected in the slower and more unstable convergence of the class 0 agent. Training required four times as many PPO updates (≈ 4000) as in the arbitrage case, and the policy entropy and KL divergence metrics exhibited greater volatility throughout training. Although the final explained variance was higher than in the arbitrage setting, it remained suboptimal in absolute terms, highlighting the persistent challenges of fitting a value function to such a diverse set of behaviors.

6.3.4 Behavioral Alignment with External Evaluation Reward

To assess whether learned policies generalize beyond the training objective, an auxiliary evaluation reward was defined: agents receive $+1$ for matching the gas-price tier (aggressive or conservative) used in the historical transaction, and 0 otherwise. This reward is not observed during training and thus isolates behavioral alignment. For class 1, all three encoders produced substantial improvements to the mean reward per episode: the supervised encoder improved from 0.49 to 0.84, the unsupervised encoder from 0.65 to 0.83, and the semi-supervised encoder from 0.64 to 0.84—an average improvement of 0.24, compared to class 1, where the average improvement was only 0.072. These consistent gains suggest that AIRL internalizes latent preferences that manifest in gas-price bidding, despite this dimension being unobserved by the reward network. This result is especially intuitive in the context of arbitrage, where successful execution often depends on aggressive fee strategies to win bidding wars for the same opportunity. Consequently, gas-price choice becomes more predictive of strategic behavior in arbitrage than in heterogeneous non-arbitrage cases, such

as user transfers. The fact that the supervised encoder achieved the largest post-training improvement further supports the idea that task-aligned supervision helps encode decision-making features, such as fee selection, that are correlated with profitable behavior but not explicitly trained on.

6.3.5 Discriminative Utility of AIRL Reward Networks

A key objective of this thesis was to determine whether AIRL-learned rewards could serve as standalone classifiers of arbitrage activity. When evaluated on a held-out dataset (19,794 arbitrage, 4,466 non-arbitrage), supervised-based reward scores achieved 99.1% accuracy, 99.6% precision, and 99.3% recall. With a F1-score of 99.4%, only 110 false negatives and 65 false positives were observed, affirming the classifier's efficacy under class imbalance. Semi-supervised embeddings achieved 97.7% F1-score, recovering most of the discriminative signal with a frozen encoder. Unsupervised embeddings trailed at 90.4%, confirming the limitations of label-agnostic representations in downstream classification tasks.

Notably, the supervised AIRL classifier outperformed the hybrid model (F1-score = 99.2%) and the baseline model (F1-score = 98.8%), although a direct comparison is not entirely fair: the AIRL evaluation excluded class 0 addresses with fewer than two transactions to ensure adequate trajectory information, whereas the baseline and GNN models used a broader dataset. Still, this result highlights that AIRL can be used not only for imitation learning but also to train effective discriminative models, even under noisy and sparse supervision. Furthermore, these classifiers rely solely on transaction receipts and ETH account balances without access to ABIs, token metadata, or protocol-specific labels, offering a scalable and domain-agnostic approach to arbitrage detection. This makes the method particularly appealing in settings where semantic data is unavailable or obfuscated.

6.4 Implications

The findings presented in this thesis offer several important implications for the broader domains of graph reinforcement learning, inverse reinforcement learning, and MEV analysis.

MEV Detection. This work extends recent efforts in MEV detection that rely on supervised graph neural networks (Park et al., 2024; Yao et al., 2025) by introducing a learning-based framework that integrates graph representation learning with inverse reinforcement learning. Our method does not rely on transaction ABIs or protocol-specific heuristics; instead, it learns to distinguish arbitrage from non-arbitrage behavior via environment interaction and reward inference. By modeling MEV behavior as a dynamic process and leveraging GNN-based state encoders, the approach supports more generalizable detection pipelines that go beyond static feature classification.

Graph Reinforcement Learning. The results demonstrate that graph neural networks, when applied to transaction subgraphs derived from local interaction histories, are highly effective at encoding useful state representations. In both classification and reinforcement learning tasks, GNN-based encoders—particularly GraphSAGE—captured domain-relevant transaction patterns and yielded representations that generalized across tasks. This affirms that GNNs are a natural and effective choice for state construction in graph reinforcement learning settings involving blockchain data. This aligns with the broader literature on graph reinforcement learning, where GNN-based architectures have been shown to effectively capture combinatorial structure and support efficient policy learning over graph-structured domains (Darvariu et al., 2024). To the best of our knowledge, this work is the first to combine AIRL with a GNN-based state encoder trained directly on blockchain transaction graphs. Prior work applying inverse reinforcement learning to blockchain domains has relied exclusively on tabular or sequence-based inputs; graph-based transaction representations have not previously been used in this context.

Adversarial Inverse Reinforcement Learning. A central contribution of this work is the demonstration that AIRL-learned reward functions can be directly repurposed as high-performance classifiers. When paired with task-aligned state encoders, these rewards rival and even exceed traditional supervised models in discriminative performance, while also preserving interpretability and policy-learning capabilities. To the best of our knowledge, this is the first demonstration in the literature where two separate AIRL models—each trained on distinct behavior classes—are calibrated and directly compared to classify unseen trajectories based on reward magnitude. This highlights AIRL's versatility not only as a tool for imitation learning but also as a modular framework for learning reward-aligned utility functions that support multiple downstream applications.

6.5 Limitations

While the results presented in this thesis demonstrate strong performance and novel methodological contributions, several limitations remain.

Dataset quality and coverage. All arbitrage labels were sourced from Flashbots' MEV-Inspect dataset, which, while highly regarded in the industry, inevitably reflects the biases and blind spots of the underlying detection heuristics. As a result, the models are trained only to recognize the subset of arbitrage patterns that MEV-Inspect detected. Furthermore, the dataset exhibits strong class imbalance: only 5% of transactions from arbitrage-containing blocks were sampled as non-arbitrage, leading to skewed training distributions. This imbalance was exacerbated in the AIRL experiments, where non-arbitrage accounts with fewer than two observed transactions were excluded, reducing the effective training set size for (mostly) class 0 and introducing further bias.

Scope limited to arbitrage. The scope of this work is restricted to arbitrage-related MEV behaviors. While arbitrage is one of the most prominent and reliably profitable MEV strategies, many other forms, such as sandwich attacks and liquidations, remain unmodeled. These behaviors often involve different structural and temporal dynamics, which may not be captured by the current architecture or training pipeline.

Challenges of AIRL in MEV settings. The action space in our environment is intentionally simplified to focus on high-level decisions like gas-price tier selection. However, real-world MEV involves highly granular and nearly continuous action spaces, including precise fee adjustments, smart-contract interactions, transaction ordering, and bundling strategies. This makes the direct application of AIRL to modeling general MEV searcher strategies particularly challenging.

6.6 Future Work

This thesis opens several promising avenues for future research:

Full-Block Generalization. While current classification experiments were conducted on a curated set of transactions, an important next step is to evaluate AIRL-based classification performance at the scale of entire Ethereum blocks. This would test the robustness and scalability of the reward-net classifier when exposed to the full distribution of on-chain activity, including rare or noisy transaction types.

Extension to Broader MEV Strategies. The present work focuses exclusively on arbitrage-related MEV, which, despite being among the most ubiquitous and economically significant MEV strategies, remains challenging to detect due to its diverse internal transaction patterns and lack of explicit markers. While sandwich attacks and liquidations are often easier to capture using predefined heuristics and rule-based systems, arbitrage continues to elude such methods and demands more flexible, learning-based approaches (Park et al., 2024). Future research should extend this framework to cover additional MEV behaviors, such as sandwich attacks, liquidations, and suppression, each of which presents distinct structural and temporal characteristics. These extensions will likely require alternative reward shaping, richer context encoding (e.g., incorporating mempool state), and potentially multi-agent learning setups to model the competitive dynamics inherent in such MEV strategies.

AIRL for Block Building. A particularly compelling direction is to apply AIRL in the context of block building. In this setting, the agent's environment would consist of a full candidate block, and the action space would involve selecting and ordering transactions. Transaction embeddings—generated via GNN encoders—could capture fine-grained distinctions between MEV opportunities, normal

user activity, and low-value filler transactions. The learned reward function would ideally internalize which transaction arrangements maximize utility, allowing the agent to imitate relay-published blocks. Training on actual blocks selected by trusted relays would enable end-to-end learning of utility-aligned transaction ordering under realistic constraints, resulting in potentially highly profitable blocks.

Chapter 7

Conclusion

This thesis explored the intersection of graph representation learning and reinforcement learning in the context of MEV detection on Ethereum. By modeling transaction receipts using dynamic graph encoders and training AIRL agents on class-specific trajectories, we demonstrated a novel approach for detecting arbitrage-related MEV.

Our results show that GNN-based state encoders—specifically GraphSAGE—can effectively capture local transaction structure and support both policy learning and downstream classification. The use of AIRL enabled the derivation of reward functions that aligned with expert behavior, despite sparse supervision and low-dimensional action spaces. Most notably, these reward functions were successfully repurposed as classifiers, achieving performance that rivaled and in some cases surpassed traditional supervised baselines.

Beyond technical contributions, this work introduced a new paradigm for MEV detection: one that relies not on hand-crafted features or protocol-specific heuristics, but on learned behavioral incentives. By framing MEV extraction as a process that can be modeled, imitated, and interpreted through inverse reinforcement learning, we provide a foundation for more adaptive and generalizable detection frameworks.

While the approach remains constrained by data biases and task simplifications—such as focusing only on arbitrage and using simplified action spaces—it opens promising avenues for future research. These include extending AIRL to block building, modeling more MEV strategies, and scaling classification to full-block settings.

In summary, this thesis presents a novel methodology and offers a set of insights into how graph reinforcement learning models can be applied to blockchain transaction data. It establishes that inverse reinforcement learning is not only viable but powerful in capturing and generalizing strategic blockchain behavior, offering both interpretability and operational utility for the MEV research community.

Bibliography

Abbeel, Pieter and Andrew Y. Ng (2004). “Apprenticeship learning via inverse reinforcement learning”. In: *Proceedings of the Twenty-First International Conference on Machine Learning*. ICML ’04. Banff, Alberta, Canada: Association for Computing Machinery, p. 1. isbn: 1581138385. doi: 10.1145/1015330.1015430.

Ahmed, Amr, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J. Smola (2013). “Distributed large-scale natural graph factorization”. In: *Proceedings of the 22nd International Conference on World Wide Web*. WWW ’13. Rio de Janeiro, Brazil: Association for Computing Machinery, pp. 37–48. isbn: 9781450320351. doi: 10.1145/2488388.2488393.

Ansel, Jason, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala (Apr. 2024). “PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation”. In: *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS ’24)*. ACM. doi: 10.1145/3620665.3640366.

Arora, Saurabh and Prashant Doshi (2021). “A survey of inverse reinforcement learning: Challenges, methods and progress”. In: *Artificial Intelligence* 297, p. 103500. issn: 0004-3702. doi: https://doi.org/10.1016/j.artint.2021.103500.

Belkin, Mikhail and Partha Niyogi (2001). “Laplacian eigenmaps and spectral techniques for embedding and clustering”. In: *Proceedings of the 15th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS’01. Vancouver, British Columbia, Canada: MIT Press, pp. 585–591.

Biau, Gérard and Erwan Scornet (Apr. 2016). “A random forest guided tour”. In: *TEST* 25.2, pp. 197–227. issn: 1863-8260. doi: 10.1007/s11749-016-0481-7.

Brody, Shaked, Uri Alon, and Eran Yahav (2022). “How Attentive are Graph Attention Networks?” In: *International Conference on Learning Representations*.

Buterin, Vitalik (Jan. 2022). *State of research: increasing censorship resistance of transactions under proposer/builder separation (PBS)*. url: https://notes.ethereum.org/@vbuterin/pbs_censorship_resistance (visited on 04/14/2025).

Cao, Shaosheng, Wei Lu, and Qiongkai Xu (2015). “GraRep: Learning Graph Representations with Global Structural Information”. In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. CIKM ’15. Melbourne, Australia: Association for Computing Machinery, pp. 891–900. isbn: 9781450337946. doi: 10.1145/2806416.2806512.

Daian, Philip, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels (2020). “Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability”. In: *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 910–927. doi: 10.1109/SP40000.2020.00040.

Darvariu, Victor-Alexandru, Stephen Hailes, and Mirco Musolesi (2024). “Graph Reinforcement Learning for Combinatorial Optimization: A Survey and Unifying Perspective”. In: *Transactions on Machine Learning Research*. Survey Certification. issn: 2835-8856.

DefiLlama (2025). *DefiLlama - DeFi Dashboard*. url: https://defillama.com/chain/ethereum (visited on 04/15/2025).

Eigenphi (2022). *MEV Data*. url: https://eigenphi.io (visited on 04/14/2025).

Fey, Matthias and Jan E. Lenssen (2019). “Fast Graph Representation Learning with PyTorch Geometric”. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Finn, Chelsea, Sergey Levine, and Pieter Abbeel (2016). “Guided cost learning: deep inverse optimal control via policy optimization”. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML’16. New York, NY, USA: JMLR.org, pp. 49–58.

Finn, Chelsea, Tianhe Yu, Justin Fu, Pieter Abbeel, and Sergey Levine (2017). “Generalizing Skills with Semi-Supervised Reinforcement Learning”. In: *International Conference on Learning Representations*.

Flashbots (2022). *MEV-Boost Integration Overview*. Diagram adapted from Flashbots documentation. url: https://docs.flashbots.net/flashbots-mev-boost/introduction (visited on 04/14/2025).

— (2025a). *Block Builders*. url: https://docs.flashbots.net/flashbots-mev-boost/block-builders (visited on 04/14/2025).

— (2025b). *Block Proposers*. url: https://docs.flashbots.net/flashbots-mev-boost/block-proposers (visited on 04/14/2025).

— (2025c). *Flashbots Explorer*. url: https://explore.flashbots.net (visited on 04/10/2025).

— (2025d). *Flashbots: Democratizing MEV Extraction*. url: https://flashbots.net (visited on 04/10/2025).

Flashbots (2025e). *Overview*. url: https://docs.flashbots.net/flashbots-auction/overview (visited on 04/14/2025).

— (2025f). *Relay Fundamentals*. url: https://docs.flashbots.net/flashbots-mev-boost/relay (visited on 04/14/2025).

Fu, Justin, Katie Luo, and Sergey Levine (2018). "Learning Robust Rewards with Adverserial Inverse Reinforcement Learning". In: *International Conference on Learning Representations*.

Gilmer, Justin, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl (2017). "Neural message passing for Quantum chemistry". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, pp. 1263–1272.

Gleave, Adam, Mohammad Taufeeque, Juan Rocamonde, Erik Jenner, Steven H. Wang, Sam Toyer, Maximilian Ernestus, Nora Belrose, Scott Emmons, and Stuart Russell (2022). *imitation: Clean Imitation Learning Implementations*. arXiv:2211.11972v1 [cs.LG]. arXiv: 2211.11972 [cs.LG].

Gosselin, Stephane (Nov. 2020). *Flashbots: Frontrunning the MEV crisis*. url: https://ethresear.ch/t/flashbots-frontrunning-the-mev-crisis/8251 (visited on 04/10/2025).

Gramlich, Vincent, Dennis Jelito, and Johannes Sedlmeir (Oct. 2024). "Maximal extractable value: Current understanding, categorization, and open research questions". In: *Electronic Markets* 34.1. doi: 10.1007/s12525-024-00727-x.

Grover, Aditya and Jure Leskovec (2016). "node2vec: Scalable Feature Learning for Networks". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, pp. 855–864. isbn: 9781450342322. doi: 10.1145/2939672.2939754.

Hagberg, Aric A., Daniel A. Schult, and Pieter J. Swart (2008). "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, pp. 11–15.

Hamilton, William L. (2022). *Graph Representation Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Springer Nature Switzerland. isbn: 9783031015885.

Hamilton, William L., Rex Ying, and Jure Leskovec (2017). "Inductive representation learning on large graphs". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., pp. 1025–1035. isbn: 9781510860964.

Ho, Jonathan and Stefano Ermon (2016). "Generative Adversarial Imitation Learning". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates, Inc.

Hu, Weihua, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec (2020). "Strategies for Pre-training Graph Neural Networks". In: *International Conference on Learning Representations*.

Jaynes, E. T. (May 1957). “Information Theory and Statistical Mechanics”. In: *Phys. Rev.* 106 (4), pp. 620–630. doi: 10.1103/PhysRev.106.620.

Kipf, Thomas N. and Max Welling (2017). “Semi-Supervised Classification with Graph Convolutional Networks”. In: *International Conference on Learning Representations*.

McNemar, Quinn (1947). “Note on the Sampling Error of the Difference Between Correlated Proportions or Percentages”. In: *Psychometrika* 12.2, pp. 153–157. doi: 10.1007/BF02295996.

Merkwirth, Christian and Thomas Lengauer (Sept. 2005). “Automatic Generation of Complementary Descriptors with Molecular Graph Networks”. In: *Journal of chemical information and modeling* 45, pp. 1159–68. doi: 10.1021/ci049613b.

Ng, Andrew Y. and Stuart J. Russell (2000). “Algorithms for Inverse Reinforcement Learning”. In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML ’00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 663–670. isbn: 1558607072.

Ou, Mingdong, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu (2016). “Asymmetric Transitivity Preserving Graph Embedding”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, pp. 1105–1114. isbn: 9781450342322. doi: 10.1145/2939672.2939751.

Park, Seongwan, Woojin Jeong, Yunyoung Lee, Bumho Son, Huisu Jang, and Jaewook Lee (2024). “Unraveling the MEV enigma: ABI-free detection model using Graph Neural Networks”. In: *Future Generation Computer Systems* 153, pp. 70–83. issn: 0167-739X. doi: https://doi.org/10.1016/j.future.2023.11.014.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena (2014). “DeepWalk: online learning of social representations”. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’14. New York, New York, USA: Association for Computing Machinery, pp. 701–710. isbn: 9781450329569. doi: 10.1145/2623330.2623732.

Piet, Julien, Jaiden Fairoze, and Nicholas Weaver (2022). “Extracting Godl [sic] from the Salt Mines: Ethereum Miners Extracting Value”. In: arXiv: 2203.15930 [cs.CR].

Qin, Kaihua, Liyi Zhou, and Arthur Gervais (May 2022). “Quantifying Blockchain Extractable Value: How dark is the forest?” In: *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 198–214. doi: 10.1109/sp46214.2022.9833734.

Raffin, Antonin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann (2021). “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.268, pp. 1–8.

Russell, Stuart J. (1998). “Learning agents for uncertain environments (extended abstract)”. In: *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*. COLT’ 98. Madison, Wisconsin, USA: Association for Computing Machinery, pp. 101–103. isbn: 1581130570. doi: 10.1145/279943.279964.

Scarselli, Franco, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini (2009). “The Graph Neural Network Model”. In: *IEEE Transactions on Neural Networks* 20.1, pp. 61–80. doi: 10.1109/TNN.2008.2005605.

Schulman, John, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel (2016). “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. doi: https://doi.org/10.48550/arXiv.1506.02438.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). *Proximal Policy Optimization Algorithms*. arXiv: 1707.06347 [cs.LG].

Smedley, Brock (Aug. 2022). *Searching Post-Merge*. url: https://writings.flashbots.net/searching-post-merge#proposerbuilder-separation-pbs (visited on 04/10/2025).

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction*. Second. MIT Press.

Tang, Jian, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei (2015). “LINE: Large-scale Information Network Embedding”. In: *Proceedings of the 24th International Conference on World Wide Web*. WWW ’15. Florence, Italy: International World Wide Web Conferences Steering Committee, pp. 1067–1077. isbn: 9781450334693. doi: 10.1145/2736277.2741093.

Towers, Mark, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis (2024). *Gymnasium: A Standard Interface for Reinforcement Learning Environments*. arXiv: 2407.17032 [cs.LG].

Trivedi, Rakshit and Hongyuan Zha (2020). “Learning Strategic Network Emergence Games”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., pp. 6791–6802.

Veličković, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio (2018). “Graph Attention Networks”. In: *International Conference on Learning Representations*.

Weintraub, Ben, Christof Ferreira Torres, Cristina Nita-Rotaru, and Radu State (Oct. 2022). “A flash(bot) in the pan: measuring maximal extractable value in private pools”. In: *Proceedings of the 22nd ACM Internet Measurement Conference*. IMC ’22. ACM, pp. 458–471. doi: 10.1145/3517745.3561448.

Wulfmeier, Markus, Peter Ondruska, and Ingmar Posner (2015). “Maximum entropy deep inverse reinforcement learning”. In: *arXiv preprint arXiv:1507.04888*.

Xu, Keyulu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka (Oct. 2018). “Representation Learning on Graphs with Jumping Knowledge Networks”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 5453–5462.

Yao, Zihao, Fanding Huang, Yannan Li, Wei Duan, Peng Qian, Nan Yang, and Willy Susilo (Jan. 2025). “Mecon: A GNN-based graph classification framework for MEV activity detection”. In: *Expert Systems with Applications* 269, p. 126486. doi: 10.1016/j.eswa.2025.126486.

Ziebart, Brian D, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. (2008). “Maximum entropy inverse reinforcement learning.” In: *Aaai*. Vol. 8. Chicago, IL, USA, pp. 1433–1438.

Appendices

Appendix A

Code and Resources

A.1 Code and Resources

The full source code and instructions for reproducing the experiments presented in this thesis are available at:

https://gitlab.uzh.ch/liam.tessendorf/graph-reinforcement-learning-using-blockchain-data

Appendix B

AIRL and PPO Training Hyperparameters

B.1 AIRL and PPO Hyperparameters

This section details the hyperparameters used during AIRL training for both arbitrage (class 1) and non-arbitrage (class 0) expert trajectories. The configuration distinguishes between parameters used for PPO and those specific to the AIRL discriminator.

B.1.1 PPO Parameters

Arbitrage (Class 1)

- Learning rate: `2e-4`
- Batch size: `256`
- Number of epochs: `10`
- Discount factor (γ): `0.99`
- GAE lambda: `0.95`
- Clipping range: `0.25`
- Entropy coefficient: `0.05`
- Value function coefficient: `1.0`
- Max gradient norm: `0.5`
- Normalize advantage: `False`
- Use state-dependent exploration (SDE): `False`
- SDE sample frequency: `-1`
- Policy kwargs: `{"use_expln": True}`
- Rollout steps per PPO update: `2048`
- Total timesteps: `2048 * 100`

Non-Arbitrage (Class 0)

- Learning rate: `5e-4`
- Batch size: `256`
- Number of epochs: `10`
- Discount factor (γ): `0.9`
- GAE lambda: `0.8`
- Clipping range: `0.25`
- Entropy coefficient: `0.03`
- Value function coefficient: `0.5`
- Max gradient norm: `0.5`
- Normalize advantage: `True`
- Use state-dependent exploration (SDE): `False`
- SDE sample frequency: `-1`
- Policy kwargs: `{"use_expln": True}`
- Rollout steps per PPO update: `1024`
- Total timesteps: `2048 * 200`

B.1.2 AIRL-Specific Parameters

Arbitrage (Class 1)

- Number of discriminator updates per round: `8`
- Discriminator learning rate: `1e-3`
- Discriminator weight decay: `1e-4`
- Expert batch size: `640`
- Expert minibatch size: `128`
- Allow variable horizon: `True`

Non-Arbitrage (Class 0)

- Number of discriminator updates per round: `6`
- Discriminator learning rate: `1e-3`
- Discriminator weight decay: `1e-4`
- Expert batch size: `512`
- Expert minibatch size: `128`
- Allow variable horizon: `True`

B.2 AIRL Training Metrics

Table B.1 reports summary statistics collected during AIRL training runs for all experimental settings. Each column corresponds to one configuration, defined by the encoder type (supervised, unsupervised, or semi-supervised) and the transaction class (arbitrage or non-arbitrage). In the column headers, **Sup**, **Unsup**, and **Semi** refer to the encoder type—supervised, unsupervised (DGI), and semi-supervised, respectively—while **C0** and **C1** indicate the transaction class: **C0** for non-arbitrage and **C1** for arbitrage samples.

Table B.1 – Full AIRL–PPO training metrics (mean values over the last reporting window)

Metric	Sup-C0	Sup-C1	Unsup-C0	Unsup-C1	Semi-C0	Semi-C1
Generator (policy) – time and rollout						
Time elapsed [s]	17.5	34	16	33	15	29
Total timesteps	409 088	204 800	409 088	204 800	409 088	204 800
Iterations	1.5	1	1.5	1	1.5	1
FPS	84	60	93.5	61	96.5	69
Episode length (mean)	8.47	29.17	8.47	29.17	8.47	29.17
Episode reward *wrapped* (mean)	–6.09	–6.07	–5.29	–21.69	–7.07	–23.90
Episode reward (mean)	5.17	23.43	5.10	23.85	5.05	23.78
Generator (policy) – optimisation						
Clip fraction	0.0246	0.0020	0.0232	0.0010	0.0326	0.0015
Entropy loss	–0.160	–0.0758	–0.147	–0.0072	–0.185	–0.0074
Generator loss	0.427	54.93	0.760	68.41	0.309	59.83
Policy-gradient loss	–0.0043	2.306	–0.0041	–1.864	–0.0035	–2.226
Learning rate	5e-4	2e-4	5e-4	2e-4	5e-4	2e-4
Clip range	0.25	0.25	0.25	0.25	0.25	0.25
Approx. KL	0.00664	0.00073	0.00319	0.00012	0.00467	0.00015
Explained variance	0.481	0.138	0.413	0.139	0.317	0.207
# PPO updates	3 995	1 000	3 995	1 000	3 995	1 000
Value loss	0.888	59.45	1.43	91.36	0.735	82.04
Discriminator						
Global step	200	100	200	100	200	100
Accuracy (generated)	0.941	0.813	0.884	0.771	0.857	0.768
Accuracy (expert)	0.339	0.510	0.428	0.598	0.424	0.609
Overall accuracy	0.640	0.662	0.656	0.684	0.641	0.688
Discriminator loss	0.162	0.124	0.154	0.115	0.155	0.111
Discriminator entropy	0.581	0.552	0.549	0.575	0.557	0.564
Proportion expert (true)	0.50	0.50	0.50	0.50	0.50	0.50
Proportion expert (predicted)	0.199	0.348	0.272	0.414	0.284	0.421

Eidesstattliche Erklärung

Der/Die Verfasser/in erklärt an Eides statt, dass er/sie die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als die angegebenen Hilfsmittel angefertigt hat. Die aus fremden Quellen (einschliesslich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Zürich, 22.06.2025
Ort, Datum

Unterschrift des/der Verfassers/in