RESPONSIBLE MACHINE LEARNING: SUPPORTING PRIVACY PRESERVATION AND NORMATIVE ALIGNMENT WITH MULTI-AGENT SIMULATION

A Dissertation
Presented to
The Academic Faculty

By

David Byrd

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Computing
School of Interactive Computing

Georgia Institute of Technology

August 2021

RESPONSIBLE MACHINE LEARNING: SUPPORTING PRIVACY PRESERVATION AND NORMATIVE ALIGNMENT WITH MULTI-AGENT SIMULATION

Thesis committee:

Dr. Tucker Balch, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Mark Riedl
School of Interactive Computing
Georgia Institute of Technology

Dr. Jonathan Clarke
Scheller College of Business
Georgia Institute of Technology

Dr. Thad Starner
School of Interactive Computing
Georgia Institute of Technology

Dr. Maria Hybinette
Department of Computer Science
University of Georgia

Date approved: July 15, 2021

For my wife, Evette, who encouraged a late bloomer to follow his dreams.

ACKNOWLEDGMENTS

I came to grad school with only one piece of practical advice: Find an advisor you enjoy spending time with, and everything else becomes much easier. Thank you, Tucker, for being exactly that advisor, whose confidence, support, and research intuition always helped me find the path forward.

Thank you also to Mark Riedl for guidance on normative alignment, and for patiently (and correctly!) explaining why several of my ideas wouldn't work. To Thad Starner for early lessons interpreting experimental results, and for challenging me to keep the ethical implications of my work top of mind. To Jonathan Clarke for helping me navigate the strange world of financial research, and for being one of the kindest academics I have met. And to Maria Hybinette for sharing incredible knowledge of discrete event simulation, and for thinking my research was "cool" when I was not so sure.

I am grateful to the colleagues and co-authors who worked with me on the articles that form Chapters 2-4 of this dissertation: Tucker (ch.2,4) and Maria (ch.2) again, as well as Mr. Vaikkunth Mugunthan (ch.3), Ms. Sruthi Palaparthi (ch.2), and Dr. Antogoni Polychroniadou (ch.3,4). My use of the first person plural in those chapters incorporates their contributions to the work, but all errors and omissions remain my responsibility as the primary author.

My position at the Interactive Media Technology Center at Georgia Tech was a source of stability, collegiality, and inspiration throughout my Ph.D. My thanks to Beth, Maribeth, Scott, and everyone at IMTC for their advice and companionship over the years. Special thanks to co-advisee Brian Hrolenok for guidance through the Ph.D. process, constant feedback on teaching and research, and for talking me off the ledge at several key moments. Also thanks to Jeff Wilson and Jeremy Johnson for years of help and friendly banter, and of course helping me become research faculty

in the first place. And thanks to Dr. Rosa Arriaga for above-and-beyond support of a fellow research scientist: the office hours, the cheerleading, and the mock interviews each were invaluable and came at exactly the right time.

I would like to gratefully acknowledge the National Science Foundation (through award 1741026) and J.P. Morgan (through an AI Research Fellowship) for their support of this work. The presence of any opinion or conclusion in this work should be attributed to the author(s) and not necessarily to these funding organizations.

Finally, I would like to thank my family. My parents, Betty and Marion, and grandparents Mildred and Gus, who set me on an inquisitive path from an early age, encouraging my interest in everything from math to mythology, and thereby helping me become the person I am today. And my wife Evette, to whom I have dedicated this dissertation. I quite literally could not have done it without you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This dissertation aims to advance responsible machine learning through multi-agent simulation (MAS). I introduce and demonstrate an open source, multi-domain discrete event simulation framework and use it to: (1) improve state-of-the-art privacy-preserving federated learning and (2) construct a novel method for normatively-aligned learning from synthetic negative examples.

Due to their complexity and capacity, the training of modern machine learning (ML) models can require vast user-collected data sets. The current formulation of federated learning arose in 2016 after repeated exposure of sensitive user information from centralized data stores where mobile and wearable training data was aggregated. Privacy-preserving federated learning (PPFL) soon added stochastic and cryptographic layers to protect against additional vectors of data exposure. Recent state of the art protocols have combined differential privacy (DP) and secure multi-party computation (MPC) to keep client training data set parameters private from an "honest but curious" server which is legitimately involved in the learning process, but attempting to infer information it should not have.

Investigation of PPFL can be cost prohibitive if each iteration of a proposed experimental protocol is distributed to virtual computational nodes geolocated around the world. It can also be inaccurate when locally simulated without concern for client parallelism, accurate timekeeping, or computation and communication loads. In this work, a recent PPFL protocol is instantiated as a single-threaded MAS to show that its model accuracy, deployed parallel running time, and resistance to inference of client model parameters can be inexpensively evaluated. The protocol is then extended using oblivious distributed differential privacy to a new state of the art secure against attacks of collusion among all except one participant, with an empirical demonstration that the new protocol improves privacy with no loss of accuracy to the final model.

State of the art reinforcement learning (RL) is also increasingly complex and hard to interpret, such that a sequence of individually innocuous actions may produce an unexpectedly harmful result. Safe RL seeks to avoid these results through techniques like reward variance reduction, error state prediction, or constrained exploration of the state-action space. Development of the field has been heavily influenced by robotics and finance, and thus it is primarily concerned with physical failures like a helicopter crash or a robot-human workplace collision, or monetary failures like the depletion of an investment account. The related field of Normative RL is concerned with obeying the behavioral expectations of a broad human population, like respecting personal space or not sneaking up behind people. Because normative behavior often implicates safety, for example the assumption that an autonomous navigation robot will not walk through a human to reach its goal more quickly, there is significant overlap between the two areas.

There are problem domains not easily addressed by current approaches in safe or normative RL, where the undesired behavior is subtle, violates legal or ethical rather than physical or monetary constraints, and may be composed of individually-normative actions. In this work, I consider an intelligent stock trading agent that maximizes profit but may inadvertently learn "spoofing", a form of illegal market manipulation that can be difficult to detect. Using a financial market based on MAS, I safely coerce a variety of spoofing behaviors, learn to distinguish them from other profit-driven strategies, and carefully analyze the empirical results. I then demonstrate how this spoofing recognizer can be used as a normative guide to train an intelligent trading agent that will generate positive returns while avoiding spoofing behaviors, even if their adoption would increase short-term profits. I believe this contribution to normative RL, of deriving an method for normative alignment from synthetic non-normative action sequences, should generalize to many other problem domains.

CHAPTER 1
INTRODUCTION

This dissertation introduces a multi-agent simulation framework for multi-domain empirical investigation of approaches in responsible machine learning with specific concern for end user data privacy and normative alignment. Using these key research areas as exemplars, I aim to persuade the reader of the central thesis of this document:

> ***Multi-agent simulation is an effective tool to solve problems in responsible machine learning.***

In this first chapter, I situate the work and lay out its primary motivations, including the need for responsible machine learning, better end-user data privacy, and normatively aligned algorithms, and why multi-agent simulation is an appropriate solution to these problems. In subsequent chapters, I provide evidence in support of my thesis by answering some specific research questions:

1. How can multi-agent simulation (MAS) be used to safely avoid learning undesirable behaviors?
    (a) Can MAS be used to safely learn a recognizer for a complex, vaguely defined, illegal behavior composed of individually-acceptable actions?
    (b) Given such a recognizer, can MAS help a reinforcement learning (RL) agent successfully perform a task without violating the relevant law?
2. How can MAS be used to improve the privacy of user-derived training data under a federated learning protocol?
    (a) Can MAS reduce the cost to empirically evaluate a proposed protocol's speed and model accuracy?

(b) Can MAS help to empirically quantify the privacy gains of an improved protocol?

These research questions are addressed in Chapters 2-6. The final chapter draws some overarching conclusions and suggests future work in the area.

1.1 Responsibility and Algorithmic Bias

In the last decade, machine learning (ML) algorithms have been applied to complex problems with a level of success that makes them attractive to government and industry practitioners. Post hoc analysis of some such systems has raised questions concerning the responsible use of ML, particularly with regard to unintended algorithmic bias against protected groups of interest.

One area of concern is the use of ML in judicial system pre-trial release decisions. In May 2016, Angwin et al. published an investigative report for ProPublica on the ML-based COMPAS (Correctional Offender Management Profiling for Alternative Sanctions) recidivism risk predictor in which they found it to be biased against black offenders. [1] The assertion of bias was based on confusion matrix analysis with recidivism as the positive class. The researchers found that the algorithm produced a significantly higher false positive rate (FPR) for black offenders than white (45% vs 23%), and significantly higher false negative rate (FNR) for white offenders than black (47% to 28%). [2] In a more in-depth statistical analysis, Alexandra Chouldechova of CMU found the imbalance between FPR and FNR by race persists regardless of the risk score threshold selected for classification and when controlling for covariates. [3]

Another potential problem concerns unequal access to the features of ML-based systems. In 2018, Buolamwini et al. tested commercial systems from Microsoft, Face++, and IBM that predict perceived binary gender from a head shot. Determining that existing facial analysis benchmarks were not well distributed across skin tone and gender, the researchers first created a more balanced benchmark. Using

a regional subset of this data to control for lighting and image quality, they found all three systems to have a significantly higher classification error rate on darker-skinned female faces (24%, 36%, and 33% for each system respectively) compared with lighter-skinned and/or male faces (at most 6%). [4]

Responsible ML is a nascent field. Research effort to date has focused largely on issues of algorithmic bias and fairness such as the examples above. In these domains, answers can seem straightforward, however in 2018, Verma et al. produced a survey of fairness definitions and measures that illustrates a major challenge to the area: there is no agreement as to what fairness, or a lack of bias, means. [5] Common approaches to fairness include:

- **Anti-classification** or "fairness through unawareness", under which protected attributes may not be used as input features. For example, a loan approval classifier may not use race or gender as a feature.

- **Classification parity**, under which some calculation of the confusion matrix (e.g. false negative rate) must be equal across protected groups. For example, equal proportions of men and women must be incorrectly denied credit.

- **Conditional statistical parity**, under which outcomes must be conditionally independent of protected attributes given relevant unprotected attributes. For example, men and women with similar age, income, married status, and payment history must be equally likely to receive credit.

While standardization on a definition may be possible, it is likely that no definition will satisfy everyone, as Corbett-Davies et al. have shown that all of the above approaches have statistical limitations and can, in fact, harm the protected groups they are trying to protect. [6] For example, in the earlier COMPAS case, if women are statistically less likely to reoffend, disallowing gender as an input feature will cause more women to be denied pretrial release.

The study of algorithmic bias has brought attention to the need that we consider the potentially disparate outcomes that arise from the use of our models and systems, but avoiding biased decision making should be the beginning, not the end, of our commitment to responsible ML. From the safety of robotic systems, to the privacy of user training data, to the ability of autonomous systems to obey the law, any kind of unintentionally-harmful outcome is worthy of attention and mitigation. The next two sections make the case for the two focus areas of this dissertation.

1.2 Responsibility and Training Data Privacy

Modern machine learning based applications rely on large data sets passively collected from their users. Detailed traffic estimation requires frequent sampling of mobile device location and velocity. Smart touchscreen keyboards leverage a massive corpus of prior text entry and correction. Customer retention algorithms use all data available to a company when determining if a special offer should be extended.

The storage, handling, and use of such private customer information is topic of growing concern. The traditional response has been to centralize to a secure data warehouse and concentrate resources on its protection. The data must still be accessible to learning algorithms or employees, however, and a large trove of potentially valuable information in one place makes a tempting target. Unsurprisingly, data breaches from centralized corporate data stores have become increasingly common, resulting in a White House consumer privacy report recommending data minimization as early as 2012. [7] In a 2015 effort to estimate the risks and costs of data breaches, Sen et al. aggregated government and institutional sources on the frequency, nature, and severity of such events. [8] At that time, they found that in the United States the organizational cost *per incident* was $5.9 million, the direct annual cost of identity theft to individuals was $16 billion, and 74% of data breaches exposed personally identifiable information. By way of example, two 2017 breaches gained popular me-

dia attention, when Equifax exposed the names, home addresses, and social security numbers of 148 million Americans [9] and the `ai.type` virtual keyboard app exposed the names, email addresses, and contacts of 31 million Android phone users. [10]

1.2.1 Federated Learning

In response to concerns over centralized data transmission and storage, McMahan et al. constructed a new paradigm of *federated learning* in 2017, in which user data is not centralized. [11] Instead, an iterative convergence process is undertaken in which local models are trained on each user device as updates to the current global model, and only trained model parameters are ever transmitted. Leaving user data on its device of origin reduces the risk of a data breach: one user's data can still be compromised by physically taking their phone, but there is no single place to acquire everyone's data at once.

There are still demonstrated privacy risks associated with federated learning. In the "white box" case, in which an adversary has access to a complete trained model, Nasr et al. demonstrate effective membership inference attacks, exploiting the privacy vulnerabilities of stochastic gradient descent to answer the question: Was *that* record used to train *this* model? [12] Even for the more challenging "black box" case, in which an adversary can only query the model as through an API, Shokri et al. demonstrate similar (albeit less effective) attacks. [13] Consider a medical condition like clear cell renal carcinoma, a rare form of kidney cancer. While some negative training examples will be included in such a model, a posterior estimate that an individual has the disease, conditioned on the knowledge that their medical record was used in training, would be significantly higher than a prior assumption based on the general population.

The privacy risks associated with federated learning can be mitigated by various means. Differential privacy, in which individual records are distorted with ran-

dom noise, offers a bounded trade-off between privacy and statistical accuracy. [14] Multi-party computation (MPC), in which parties cooperate to perform a calculation without revealing their true inputs, can assure privacy without any loss of accuracy to the output, subject to certain assumptions. [15] Both techniques have been used in the construction of privacy-preserving federated learning (PPFL), for example by Bonawitz et al. [16] The introduction to Chapter 3 gives a comprehensive overview of these ideas for those new to the area.

At the implementation level, most empirical studies of PPFL have followed one of two evaluation approaches. In the first approach each participating party is assigned to a separate Amazon Web Services (AWS) instance as in Hardy et al. [17] At 2021 pricing, AWS Elastic Cloud Computing (EC2) instances can cost as much as $1.25 per hour. If instances are geographically distributed, this approach accounts for communication (but not necessarily computation) load, but when large scale PPFL studies include 5,000 parties or more, it can significantly burden project budgets. To avoid these costs, the second approach relies on local simulation, which is taken to mean nested iteration in which each party's state is accessible as a set of object-oriented attributes as in Smith et al. [18] This approach is both faster and cheaper, but typically ignores communication load. If computation load is considered, it is reported as total runtime for the simulation, either with all parties acting serially or in perfect parallel. A recent survey of the field by its founders (Kairouz, McMahan, and others) acknowledges these challenges and notes that "the development of open software frameworks for federated learning research has the potential to greatly accelerate research progress". [19] One important aim of this work is to fulfill this need, as presented in the cryptographic adaptation of ABIDES in Chapter 3.

1. Alice sends Bob: (K_A, x_0, x_1), where $\forall i \in \{0,1\} : x_i \leftarrow PRG()$
2. Bob selects $b \in \{0,1\}$ and generates $k \leftarrow PRG()$
3. Bob sends Alice: $v = x_b + \{k\}_{K_A}$ (∗1)
4. Alice recovers the two possible values of k (∗2)
$$k_0 = \{v - x_0\}_{K_A^{-1}} \qquad k_1 = \{v - x_1\}_{K_A^{-1}}$$
5. Alice sends Bob two encoded messages:
$$m'_0 = m_0 + k_0 \qquad m'_1 = m_1 + k_1$$
6. Bob recovers $m_b = m'_b - k$ (∗3)

(∗1) Decryption $\left\{x_b + \{k\}_{K_A}\right\}_{K_A^{-1}}$ will fail, because $\{k\}_{K_A}$ was modified after encryption.

(∗2) Exactly one of k_0 or k_1 equals k, but Alice cannot know which one.

(∗3) Bob can recover m_b but not m_{1-b}, because $k_b = k$ but $k_{1-b} \neq k$.
Therefore $m'_b - k = m_b + k_b - k = m_b$, but $m'_{1-b} - k = m_{1-b} + k_{1-b} - k \neq m_{1-b}$.

Figure 1.1: A typical one-out-of-two oblivious transfer algorithm. Note that K_A and K_A^{-1} respectively denote Alice's public and private keys, $\{M\}_{K_A}$ denotes processing message M with key K_A (i.e. encrypting/decrypting), and $X \leftarrow PRG()$ denotes drawing a large value X from an appropriate pseudo-random generator.

1.2.2 One-Out-of-Two Oblivious Transfer

Another aim is improve privacy under federated learning scenarios in the particular case where all other parties collude to expose the private information of a single honest party. This is achieved starting with a recent state of the art PPFL protocol that incorporates both MPC and differential privacy to secure against outside observation or unwanted inferences by the server that coordinates the learning process. A novel mechanism is added based on one-out-of-two (or 1-2) oblivious transfer, a powerful technique from the field of public key cryptography. [20]

As depicted in Figure 1.1, using the customary cryptographic party stand-ins, Alice and Bob, 1-2 oblivious transfer allows a party (Party A) to offer two pieces of information (messages) to a second party (Party B), with the intention that Party B may select exactly one message to decode. The transfer is called one-out-of-two

because it is not possible for Party B to successfully decode the non-selected message, and is called oblivious because it is not possible for Party A to determine which message was selected. The improved privacy protocol using 1-2 oblivious transfer is presented in Chapter 4.

1.3 Responsibility and Safe Autonomous Behavior

Unintended outcomes in deployed ML-based systems have already had negative consequences, including problems stemming from algorithmic bias as discussed in Section 1.1. As deployed systems transition away from having a human decision maker (a judge or credit analyst) in the loop, to more autonomous direct action, the potential range of unintentional negative outcomes will increase. Autonomous agents with physical presence have an obvious capacity for accidental harm, such as a construction robot crushing a human worker or inflicting expensive damage to itself. But even virtual autonomous agents may wreak havoc by downloading illegal content from the internet or engaging in unlawful financial transactions, all without the knowledge or intent of the owner, or even the creator.

1.3.1 Reinforcement Learning

In contrast with instantaneous classification or regression predictions, for example "Does this image contain a cat?" or "What is the quality rating of this wine?", an interesting subset of ML problems require an agent to make a sequence of decisions over time. These decisions may involve uncertainty about the environment or the future. The environment may contain significant positive outcomes that are hard to find, or are "hidden" behind smaller negative outcomes. There may be no-win situations that cannot be anticipated and from which there is no way to avoid a severe negative outcome. This class of problem is typically constructed as a Markov decision process (MDP).

An MDP can be thought of as a state machine in which the transitions are governed by a probability distribution conditioned on the action attempted by an agent. [21] After each action, the agent may be assigned a numeric reward r, and it will attempt to maximize these rewards over the long term. For example, in a simple computer game, taking the action "swing sword" from the environmental state "in combat with monster" may lead to either the "struck monster" state $[Pr(0.5), r = +1]$ or "missed monster" state $[Pr(0.5), r = 0]$, while the action "run away" may lead to either the "out of combat" state $[Pr(0.7), r = +2]$ or "in combat with monster" state $[Pr(0.3), r = 0]$. Classical solutions to MDP involve dynamic programming (DP) approaches like value iteration or policy iteration, in which the agent is supplied all information concerning states, actions, transition probabilities, and rewards in advance, and solves some variation of the Bellman equation:

$$V^{\pi *}(s) = \max_a \{R(s,a) + \gamma_{s'} P(s'|s,a) V^{\pi *}(s')\} \tag{1.1}$$

to produce a reward-maximizing *policy* which maps every state to its optimal action. [22] Only then does the agent begin to interact with the environment by following this optimal policy.

Reinforcement learning (RL) is the name given to a collection of approaches that solve MDPs when the problem definition must be discovered through experimentation with the environment. [23] RL techniques can be broadly divided into model-based and model-free. Model-based approaches to RL employ some experimental method to approximate the transition probabilities and rewards, then use DP methods to compute the optimal policy. Model-free approaches do not attempt to obtain a complete definition of the problem and environment, but instead directly learn the overall utility of taking a certain action from a particular state.

Perhaps the most common approach to model-free RL is Q-learning, under which

the agent requires minimal information: at each time step, it must be given a unique identifier for the current state of the environment and for each action that it might take. [24] It is not necessary that the agent understand what the states or actions represent, only that the identifiers be consistent. The agent will explore its world by sequentially receiving an environmental state, selecting an action, then receiving a reward and an updated environmental state. It must use this information to learn the optimal action a for each state s by optimizing the Q-function:

$$Q^{\pi}(s,a) = R_s(a) + \gamma \sum_{s'} Pr_{ss'}[\pi(s)]V^{\pi}(s') \tag{1.2}$$

which represents the sum of immediate and discounted expected future rewards that will result from doing a while in s and then continuing to follow policy π. R_s is the reward function for state s, γ is the stepwise discount for future rewards, $Pr_{ss'}[\pi(s)]$ is the probability of a transition from state s to new state s' under policy π, and V represents the utility or value of reaching a given state.

A full discussion of the many fundamental challenges in RL is beyond the scope of this overview, but I present several for consideration. The temporal *credit assignment problem*, first explored by Marvin Minsky in 1963: given a sequential problem with delayed rewards (e.g. winning a game of chess), how can a learning system know which actions contributed to the eventual reward and which did not? [25] The *exploration-exploitation dilemma*, as surveyed by Sebastian Thrun in 1995: given that an RL agent is meant to maximize its long-run rewards, how should it divide time between obtaining the rewards of which it is aware or exploring the environment in search of potentially better rewards? [26] Problems of the environment including *non-stationarity*, *partial observability*, and the *curse of dimensionality*: how can an agent learn when its world changes too quickly, cannot be effectively measured or understood, or requires granular enough state information that it is impossible to

fully explore? [27] In this dissertation, I focus on a different problem: how to best ensure that an RL agent's learned policy does not accidentally incorporate unwanted behavior?

1.3.2 Safe Reinforcement Learning

There are effective techniques to guide the behavior of autonomous agents away from negative outcomes to certain kinds of problems, many of which are outlined in Garcia's "Safe Reinforcement Learning" survey. [28] Hutter has considered *ergodic MDPs*, for which he gives the definition: "An MDP μ is called ergodic if there exists a policy under which every state is visited infinitely often with probability 1." [29] This implies the principle of *recoverability* as introduced by Ryabko et al., requiring that the environment contain no "traps" from which it is no longer possible to achieve the best value. [30] Other authors consider risk-sensitive optimization criteria. Gosavi employs *variance-penalized control*, which applies a negative reward modification proportional to the long-run reward variance of a policy. [31] Giebel et al. limit outcomes to a set of *feasible* policies under which the probability of reaching an error state is thresholded by a configurable parameter. [32]

Such techniques are appropriate and effective for embodied agents subject primarily to the laws of physics, or even to virtual agents concerned with avoidance of immediately measurable consequences. For example, *action rejection* is sufficient to avoid a helicopter agent crashing or a bipedal robot agent falling over: disallow helicopter actions that would exceed limits on a combination of relative altitude, trajectory, and speed; or disallow bipedal robot actions that would move its center of gravity past a tipping point. For an investment agent, a penalty proportional to the variance of (monetary) returns or a requirement to keep the probability of bankruptcy below a threshold, are perfectly sufficient. There are, however, task components that would thwart these safety-focused approaches:

- Failure modes are imprecisely defined or objectively similar to target behavior.
- Failure modes are impossible to detect or penalize during training.
- The acceptability of individual actions is difficult to evaluate.
- There are negative outcomes not directly related to primary task failure.

An example of a failure mode that fits all of these criteria would be the algorithmic violation of a law. Many laws reference a guilty state of mind, require subtle interpretation, and result in penalties (which should be incorporated to the rewards) only years later. My work on normative autonomous stock trading in Chapters 5 and 6 concerns exactly this challenging scenario.

1.3.3 Spoofing in Financial Markets

The Commodity Exchange Act (CEA), as modified by the Dodd-Frank Wall Street Reform and Consumer Protection Act (Dodd-Frank), makes it unlawful to engage in order "spoofing", which it defines as "bidding or offering with the intent to cancel the bid or offer before execution". [33] In their guidance on the topic, the Commodity Futures Trading Commission (CFTC) lists some reasons a party might engage in spoofing, such as: overloading a price quotation system, delaying another party's trade executions, creating the false appearance of supply or demand, or creating artificial price movements upward or downward. [34]

In 2012, Lee et al. presented evidence of widespread spoofing with a price manipulation motive based on a custom data set provided by the Korea Exchange (KRX). [35] A plurality of the manipulation was traced to day traders seeking quick round-trip profits from unnatural price distortions, with the remainder serving to improve the entry or exit price of a longer-term investment strategy. According to a 2015 member survey from the CFA Institute, an international organization certifying Chartered Financial Analysts, *market fraud* and *market trading practices* account for a significant

proportion of surveyed members' "most serious ethical issue facing local market": 49% in China, 36% in Japan, and 38% each in the UK and US. [36]

The given legal definition of spoofing relies on the intent of a market actor. Fortunately, exploration of the "intent" of a black box learning algorithm which produces unexpected behavior is moot for the immediate question, because interpretive guidance from the CFTC clarifies that, while "reckless disregard" can be considered a violation of other sections of the CEA, "reckless trading, practices, or conduct will not constitute a 'spoofing' violation". [34] It thus appears that, while training an autonomous trading agent to spoof by design is probably a violation, a "recklessly" unconstrained profit-maximizing agent (that learns spoofing behavior) is probably not. However, the Dodd-Frank Act grants the CFTC regulatory authority to prohibit spoofing "and any other trading practice that is disruptive of fair and equitable trading", with the clear implication that spoofing is considered disruptive of fair and equitable trading, or in other words, non-normative. [37]

Market manipulation is not only of academic interest. As summarized in a 2012 survey by Putniņš, it has been demonstrated in both theoretical and empirical economics literature to be profitable, implying harm to other traders from whom the excess profits must be extracted. [38] Spoofing in particular has become increasingly practical with the decimalization of stock prices and the automation of stock exchanges through computing technology, resulting in a recent spate of enforcement actions. For example, in the year 2020 alone, the CFTC filed and resolved 16 spoofing cases. [39] The largest settlement, that against J.P. Morgan Chase, included total monetary relief of $920 million, suggesting that this is a keen area of interest for financial regulators today. [40] However, there are significant obstacles to the study or detection of spoofing in financial markets, in particular the sparsity and anonymity of available data. One aim of this work is to improve spoofing detection through the generation of a large synthetic data set suitable for use in a spoofing behavior

classifier.

Suppose that a responsible practitioner of financial machine learning undertakes to create an RL-based stock trading agent with an input space consisting of various features derived from technical stock market data such as price and volume, along with sufficient internal state (such as current holdings) to render the problem Markovian. The output space will be flexibly designed to permit placing or cancelling orders to buy or sell stock at various offsets to the best available pricing at the time. The practitioner hopes that, given the nature and power of model-free RL, the agent will discover an optimal, or at the least very profitable, trading strategy. It is entirely plausible that such a model, unconstrained, will discover that the optimally profitable solution is through a disruptive price manipulation technique such as spoofing. If the practitioner anticipates this negative potential outcome, and desires to affirmatively avoid it, how could that be achieved? This will require the agent to both identify its emerging non-normative spoofing behavior and to accept potentially lower financial gains by eschewing it. I develop a suitable spoofing classifier using synthetic data in Chapter 5 and use it as normative guidance to construct a non-spoofing RL trading agent in Chapter 6.

1.4 Responsibility and Agent-Based Simulation

In their classic textbook *Artificial Intelligence: A Modern Approach*, Stuart Russell and Peter Norvig define an *agent* as "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators." [41] They suggest, but do not require, that an agent should be *autonomous* rather than rely solely on the prior knowledge of its designer. Michael Wooldridge, in his textbook *An Introduction to Multi-Agent Systems*, adds the requirement that an agent be "capable of autonomous action in [its] environment in order to meet its design objectives". [42] I have adopted the term *autonomous agents* to emphasize my focus on

agents whose behavior is informed by their own observations and learning processes, and whose actions are not directly moderated by a human supervisor.

An agent-based model (ABM) is one formed by a collection of autonomous agents that interact with their environment, including other agents, through a set of internal rules to achieve their objectives. [43] An agent-based *simulation* (ABS) is typically used when mathematical solution or real-world study of an ABM are impractical or impossible, and instead a computer representation is executed to obtain empirical observations through some experimental process. [44] Agent-based modeling and simulation has advanced research in many domains [45, 46], including the social sciences [47], computational economics [48], and marketing. [49] A multi-agent system is simply one "composed of multiple, interacting agents" [50], and like ABM there is no inherent requirement that it be a multi-agent *simulation* (MAS).

The use of the above terms is fluid. Siebers et al. consider ABS to be one of several approaches to MAS, in which an investigator "describes the decision processes of simulated actors at the micro-level", contrasted with top-down or purely mathematical approaches. [51] Niazi et al. consider the terms to be roughly interchangeable net of domain, with engineering and computer science tending to refer to MAS, while other sciences and economics refer to ABM. [52] Still other fields refer to the same general approach as individual-based models or generative social science. The work here is entirely in simulation, and I use the terms MAS and ABS interchangeably to indicate a computer simulation of multiple interacting intelligent agents in a dynamic environment.

1.4.1 Financial Market Simulation

In an influential early work, Daniel Friedman considered the strengths and weaknesses of three major approaches to one of my major application areas: empirical financial market research. [53] He concluded that field studies are clearly relevant,

but do not provide experimental access to all relevant information; laboratory studies improve control and observation, but are of necessity small and expensive; while computer simulations feature perfect control and observation. However, he notes a significant shortcoming, that a "trader's strategies are not endogenously chosen, but rather must be specified exogenously". I find Friedman's analysis relevant across empirical domains and mitigate his criticism through the introduction of intelligent agents, such that participant behaviors *are* endogenously learned.

There is a rich history of financial market simulation in the economics literature. Contemporaneously with Friedman's analysis, Gode and Sunder in 1993 introduced the Zero Intelligence (ZI) agent which stochastically generates bids and offers from an IID uniform distribution covering the entire pricing spectrum, and found that even this trivial agent could reproduce the allocative efficiency of a real continuous double auction (CDA) like the stock market. [54] From this, they conclude that it is the structure of the market, rather than the behavior or rationality of the individual participants, which produces some of its key properties. In 1997, Cliff and Bruten added to these agents an individual desired profit margin, and the ability to alter that margin in response to the most recently placed order. [55] They call the resulting agent Zero Intelligence Plus (ZIP) and find that, unlike original ZI, it consistently produces a market equilibrium close to levels both expected in theory and observed in laboratory market experiments with human traders. Additional capabilities have been added to trading agents in numerous following works to more closely approximate various features observed in real markets, including the maximization of expected profit surplus by estimating the likelihood of successful order transaction based on a limited memory of order history, [56] and the addition of a strategic threshold parameter which permits agents that normally require a minimum expected profit to accept lower values when profit is guaranteed. [57]

A key contribution of this dissertation is the construction of a novel, open-source

Agent-Based Interactive Discrete Event Simulation (ABIDES) framework in Python 3. ABIDES is used for all of the empirical work presented herein, but having been previously published, it also forms the basis of other published research, a selection of which is listed here. Vyetrenko et al. curated a large collection of known stylized facts about the stock market to serve as realism benchmarks and established baseline metrics for how well various ABIDES configurations reproduce those facts. [58] They found that current ABIDES configurations accurately reproduce some stylized facts, like aggregation normality and volatility clustering, but not others, like intraday volume patterns and interarrival times, and suggested future realism enhancements. Balch et al. leveraged ABIDES to evaluate whether historical prices or stochastic processes can better model the impact of large trades on the stock market, finding that stochastic processes produce an "impact curve" closer to real observations. [59] Mahfouz et al. used ABIDES as a platform for opponent modeling, in which each agent explicitly models the policies of other agents and uses these models to inform its own actions. [60] In this work, they primarily attempt to assign opponents to strategy clusters and evaluate the accuracy of the approach. Karpe et al. implement double deep Q-learning (DDQL) within ABIDES and take a multi-agent reinforcement learning (MARL) approach to the optimal execution problem, in which a trader (presumed to be acting on behalf of a client) must liquidate a specified set of inventory by a given deadline at the best overall price. [61] The architecture and features of the ABIDES framework are presented in Chapter 2.

1.4.2 Multi-Agent Simulation for Responsible Machine Learning

An underlying theme of this dissertation is the idea that agent-based simulation (ABS), and particularly multi-agent simulation (MAS), can solve many of the challenges faced by machine learning practitioners. Experiments in MAS allow direct observation of important latent factors, such as the intent of an actor. Stochastic MAS

provides a natural form of data augmentation, which can improve model performance for important edge cases with sparse training coverage, such as near collisions in aviation. Developing, training, and evaluating a learning agent in MAS can ameliorate potential serious harms certain agents might inflict during the training process, such as driving on a sidewalk or crashing a financial market. For all these reasons I believe that, when included as part of mechanism design and model development, MAS can reduce the likelihood that our ML-based systems display unexpected non-normative behaviors after deployment. MAS can also encourage adoption of private and secure learning techniques by making them cost effective in expensive domains such as distributed computing at scale, where the alternative may be to rent thousands or millions of geographically distributed computing nodes. I aim to position MAS as a training cornerstone for autonomous agents in precisely these cases where a sufficiently normative and general model cannot or should not be learned through direct interaction. For current purposes, this is my interpretation of *responsible machine learning* and the overarching motivation for the remainder of the work.

CHAPTER 2
ABIDES: TOWARDS HIGH-FIDELITY MULTI-AGENT MARKET SIMULATION

In this chapter I introduce ABIDES, an open source agent-based interactive discrete event simulator. ABIDES is designed as a flexible, multi-domain, multi-agent simulation (MAS) framework. The initial version of ABIDES was developed to be "batteries included" for agent-based research in market applications because this domain features: well-defined mechanisms but complex and unpredictable emergent behaviors; a relative scarcity of literature, particularly with learning agents; and a lack of publicly available high-fidelity market simulation environments. In this context, ABIDES enables the simulation of tens of thousands of custom trading agents interacting, through a realistic network model, with an exchange agent to facilitate transactions.

I discuss the motivation and design of the simulator, illustrate its use and configuration with sample code, and present results that demonstrate its abilities to: reconstruct noisy historical trading days (e.g. for data augmentation), conduct event studies on the impact of large transactions, and examine complex phenomena like high-frequency trading. The market simulation and agents presented in this chapter form the basis of my normative "learning not to spoof" efforts in Chapters 5 and 6. ABIDES is extended with new agents and features to a second domain, the simulation of privacy-preserving federated learning protocols, in Chapters 3 and 4.

2.1 Background

ABIDES (Agent-Based Interactive Discrete Event Simulation) is intended to facilitate the creation, deployment, and study of strategic agents in a highly configurable market environment. As discussed in Section 1.4.1, we were inspired by Daniel Freidman's

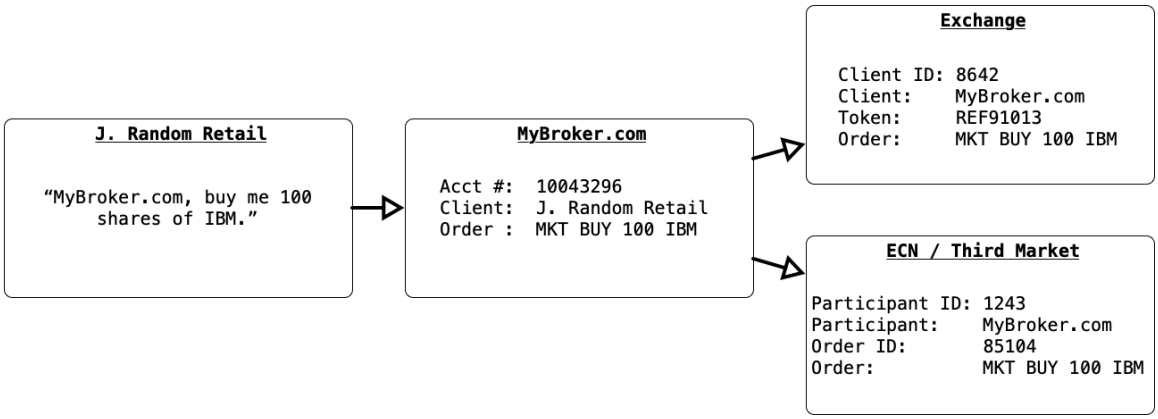


Figure 2.1: Simulation allows agent-identifiable data which is lost in the flow of real-world orders.

view that simulation of markets provides a powerful tool to analyze individual participant behavior as well as overall market outcomes that emerge from the interaction of the individual agents [62]. Freidman conducted a review of empirical approaches to the analysis of continuous double auction (CDA) markets such as NASDAQ and the New York Stock Exchange, concluding that computer simulation provided important benefits to experimental control and observation, but with the critical weakness that a designer must specify each participant's strategy in detail.

Accordingly, simulation provides an attractive platform for research in equity trading questions. With ABIDES, we aim to address Freidman's primary concern regarding computerized market simulations – that strategies must be exogenously specified – by enabling powerful *learning* agents to participate in a realistically structured market via a common framework. We believe this is necessary to properly investigate the behavior and impact of intelligent agents interacting in a complex market environment. ABIDES is a curated, collaborative open-source project that provides researchers with *tools* that support the rapid prototyping and evaluation of complex market agents. With it, we hope to further empower researchers of financial markets to undertake studies which would be difficult or impossible in the field, due to the

absence of fine-grained data identifiable to individual traders (see Figure 2.1), a lack of knowledge concerning participant motivation, and an inability to explore historical counterfactuals.

2.1.1 Discrete Event Simulation

Modern financial markets allow transactions to be concluded at an exchange anytime the market is open. Simulations of these systems rely on representations of entities within the simulated environment to estimate the state of the system at times after initialization. When constructing a simulation, a fundamental decision is therefore the handling of time. It can be considered continuous, as in the Black Scholes options pricing model which relies on geometric Brownian motion; in this case the system state can be estimated at any arbitrary time by solving a differential equation. [63, 64] Or it can be considered discrete, such that system state can change only at the edges of discrete time slices. In the discrete case, there are then two specific approaches to chronological advancement: fixed time and next event. [65] In the first of these, each agent is polled for potential action at every time in a series of fixed increments which can be neither subdivided nor skipped. This works well for simulations modeling physics or computer games such as StarCraft II, in which there is a natural "refresh rate". [66] In systems where many time units will contain no state changes, however, it is advantageous to immediately advance to the next scheduled event without simulating the intervening time. This is called discrete event simulation (DES), and it permits efficient handling of agent actions that are sparsely distributed through time. As an example, our financial simulations include high-frequency traders who may act every few microseconds; because their arrival order to the market is critical, we must therefore use nanosecond resolution for the overall simulation. However, there would be no sense in polling retail agents, who act a few times per day, at each nanosecond. This combination of flexibility and efficiency is the reason we adopt DES

as our temporal approach to simulation.

Discrete event simulations often use random variables as models. For example, a Poisson distribution might be used to determine the periods between arrival times (inter-arrival time) of phone calls at a call center. Discrete-event models are therefore better suited than continuous-time models when underlying parameters must vary over time. A discrete event simulation system also is fast and efficient because the time between state changes can be ignored and skipped over. Further, DES are amenable to computational parallelization that further speeds up execution. [67, 68]. While ABIDES is currently not parallelized nor distributed at the agent level, its software design follows principles of distributed simulation. For example, "agents" in ABIDES are mapped to logical processes and ABIDES progresses in time by scheduling events for these agents.

2.1.2 Agent-based Economics

Several emerging simulation applications, like the modeling of the stock market, call for an agent-based view. An agent-based model (ABM) is a model that is formed by a set of autonomous agents that interact with their environment (including other agents) through a set of internal rules to achieve their objectives. Agent-based modeling and simulation (ABMS) is useful, usable, and already used in a variety of application domains [69]. ABMS helps research and investigation in social sciences [47], computational economics [48], and marketing [49]. Many agent-based simulators have been developed (e.g., Swarm [70] and Mason [71]). ABIDES entities, mapped to logical processes, are indeed agents. Consequently, ABIDES provides both performance and efficiency leveraging from the design of PDES, and flexibility and familiarity of an of an agent based interface leveraging the growing literature in ABM.

Agent-based financial market simulation has been shown to be an effective approach when agents can learn and adapt to different investment strategies [72]. In

the financial literature there are simulators that use learning behaviors with differing perspectives of past data [73]. Financial market approaches are either synchronous or asynchronous. Levy et al [74] propose a synchronous approach, but we believe that asynchronous approaches are more flexible and scalable. This view is shared by Jacobs et al [73, 75], who proposed a framework called JLMSim. JLMSim is a discrete event simulator that incorporates trading rules (albeit simple strategies) and reproduces the changes in the market by executing buy and sell orders from the order book. In ABIDES we represent individual investors as agents, which is similar to the approaches of Levy et al [74] and Jacobs et al [73]. JLMSim is implemented in C++ and runs at a few thousands events per second on a 2.4 GHz PC. ABIDES is implemented in Python (which offers MATLAB-like functionality) and can run over 10K events per second on a similar 2.4GHz processor. Unlike ABIDES, JLMSim does not provide interfaces to implement complex trading strategies or learning agents, and is currently not available as open source software.

The success of agent-based economics has led to the development of a number of other simulation platforms such as those on which J. Wang et al. have reported their results. [76] X. Wang and Wellman have used their own simulation platform to study spoofing agents in a market environment populated by zero intelligence (ZI) and heuristic belief learning (HBL) traders [77]. Their approach analyzes the results from an empirical game-theoretic view [78]. ABIDES is a fresh implementation incorporating lessons learned from prior platforms.

2.2 Important Questions About Markets That Simulation Can Help Us Address

ABIDES can support a number of different kinds of investigations into market behavior that are not easily conducted using historical data or live experiments.

- **The benefits of co-location**: In the past 20 years hedge funds and other

market participants have invested in the deployment of computing resources at major exchanges [79]. This so-called "co-location" enables quicker access to market information than if the trading server were located further away. It is not feasible to investigate the value of the advantage co-location provides with available historical data, because it does not include information about the geographic location, network latency, or network reliability of each actor. With a platform that does not require formal arms-length messaging using a realistic network model, we cannot simulate the effects of these factors even if they are known. ABIDES provides a network model and mandatory messaging protocol that enables detailed experiments in this area: Creating a population of agents with a realistic and known distribution of network latency, jitter, and reliability; conducting trials in which one agent, pursuing a low-latency order book imbalance strategy, is incrementally shifted from a co-location facility out to a great distance; and evaluating the impact of this shift on all agents' profitability while otherwise pursuing the same strategies.

- **The impact of large orders on price**: The very act of placing orders in a market may affect the price. For instance, if there is significant selling pressure evidenced by a large volume of sell orders, it is generally expected that the price will go down. The extent to which the price moves because of an order is referred to as *market impact*. Market participants of course want to minimize such impact, because the market usually moves contrary to their profit incentives. In a market field study, it is not feasible to perform controlled A/B tests. One cannot place a market buy at the NYSE for one million shares of IBM at 10 AM on Oct 22, 2018, and then also *not* place that order, and compare the difference. Without the "control", any observed result from the large order could be attributable to some other factor. A key feature of ABIDES is the ability to re-simulate the same historical market day with known, limited changes while

holding all other factors constant, thus enabling the desired experimental control population.

- **Cost-benefit analysis of AI**: In a simulation without a model for computational time delays that directly impact time-to-market for the resulting orders, we cannot readily study the trade-off between simpler, faster predictors and slower, more powerful predictors. ABIDES introduces a flexible, integrated model for computation delay that permits the "speed" of each agent's thought process to be represented, and to have that representation affect the timing of all of outbound messages as well as the next time at which the agent can be roused for participation. These computation delays can be specified, or can be measured and applied in real time during simulation, such that an agent is delayed according to the actual runtime of each computation. Thus heavier thinkers will take longer to deliver a resulting order to the exchange and will be unable to act as frequently.

- **Explanation of behavior**: When analyzing historical market data, we cannot know the motivation behind individual trader actions, but *intent* is a key component of many market regulations. For example, "spoofing" (placing limit orders one does not intend to fulfill) is not permitted in U.S. markets. It is also extremely difficult to detect or study, because an identical pattern of placing and canceling orders may be lawful or unlawful depending on the trader's intentions. Similarly, with the shift away from knowledge-based AI toward "black box" ML models, explaining the actions of intelligent agents has become more challenging. ABIDES provides a platform that features high-resolution time-synced event logging and visualization for: trading agent state, portfolio, strategy, and orders; exchange agent order books, order handling, and order execution; and any extrinsic price-time series used to guide value-conscious strategies. Combin-

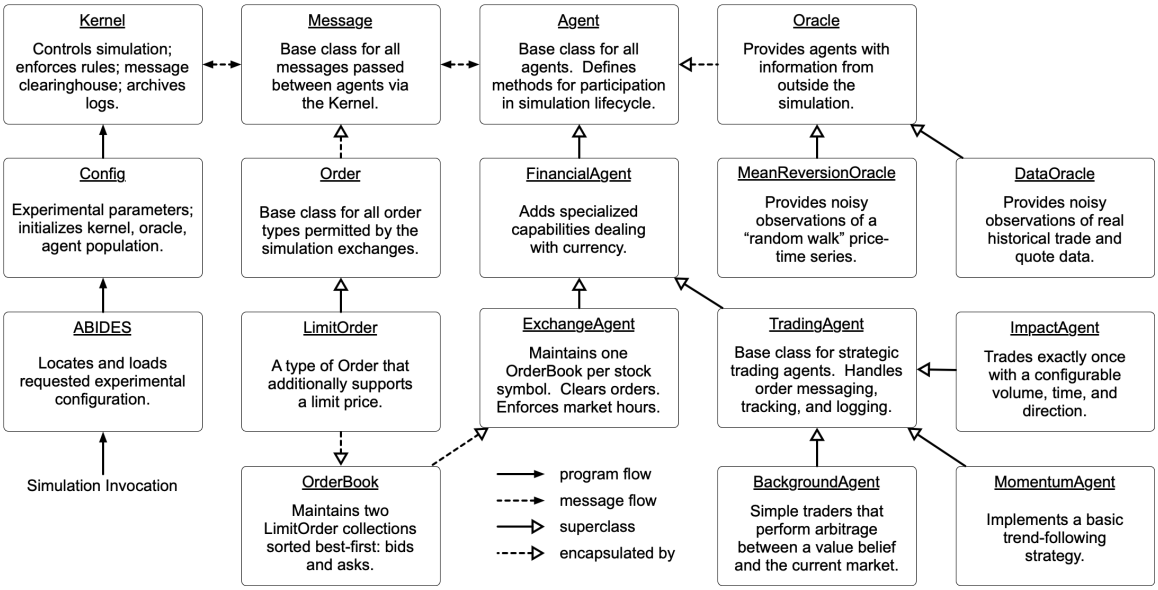


Figure 2.2: Class relations within the ABIDES simulation framework.

ing the precision logs with a quality simulation architecture which requires all inquiries and impulses to pass as messages through a central Kernel for scheduling and tracking, such that each agent's decisions, intentions, communications, and results for every action are fully visible, we produce the full scope of information needed for explanatory reconstructions. We hope to use this ability to dive deeply into the *why* of trading policies learned by agents or observed in real markets.

2.3 ABIDES Architecture

The ABIDES framework includes a customizable configuration system, a simulation kernel, and a rich hierarchy of agent functionality, partially illustrated in Figure 2.2.

2.3.1 Functions and Features of the ABIDES Kernel

ABIDES is built around a discrete event-based kernel [80] which is required in all simulations. All agent messages must pass through the kernel's event queue. The kernel supports simulation of geography, computation time, and network latency.

It also acts as enforcer of simulation physics, maintaining the current simulation time, tracking a separate "current time" for each agent, and ensuring there is no inappropriate time travel. Some key features of the ABIDES kernel include:

- **Historical date simulation** All simulation occurs on a configurable historical date. This permits "real" historical information to be seamlessly injected into the simulation at appropriate times when required for a particular experiment. ABIDES can currently be configured to run a market replay with a liquidity injection agent placing orders from historical data, a historical agent-based simulation in which background agents can receive noisy observations of historic transactions, or an ahistoric agent-based simulation in which fundamental stock values follow a mean-reverting or other mathematical process.

- **Nanosecond resolution**: Because we seek to emulate real markets, we simulate time at the same resolution as an example exchange: the NASDAQ. All simulation times are represented as timestamp objects with nanosecond resolution. This allows a mixture of agents to participate in the simulation on very different time scales with minimal developer overhead. In the unlikely case that multiple events occur in the same nanosecond, they are handled in order of event object creation.

- **Global Virtual Time (GVT)**: GVT is the latest simulated time for which all messages are guaranteed to have been processed. The kernel tracks GVT as the simulation progresses. Since the simulation is single-threaded (although agents act in *simulated* parallel), it is not possible for any agent to affect the past. GVT may thus simply advance to the delivery timestamp of each dequeued message in chronological order and remain monotonically non-decreasing. It is usually the case that GVT advances much more quickly than wall clock time, but for very complex scenarios, it may not. The GVT value is not available to

the agents.

- **Current time per agent**: The kernel tracks a "current time" per individual participating agent which is incremented upon return from any activation of that agent. In situations where the current time for the agent is "in the future" (i.e., larger than GVT), the kernel will delay delivery of messages or wakeup calls to this agent until GVT catches up.

- **Computation delay**: The kernel stores a computation delay per agent which is added to the agent's "current time" after each activity. The delay is also added to the sent time and delivery time of any outbound message from an agent to account for the agent's computation effort. Agents may alter this computation delay to account for different sorts of computation events, or the simulation can be configured to measure and use the *real* computation time of each agent action.

- **Configurable network latency**: The kernel maintains a pairwise agent latency matrix and a realistic cubic network jitter model which are applied to all messages between agents. This permits simulation of network conditions and agent location, including co-location.

- **Deterministic but random execution**: The kernel accepts a single, global pseudo-random number generator (PRNG) seed at initialization. This PRNG is then used to generate seeds for an individual PRNG object per agent, which must rely solely on that object for stochastic methods. Since our system is currently single-threaded, this allows the entire simulation to be guaranteed identical when the same seed is initialized within the same experimental configuration. This would not ordinarily permit the desired A/B testing, because the "agent of change" might consume an additional pseudo-random number from the sequence and thus change the stochastic source for all subsequent agents.

> Because of our careful use of the primary PRNG only to generate subsidiary PRNGs per agent, the "agent of change" in an ABIDES A/B experiment will not alter the set of pseudo-random numbers given to any other agent throughout the simulation, even if it uses more or fewer such inputs for its changed activity. In this way, changes in the behavior of other agents will be caused by a changed simulation environment (e.g. stock prices) and not simple stochastic perturbation.

During a simulation, the kernel follows a typical series of life cycle phases: kernel initialization, kernel start, event queue processing, kernel stop, and kernel termination. All except the event queue processing phase consist entirely of sending a corresponding event notification to all agents.

While processing the event queue, the kernel extracts the next scheduled event in chronological order and advances the global virtual time (GVT) to match it. Recall that each agent has an individual "current" time representing the conclusion of its most recent activity. If the target agent is still in the future with respect to GVT, the event is rescheduled for the target agent's current time, placed back into the priority queue, and the kernel moves to the next chronological event. Otherwise, the target agent's current time is advanced to the GVT and the event is dispatched to the agent. When the agent's event handling method returns, the agent's current time is advanced by its computation delay.

Agents may request several critical functions from the kernel: To send a message to another agent; To schedule a wakeup call for some future time; And to learn the simulation identifier of another agent of a specific type (for example, a stock exchange). Messages will be sent as of the sender's current time, plus its computation delay, plus an optional additional delay upon request. Message receipt will be scheduled based on the send time plus network latency and jitter. Agents may learn the numeric identifier of other agents, but may never receive a *reference* to another agent

(as this could allow bypassing the kernel in the future).

2.3.2 The Agent Class

All participants in a simulation must inherit from a base agent class, which implements a number of required methods that allow participation in the full life cycle of the simulation.

The simulation lifecycle methods for kernel initialization, kernel start, kernel stop, and kernel termination must be supported by all simulation agents and will be called exactly one time per agent by the kernel. The order in which agents are activated in each life cycle phase is arbitrary. The basic agent class provides sensible default behavior for each phase.

Two simulation activation methods, for receipt of messages and wakeup calls, must also be supported by all simulation agents. These are called repeatedly by the kernel in order of increasing delivery timestamp of queued messages and wakeup calls. The basic agent class handles these methods by simply updating its internal current time and displaying an informative message.

While not required by the simulation kernel, the basic agent class also provides functionality for fine resolution timestamped activity logging and serialization to disk.

2.3.3 The Exchange Agent Class

The provided exchange agent inherits from the basic agent class and represents a stock exchange such as NASDAQ. The functionality of the message protocols supported by this agent is loosely based on NASDAQ's published ITCH and OUCH protocols. [81, 82] The exchange is initialized with market opening and closing times, which it will enforce. These are not required to match the simulation start and stop times. The exchange agent is not privileged in any way; it must participate in the simulation just as any other agent.

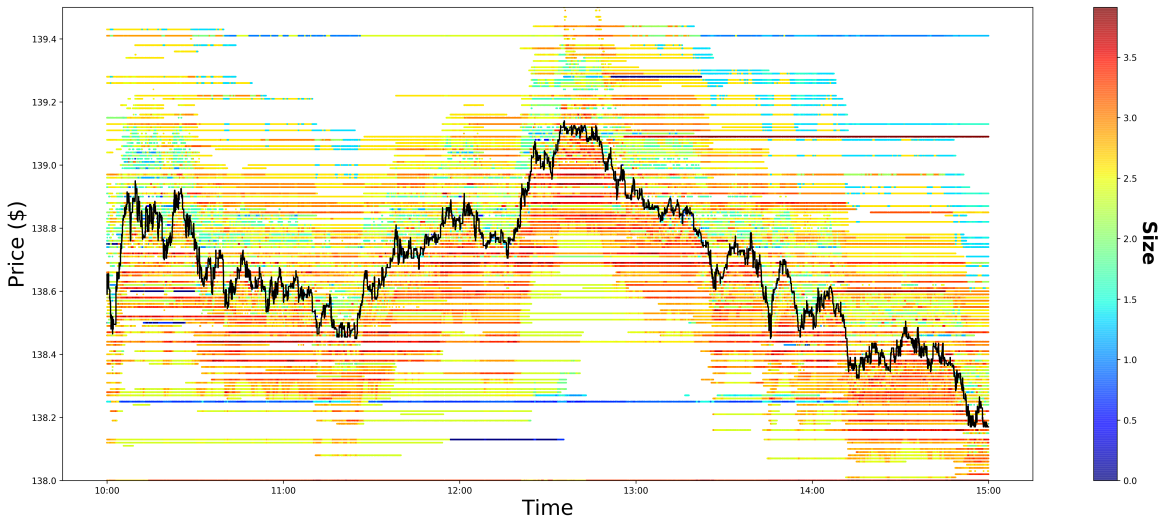


Figure 2.3: Example simulation of IBM stock for 2019-06-28.

The exchange agent understands how to respond to these types of messages that are specific to the operation of a financial market:

- **Market Open Time**: Returns the timestamp at which the exchange will begin processing order-related messages.
- **Market Close Time**: Returns the timestamp at which the exchange will stop processing order-related messages.
- **Query Last Trade**: Returns the last trade price for a requested symbol. Until the first trade of the day, the exchange reports the oracle open price (historical or generated data) as the "last trade price". The exchange does not yet implement the opening cross auction.
- **Query Spread / Depth**: Returns a list of the N best bid and best ask prices for a requested symbol and the aggregate volume available at each price point. With a requested depth of one, this is equivalent to querying "the spread".
- **Limit Order**: Forwards the attached limit order to the requested symbol's order book for matching or acceptance. Agents currently simulate market orders using a limit order with an arbitrarily high or low limit price.

- **Cancel Order**: Forwards the attached order to the requested symbol's order book to attempt cancellation.

Outside of market hours, the exchange will only honor messages relating to market hour inquiries and final trade prices (after the close). The exchange sends a "market closed" message to any agent which contacts it with disallowed messages outside of market hours.

The exchange agent demonstrates one use of the inbuilt Kernel logging facility, recording either the full order stream or snapshots of its order books at a requested frequency, enabling extremely detailed visualization and analysis of the order book at any time during simulation. For example, Figure 2.3 shows a "market replay" style simulation of IBM stock on June 28, 2019, in which autonomous trading agents can also participate and affect the market.

2.3.4 The Order Book

Within an Exchange Agent, an order book tracks all open orders, plus the last trade price, for a single stock symbol. All order book activity is logged through the exchange agent. The order book implements the following functionality:

- **Order Matching** Attempts to match the incoming order against the appropriate side of the order book. The best price match is selected. In the case of multiple orders at the same price, the oldest order is selected.

- **Partial Execution** Either the incoming order or the matched limit order may be partially executed. When the matched limit order is partially executed, the order is left in the book with its quantity reduced. When the incoming order is partially executed, its quantity is reduced and a new round of matching begins. Participants receive one "order executed" message, sent via the exchange, per partial execution noting the fill price of each. When the incoming order is

executed in multiple parts, the average price per share is recorded as the last trade price for the symbol.

- **Order Acceptance** When the incoming limit order has remaining quantity after all possible matches have been executed, it will be added to the order book for later fulfillment, and an "order accepted" message will be sent via the exchange.

- **Order Cancellation** The order book locates the requested order by unique order id, removes any remaining unfilled quantity from the order book, and sends an "order cancelled" message via the exchange.

One might reasonably expect the order book in a market simulation to include a model for slippage. We assert that our platform produces realistic slippage naturally, without the need for such a model. Orders directed to the exchange suffer dynamic computation and network delays, during which time other orders are being executed.

2.3.5 The Trading Agent Class

The provided trading agent inherits from the basic agent class and represents the base class for a financial trading agent. It implements a number of additional features upon which subclassed strategy agents may rely:

- **Portfolio** The base trading agent maintains an equity portfolio including a cash position. It automatically updates this portfolio in response to "order executed" messages.

- **Open Orders** The trading agent keeps a list of unfilled orders that is automatically updated upon receipt of "order executed" and "order cancelled" messages, and when new orders are originated.

- **Last Known Symbol Info** The trading agent tracks known information about all symbols in its awareness, including the most recent trade prices, daily close prices (after the close), and order book spread or depth. These are automatically updated when receiving related messages.

- **Market Status** Upon initially waking at simulation start, the trading agent automatically locates an exchange agent, requests market open and close times, and schedules a second wakeup call for the time of market open. It also maintains and provides a simple "market closed" flag for the benefit of subclassing agents.

- **Mark to Market** The trading agent understands how to mark its portfolio to market at any time, using its most current knowledge of equity pricing. It automatically marks to market at the end of the day.

- **Messages** The trading agent knows how to originate all of the messages the exchange understands, and to usefully interpret and store all of the possible responses from the exchange.

- **Logging** The trading agent logs all significant activity: when it places orders; receives notification of order acceptance, execution, or cancellation; when its holdings change for any reason; or when it marks to market at the end of the day.

2.4 ABIDES Implementation

The ABIDES simulator is implemented using Python, currently 3.7, and the data analytical libraries NumPy [83], and Pandas [84]. It makes use of a virtual environment to provide platform independence and provides a straightforward deployment. It is seamlessly built to facilitate quick reconfiguration of varying agent populations, market conditions, exchange rules, and agent hyperparameters.

Basic execution of the simulation can be as simple as: `python abides.py -c config`, where `config` is the name of an experimental configuration file. Additional command line parameters are forwarded to the configuration code for processing, so each experimental configuration can add its own required parameters to a standard interface. Complex experimental configuration can be performed directly within the config file since it is simply Python code, however the inclusion of command line arguments is beneficial for coarse grain parallelization of multiple experiments of the same type, but with varied simulation parameters.

A typical configuration file will specify a historical date to simulate and a simulation start and stop time as nanosecond-precision timestamps. It will then initialize a population of agents for the experiment, configuring each as desired. For example, an experiment could involve 1,000 background agents (perhaps Zero Intelligence agents or Heuristic Belief Learning agents), 100 high-frequency trading agents, and one agent designed to create a market price impact by placing a very large order, with various initialization parameters to control their behavior. Each agent will at least be given a unique identifier and name. The configuration file will also construct a latency matrix (pairwise between all agents at nanosecond precision) and cubic network jitter model which will be applied to all inter-agent communications. If a "data oracle", a utility with access to a data source outside the simulation, is required for the experiment, the configuration file will initialize one. Finally a simulation kernel will be initialized and run, passing it the agent population, oracle, and other simulation parameters.

In its current form, ABIDES completes simulation of 1,000 typical "ping pong" agents that each send a single message to all other agents, and then respond to all incoming messages (for a total of approximately two million messages) in 3 minutes 18 seconds including all setup, overhead, and teardown, at a kernel processing rate of 10,230 events per second. Because the simulation is single-threaded, as many trials can be run simultaneously as available memory and processing cores permit with

relatively little performance degradation. For example, running two of the above ping pong experiments simultaneously on the same computer increases the total runtime by only four seconds. Similarly, a simulation of 1,000 Zero Intelligence (ZI) agents participating in a full day of trading at a NASDAQ-like exchange, with a mean market inter-arrival time of approximately one second, is completed in an average of 36 seconds total runtime. All simulation runtime data was collected on a notebook computer with a 2.4 GHz Intel Core i5 processor and 16 GB RAM.

Note that there is nothing finance-specific about the bootstrapper, configuration template, simulation kernel, or the basic agent class. All are appropriate for use in any discrete event simulation.

2.4.1 Momentum Trading Agent

To highlight the simplicity of creating a functional trading agent in our simulated environment, we present the code for a basic momentum trader. It wakes each minute during the day, queries the last trade price, projects a future price using linear regression over a configurable last N data points, and places a market order based on this projection. Following is the complete source, excluding import statements:

```
class MomentumAgent(TradingAgent):

  def __init__(self, id, name, symbol, startingCash,
               lookback):
    super().__init__(id, name, startingCash)

    self.symbol = symbol
    self.lookback = lookback
    self.state = "AWAITING_WAKEUP"

    self.trades = []


  def wakeup (self, currentTime):
    can_trade = super().wakeup(currentTime)

    if not can_trade: return

    self.getLastTrade(self.symbol)
    self.state = "AWAITING_LAST_TRADE"


  def receiveMessage (self, currentTime, msg):
    super().receiveMessage(currentTime, msg)

    if self.state == "AWAITING_LAST_TRADE" and \
         msg.type == "QUERY_LAST_TRADE":

      last = self.last_trade[self.symbol]
      self.trades = (self.trades + [last])[:self.lookback]

      if len(self.trades) >= self.lookback:
        m, b = np.polyfit(range(len(self.trades)),
                          self.trades, 1)
        pred = self.lookback * m + b

        holdings = self.getHoldings(self.symbol)

        if pred > last:
          self.placeLimitOrder(self.symbol, 100-holdings,
                               True, self.MKT_BUY)
        else:
          self.placeLimitOrder(self.symbol, 100+holdings,
                               False, self.MKT_SELL)

      self.setWakeup(currentTime + pd.Timedelta("1m"))
      self.state = "AWAITING_WAKEUP"
```

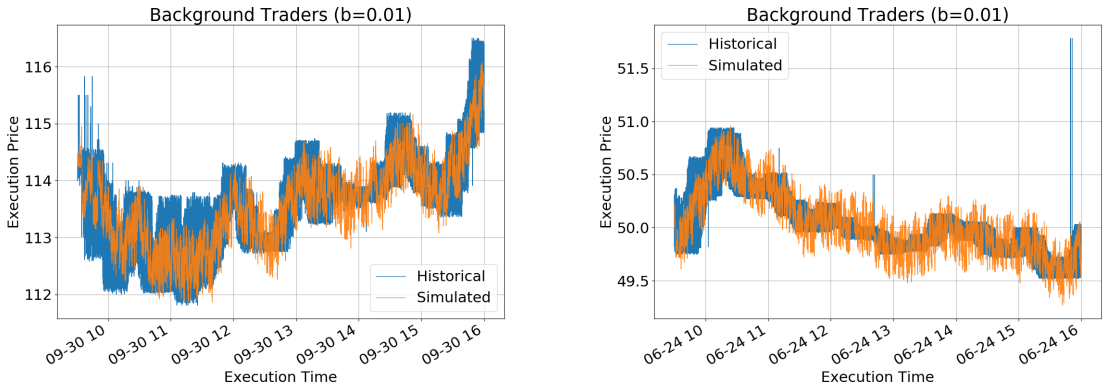
2.4.2 Noisy Background Agents

One long-term goal is to produce realistic but possibly noisy re-simulations of particular days in history to play out various "what if" scenarios. The idea is to populate the simulation with a large number of trading agents that provide a realistic environment into which experimental agents can be injected.

Our initial effort towards this goal involves the introduction of a data oracle with access to fine-resolution historical trade information, and the creation of a set of "background" agents which are able to request a noisy observation of the most recent historical trade as of the agent's current simulated time. The approach is meant to reproduce the behavior of a trader whose beliefs regarding the fundamental value of a stock are informed by interpretations of news and other incoming information. It was inspired by the concept of a stock's "fundamental value" as used in the work of Wang and Wellman. [77] Our approach is similar, but it uses historical data as a baseline rather than a mean-reverting stochastic process.

As background agents, we have implemented two common baseline agents from the continuous double auction literature. The Zero Intelligence (ZI) trader [54] submits random bids and offers to the market, usually drawn from some stochastic distribution around a central value belief for the underlying instrument. The Heuristic Belief Learning (HBL) agent [85] maintains a Bayesian belief distribution for likelihood of successful order transaction by offered price, and uses this to place orders which maximize expected surplus. HBL is based on the earlier GD agent [56], named for its authors Gjerstad and Dickhaut. We implement HBL as described by Wang and Wellman. [77] Each background agent trades only a single symbol on a single exchange.

Figure 2.4 compares the behavior of 100 background agents interacting in ABIDES with the actual intra-day price on two separate days in history. Ideally, we will see a price history that closely resembles the day in history, with similar statistical



(a) IBM: September 30, 2008 (b) MSFT: June 24, 2016

Figure 2.4: Simulated trades versus historical trades on two days.

properties.

2.5 Study: Market Impact

One area in which we believe simulation can add significant value to the current state of knowledge in finance is more accurate models of the market impact of large trades. Each order placed at the exchange potentially "moves the market" due to the nature of the market microstructure within the order book: arriving orders can add liquidity at a better price, altering the spread; or can match existing orders and remove liquidity from the market. See Figure 2.5 for an example of mechanical market impact.

Models that rely on historical data encounter limitations stemming from the inability to repeat history while introducing an experimental change and allowing subsequent events to be *altered* by that change. Models can attempt to compare "similar" days in history, but no two market days are ever the same.

If one could instead create a multi-agent simulation of a particular date in history such that a near approximation of historical trades emerged in the absence of any significant change, but the trading agents would realistically react to any such changes, a more accurate understanding of large trade impact could be attained. Here we

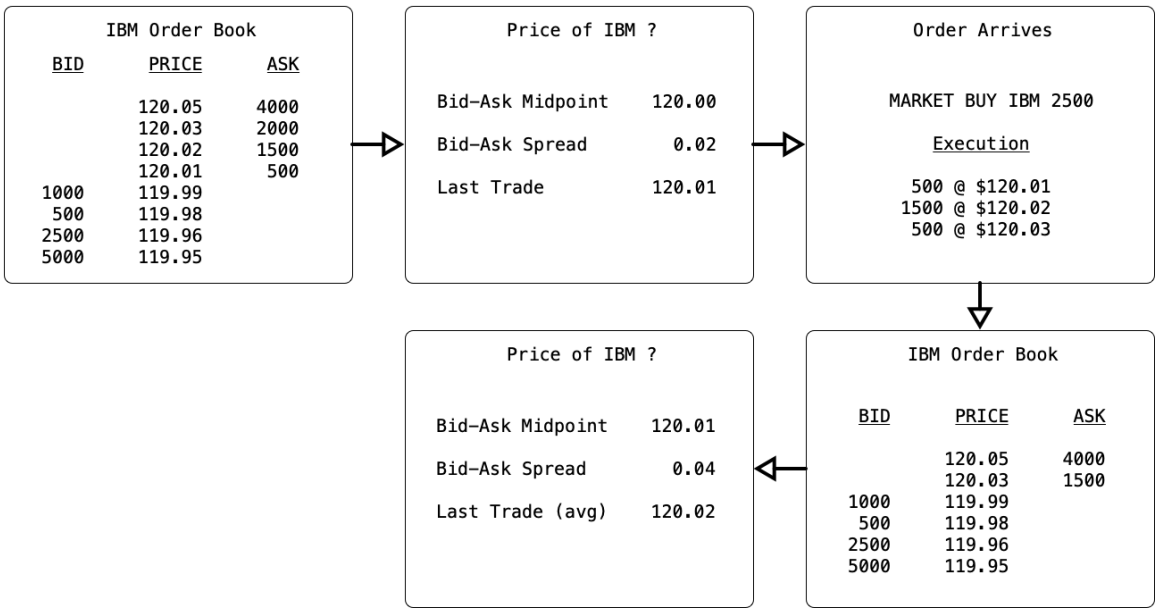


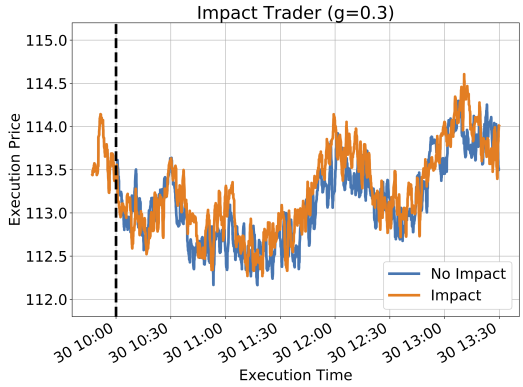
Figure 2.5: Example of mechanical market impact.

present a preliminary investigation of this idea.

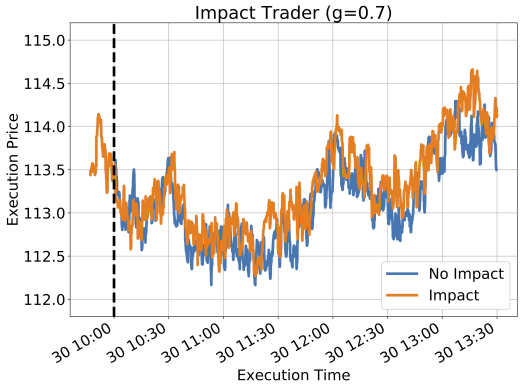
We begin each simulation with a population of background agents and at least one exchange agent. For this experiment, we add a single experimental impact agent, which simply places a single large market order at a predetermined time of day. The experimental parameter for the agent is its "greed"; that is, the proportion of available order book liquidity near the spread it consumes at the time of trade. For example, a long impact agent with $greed = 0.1$ will place a market buy order for 10% of the shares on offer.

Our experiment includes 100 background agents and one exchange agent handling an order book for a set of symbols including IBM. In Figure 2.6, the blue line represents each trade made by our population of background agents in the absence of an impact trader. The orange line shows each trade made by the simulated trading agents given the introduction of a single impact agent with varying "greed", acting one time with one trade at 10:00 AM on September 30, 2008. Both series are smoothed to improve visibility of the differences.

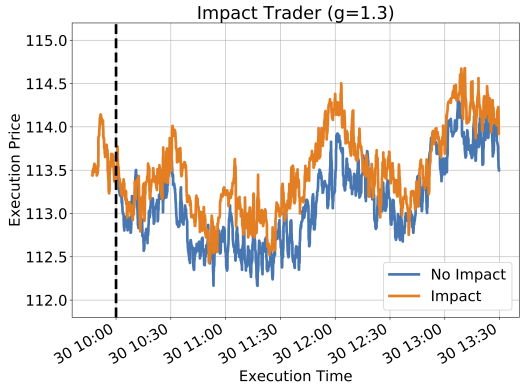
The impact trader has a clear effect on the market, despite the background agents'



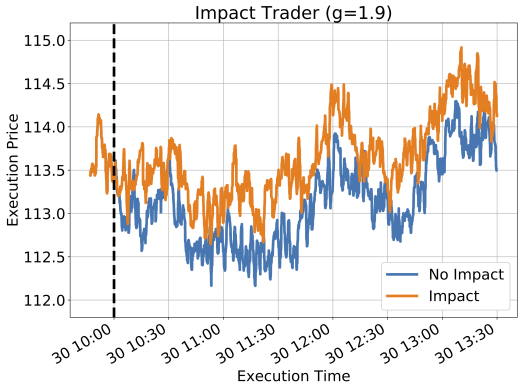
(a) MARKET BUY 1232 IBM



(b) MARKET BUY 2874 IBM



(c) MARKET BUY 5338 IBM

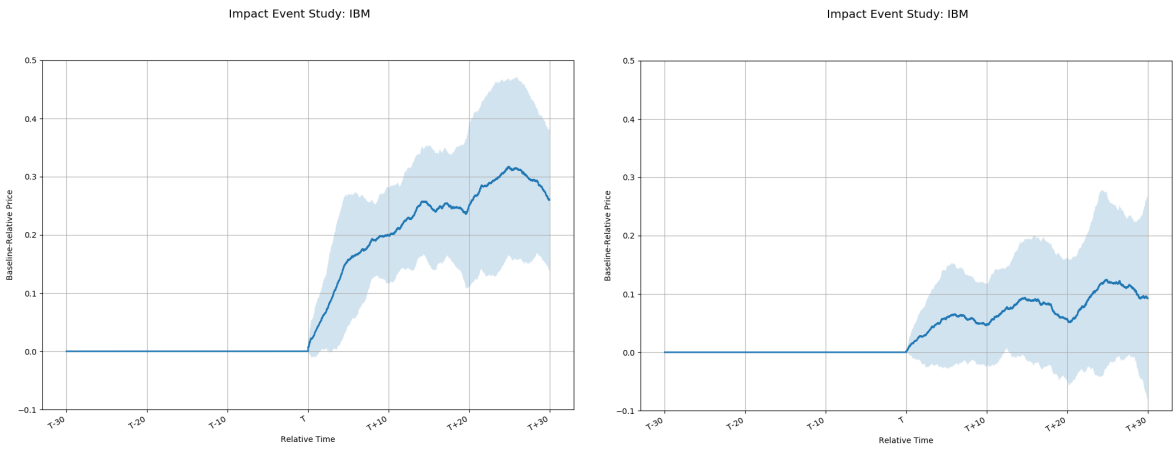


(d) MARKET BUY 7801 IBM

Figure 2.6: Market impact of trades on the same date at 10:00 AM.

central tendency to arbitrage the price toward historical levels, and the impact grows larger proportionally with its market bid size. The change is particularly noticeable in the cyclical peaks of the auction. Due to the price elevation it caused, the impact trader's total profit increased with the size of its bid from an average of \$2,633 with $greed = 0.3$ to \$12,502 with $greed = 1.9$. However its profit per share declined from \$2.14 to \$1.60. We found a correlation between profit per share and trade size of $r = -0.31$ across sixty experimental trials.

It is useful to consider these market impacts in aggregate across multiple experimental examples. ABIDES makes it easy to produce study plots from logged simulation data. Figure 2.7 shows a time-aligned event study of many impact trades



(a) Impact agent with greed 0.5 (b) Impact agent with greed 0.1

Figure 2.7: Market impact event studies.

at different times, on different days, to illustrate the range of likely price effects after the time of impact.

2.6 Study: High-Frequency Trading

A major aim of ABIDES is to provide the ability to conduct experiments that require information, like trader identity and location, that is not available in public stock market data. We illustrate that capability in a second empirical study.

There is a pervasive assumption that low-latency access to an exchange is a key factor in the profitability of many high-frequency trading strategies. This belief is evidenced by the "arms race" undertaken by certain financial firms to co-locate with exchange servers. To the best of our knowledge, this assumption has not been empirically tested for a continuous double auction market with a single exchange similar to the New York Stock Exchange. In this study, we investigate the relationship between latency of access to order book information and profitability of trading agents exploiting that information, as well as the impact on similar agents.

Lawrence Harris described three types of stylized traders relevant to market microstructure trading: liquidity traders that make short-term predictions based on

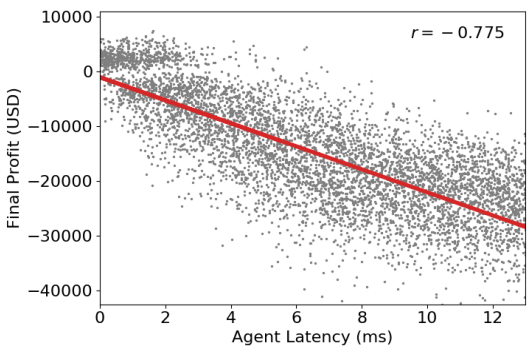
observed quotes, informed traders that place orders aggressively to monetize their knowledge advantage, and value-motivated traders that place less aggressive orders to be filled only if the price reaches their notion of a fair value. [86] Our study includes simulated versions of these stylized strategies: ZI agents with a minimum profit requirement as value-motivated traders, directional ZI agents that always bet on fundamental reversion to the mean as informed traders, and order book imbalance (OBI) agents as liquidity traders.

It is important to note that liquidity traders have no opinion about the "correct" value of a stock. They participate in anticipation of profit by observing the order flows in the market for clues that suggest short-term price movement arising from the market microstructure itself. In keeping with the spirit of liquidity trading, our OBI agents are unable to observe the exogenous price-time series (fundamental) used by the background traders. Instead, they track what proportion of total liquidity near the spread is on the buy side of the limit order book:

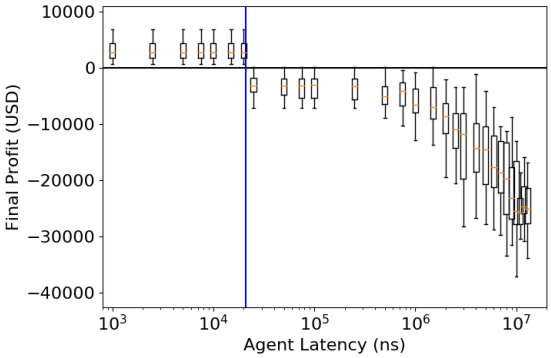
$$I = \frac{\sum_{b \in B} V_b}{\sum_{o \in O} V_o} \tag{2.1}$$

where B is the set of visible bid orders, O is the set of all visible orders, and V represents the share volume of a particular order. For example, when the indicator $I = 0.5$, liquidity is equally distributed between the two sides of the book, and when $I = 0.6$ there is 50% more bid than ask liquidity. The agent enters a directional trade when $I > 0.5 + H$ or $I < 0.5 - H$, where H is a configurable entry threshold, and exits the directional trade based on a trailing stop at configurable distance D applied to the same indicator (not the midpoint stock price). The order book depth (level) L at which to consider liquidity provision to be a positively-correlated signal is also a configurable parameter.

Our experimental HFT market is populated with 1,000 background traders, split



(a) Preliminary exploration: profit vs absolute latency of liquidity (OBI) traders. Note the anomaly at zero profit.



(b) Experiment 1: profit of experimental trader vs log scale latency. Blue line: control trader.

Figure 2.8: High-frequency trading empirical results: Preliminary exploration and Experiment 1.

evenly between the value-motivated and informed trader types described in Approach. For those agents requiring an exogenous price-time series, we employ a sparse mean-reverting fundamental as described in Byrd [87]. This mathematical series is a continuous form of the discrete mean reverting process described in Wah et al [57] with the addition of a second variance process which is applied at lower frequency but greater magnitude to represent infrequent "news shocks" that can change a trader's belief about the proper valuation of the stock.

As shown in Figure 2.8a, in a preliminary experiment of several hundred market days using the described background population and ten OBI traders having random latency and no particular control, we find a strong inverse Pearson correlation of $r = -0.775$ between the absolute latency of OBI traders and their profit at market close. This matches our intuition that lower latency should lead to higher profit for this strategy. However, there is an interesting gap in the plot at low latency values, with a cluster of positive profit agents and a cluster of negative profit agents, and virtually no agents with profit close to zero. A desire to understand this gap motivated the design of our two subsequent experiments.

2.6.1 Experiment 1: Absolute Latency with Control

The background agents are geographically situated around the United States relative to the location of the New York Stock Exchange. The minimum communication latency of each individual background agent is drawn from a uniform distribution of $21\mu s$ (roughly the other side of Manhattan) to $13ms$ (around Seattle, Washington). There are ten OBI liquidity traders. One, serving as the control, is always placed at the minimum latency permitted to background agents. Eight are randomly distributed in the same range as the background agents. The final liquidity trader is considered the experimental agent for Experiment 1: its geographic location (latency) is systematically varied to measure its impact on the returns of all liquidity traders. All OBI liquidity traders use an entry threshold of $H = 0.17$, a trailing stop distance of $D = 0.085$, and a book depth significance parameter of $L = 10$.

The experiment includes 600 market simulations, each representing a full market day from 9:30 AM to 4:00 PM at nanosecond resolution. Each of twenty random market days (i.e. different exogenous price-time fundamental series) are repeated with the experimental agent varied among thirty positions, ranging from exchange colocation (approx. $333ns$) to Seattle, Washington (approx. $13ms$). Using these 6,000 full-day observations, we show in Figure 2.8b a box-plot representation of the relationship between log-scale latency in nanoseconds and final profit for the single experimental liquidity trader at each tested location. The mean profit per tested latency is marked with a tick, the box extents mark one standard deviation, and the whiskers mark two standard deviations. Recall that a control liquidity trader (blue line) is always placed at the minimum permitted latency outside of co-location in an effort to explain the zero profit gap in the preliminary experiment.

Let X be the experimental liquidity trader with latency L_X and profit P_X, and C be the control liquidity trader with fixed latency fixed latency L_C and profit P_C. We now note two new observations. First, $\forall L_X < L_C : P_X \perp L_X$. That is, once the

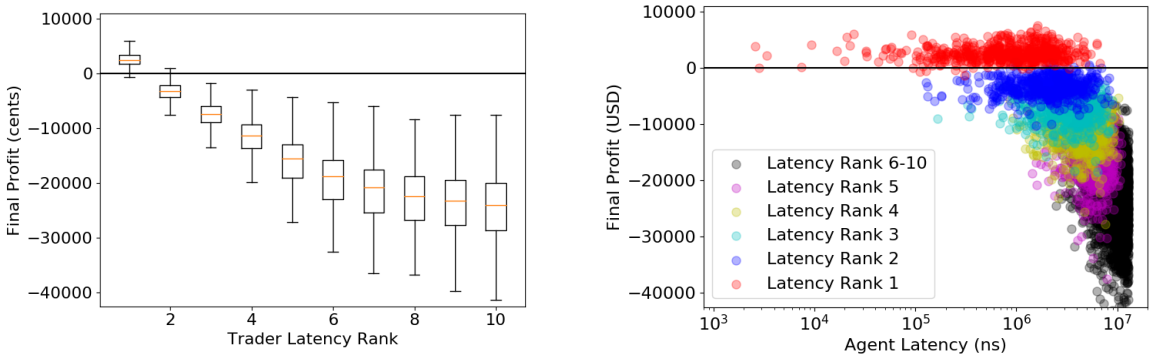
absolute latency of the experimental trader is lower than that of the control trader, its latency does not affect its profit. Second, as soon as $L_X > L_C$ there is an immediate transfer from P_X to P_C.

These results together suggest that latency *rank* among the competing liquidity agents is more significant to profit outcomes than absolute latency values. This could explain the gap in the preliminary plot: Depending on whether some other agent was even closer to the exchange, a given low latency trader might see very different results. However, we cannot support this claim based solely on Experiment 1, because the location of the *other* liquidity traders (non-experimental, non-control) on each market day is not considered.

2.6.2 Experiment 2: Latency Rank

The same agent population from Experiment 1 is randomly geolocated with latency ranging from exchange colocation ($333ns$) to Seattle, Washington ($13ms$), with the intent to examine latency rank independent of absolute latency. We again produce 6,000 observations of OBI latency and profit, this time for 600 different simulated market days. The experiment is otherwise similar to Experiment 1.

For this experiment, absolute latency and profit were recorded for all agents. Latency rank was also recorded among the experimental agents. For example, the liquidity trader closest to the exchange on a given day would be latency rank 1, the second closest rank 2, and so on. In Figure 2.9a, we present a box-and-whisker plot of aggregated profit by latency rank among the OBI liquidity agents. Over the course of a simulated trading day, the liquidity trader closest to the exchange receives mean and std marked-to-market profit of \$2,681.04 and \$1,389.37. The second closest liquidity trader receives mean and std profit of \$-3,297.30 and \$1,661.30, and the liquidity trader furthest from the exchange received mean and std profit of \$-24,473.64 and \$-6,449.97. This clearly supports the notion that latency rank is a key factor driving



(a) Profit of multiple trials with liquidity traders distributed randomly, identified by latency rank order within each trial.

(b) Profit of multiple trials with liquidity traders distributed randomly showing ranked and absolute latency.

Figure 2.9: High-frequency trading empirical results: Experiment 2. Randomly situated agents reported by latency rank, with lower rank traders closer to the exchange.

the allocation of profit within this trading strategy population.

A potential confounding factor arises from absolute latency. If the lowest ranked OBI agent were also always close to the exchange, absolute latency would present a competing explanation for the profit differential. In Figure 2.9b, we plot end of day profit against log-scale nanosecond latency, with the addition of a zero profit line. Each point represents one liquidity trader's result for one simulated market day, and each point is color-coded according to that trader's latency rank for that simulated market day, with red being rank 1, blue being rank 2, and so on. Ranks 6-10 are aggregated into a single black grouping. Because this plot shows both absolute latency and latency rank together, we can clearly see that the nearest liquidity trader (red) on a given market day does well even when situated very far from the exchange, and liquidity agents in ranks 2 and 3 perform poorly even when relatively near the exchange.

Taking Figures 2.9a and 2.9b together, we can summarize that: The liquidity trader closest to the exchange rarely loses money, and the second closest liquidity trader rarely makes money, regardless of their absolute distance from the exchange. Our initial concern regarding the preliminary experiment's zero-profit gap proved

correct. While there was a strong inverse correlation between absolute latency and profit, the excess profit of some liquidity traders is better explained by latency rank. In particular, the liquidity strategy is only consistently profitable for the single agent nearest the exchange. Thus it appears the HFT "arms race" among firms competing to be slightly closer to the exchange is indeed rational. It would not be possible to reach this conclusion using public market data due to the lack of trader identity, strategy, or location, but we can do so in the context of ABIDES.

2.7 Conclusion

In this chapter, I presented the design and implementation of ABIDES, a high-fidelity equity market simulator that provides an environment within which complex research questions regarding trading agents and market behavior can be investigated. I demonstrated how previous intra-day transaction histories are noisily reproduced by a population of interacting background trading agents communicating with an exchange agent. These background agents are designed to provide a realistic market environment into which experimental agents can be injected. I also explored ABIDES' potential through two small empirical studies: The first illustrates how large market orders create a cascading impact that affects prices over a significant period of time; the second validates the intuition that high-frequency traders nearer to the exchange will take profit from similar traders even slightly more distant.

ABIDES makes complementary contributions to existing simulations by enabling experimental focus on the "market physics" of the real world including:

- Support for continuous double-auction trading at the same nanosecond time resolution as real markets such as NASDAQ;
- Ability to simulate specific dates in market history with gated access to historical data;

- Variable electronic network latency, a realistic cubic network jitter model, and agent computation delays;
- Requirement that all agents intercommunicate solely by means of standardized message protocols;
- Easy implementation of complex agents through a full-featured hierarchy of base agent classes.

These features enable an expanded range of experimental studies. I believe ABIDES is also the first full-featured, modern market simulator to be shared with the community as an open source project. With its robust and realistic simulation environment, I hope that ABIDES will allow responsible practitioners to better anticipate the potential consequences of deploying complex, intelligent trading agents.

In the next chapter, I will further extend ABIDES' capabilities to a new domain: the simulation of a geographically-distributed collection of agents collaborating under a privacy-preserving federated learning protocol.

CHAPTER 3
DIFFERENTIALLY PRIVATE SECURE MULTI-PARTY COMPUTATION FOR FEDERATED LEARNING IN FINANCIAL APPLICATIONS

In this chapter, we adapt the open-source ABIDES simulation framework presented in Chapter 2 to the domain of *federated learning*, an approach to distributed computing that enables a population of clients, working with a trusted server, to collaboratively learn a shared machine learning model while keeping each client's data within its own local systems. A high-level introduction to federated learning can be found Section 1.2.1. Here, we first present a more detailed overview of federated learning, differential privacy, and secure multi-party computation suitable for the informed generalist, and then a more mathematical view of the underlying techniques. We discuss our approach to privacy-preserving federated learning (PPFL) in detail and present timing, accuracy, and privacy results for a set of extensive experiments in simulation.

3.1 Overview

Modern financial firms routinely need to conduct analysis of large data sets stored across multiple servers or devices. A typical response is to combine those data sets into a single central database, but this approach introduces a number of privacy challenges: The institution may not have appropriate authority or permission to transfer locally stored information, the owner of the data may not *want* it shared, and centralization of the data may worsen the potential consequences of a data breach.

For example, the mobile app ai.type collected personal data from its users' phones and uploaded this information to a central database. Security researchers gained

access to the database and obtained the names, email addresses, passwords, and other sensitive information of 31 million users of the Android version of the app. Such incidents highlight the risks and challenges associated with centralized data solutions. [10]

In this section, we motivate our approach while providing an extensive non-technical overview of the underlying techniques.

3.1.1 Federated Learning

One approach to mitigate the mentioned privacy concerns is to analyze the multiple data sets separately and share only the resulting insights from each analysis. This approach is realized in a recently-introduced technique called federated analysis. [16] Federated *learning*, already adopted by large companies like Google, allows users to share insights (perhaps the parameters of a trained model) from the data on their laptops or mobile devices *without* ever sharing the data itself, typically as follows:

1. Users train a local model on their individual data.
2. Each user sends their model weights to a trusted server.
3. The server computes an average-weight shared model.
4. The shared model is returned to all of the users.
5. Users retrain a local model starting from the shared model.

For instance, email providers could use federated learning to reduce the amount of spam their customers receive. Instead of each provider using its own spam filter trained from its customers' reported spam email, the providers could combine their models to create a shared spam-detection mechanism, without sharing their individual customers' reported spam emails. For a survey of recent advances in federated learning, see Kairouz et al. [19]

It is still possible, however, for a malicious party to potentially compromise the privacy of the individual users by inferring details of a training data set from the trained model's weights or parameters [13, 12]. It is important to protect sensitive user information while still providing highly accurate inferences.

3.1.2 Differential Privacy

Simply anonymizing data is no longer enough to guarantee the privacy of individuals whose information has been collected, due to the increasing prevalence of database reconstruction attacks and re-identification from correlated data sets.

Differential privacy [14, 88] can help prevent such reverse engineering by adding noise to the input data set, to intermediate calculations, or to the outputs. For example, in Step 2 of the Federated Learning process above, each client can add randomly-generated values to its model weights before transmission. Then, even if the data is reverse engineered, it is *not* the exact data of any user.

More formally, differential privacy is a mathematical concept that guarantees statistical indistinguishability for individual inputs by perturbing values. The use of differentially-private machine learning algorithms in centralized settings is widely discussed in the literature, and the technique has been adopted by major companies; for example, Apple uses it in web search auto-completion. The application of differential privacy adds a layer of randomness so that adversaries with additional information still have uncertainty over the original value. There is an obvious trade-off: adding randomness to the collected data preserves user privacy at the cost of accuracy. Proper application of differential privacy ensures that meaningful insights can still be derived from the aggregated data.

3.1.3 Secure Multi-Party Computation

Achieving a desired level of differential privacy can require adding a great deal of accuracy-reducing noise into the mix. An alternative method which guarantees privacy *without* compromising accuracy is secure multi-party computation (MPC) [89]. Using MPC, multiple parties collaborate to compute a common function of interest without revealing their private inputs to other parties. An MPC protocol is considered *secure* if the parties learn only the final result, and no other information.

For example, a group of employees might want to compute their average salary without any employee revealing their individual salary to any other employee. This task can be completed using MPC, such that the only information revealed is the result of the computation (i.e. the average salary). If each pair of employees holds a large, arbitrary, shared number, such that one employee will add it to their salary and the other will subtract it, then the result of the computation will not change, but no one will know any employee's real salary.

The same idea can be applied to federated learning by having the parties use a secure weighted average protocol, under which each client encrypts their model weights, but the server can still calculate the weighted average on the encrypted data.

3.1.4 Secure Federated Learning

Considering all of the above, we arrive at the idea of secure federated learning, in which clients encrypt the model weights sent in Step 2 of Federated Learning. Assuming the encryption scheme is chosen appropriately, the server will still be able to perform the necessary calculation on the encrypted data, but will not be able to discover the original weights for any user. A recent line of investigation has constructed secure federated learning using techniques from MPC. [16, 90]

MPC protects the computation inputs from exposure to the server, but the exact

final result is revealed to all parties by design. Unfortunately, for some types of computation, the final result can be used to reveal information about the inputs. For example, in the case of employees computing their average salary, once the result is known, if all but one of the employees work together, they can easily determine the salary of the final employee given the output (average salary). A secure learning approach based only on MPC may not be ideal for these cases.

By applying differential privacy on top of MPC, we can construct a federated learning system that protects from even this type of extreme collusion attack. If each client adds noise to its model weights before sending, the final calculation will still be accurate within known bounds, but we will eliminate the possible leakage of any inputs from the output. In a solution which used *only* differential privacy, the server would know the "noisy" private weights of each user. In the solution which combines MPC and differential privacy, the noisy weights sent to the server are also encrypted such that the server can calculate the result, but cannot infer anything about even the noisy weights of any particular user. The system is thus now fully private.

3.1.5 Differentially Private Secure Multi-Party Computation for Federated Learning

A protocol such as the one presented here, which combines federated learning, differential privacy, and secure multi-party computation, should be of particular interest in the finance space. These firms operate under substantial regulation with respect to the use, protection, and disclosure of client information. Data sharing, even within a firm, is thus often difficult to achieve, with negative impacts in the ability to harness new techniques in artificial intelligence (AI) to improve key performance indicators at the firm, such as accurate estimation of loan failure rates, reduction of financial market transaction costs, or optimization of product pricing.

This combination approach can improve internal data protections while still enabling the application of powerful AI to the company's data. Now each client, server,

or device's data can be kept securely in its originating silo, where local model training can safely occur, and the trained models can be shared and combined in an encrypted and differentially private manner. The data silos can thus each contribute to the overall organization learning an accurate, useful, and directly applicable model without increasing the exposure risk of any client's data. And while no firm wants to give away a competitive advantage, the protocol can also improve models through secure inter-firm collaboration to lower market execution costs or more accurately price the risk component of a loan product, benefiting all participants.

We demonstrate our approach to differentially private secure multi-party aggregation for federated learning by application to a well-known credit card fraud data set [91], and show that client populations of varying size can collaboratively build a fraud detection model without sharing or revealing their local data. We note that Jayaraman et al [90] approached the same problem with heavier cryptography tools, on a data set that is not finance related, and without an end-to-end simulation of the protocol.

The key contributions of this paper are to introduce the finance community to recent advances in secure federated learning, to provide a complete open-source platform on which such protocols can be developed, and to demonstrate a protocol that enables secure learning of a shared fraud detection model in at most 30 protocol iterations on an extremely class-imbalanced real world data set.

3.2 Background and Related Work

In this section, we provide a formal description of the important components underlying our approach.

3.2.1 Secure Multiparty Computation

Consider n parties $P_1, \ldots, P_n$ that hold private inputs $x_1, \ldots, x_n$ and wish to compute some arbitrary function $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, where the output of P_i is y_i. Secure Multi-Party Computation (MPC) enables the parties to compute the function using an interactive protocol, where each party P_i learns exactly y_i, and nothing else. Seminal results established in the 1980s [89] show that *any* computation can be emulated by a secure protocol.

It is important that the security of the protocol be preserved even in the presence of adversarial behavior. For example, several leading banks might collaborate to learn an improved model to minimize the transaction costs associated with fulfilling client orders in a financial market. The privacy of each honest bank's individual client orders should be preserved even if other banks collude by pooling their information, revealing their encryption offsets, or deviating from the specified protocol.

In this work, we focus on a semi-honest adversary who follows the protocol specification, but may attempt to learn honest parties' private information from the messages it receives, or to collude with other parties to learn private information. This is the same level of security contemplated in prior works in our setting.

3.2.2 Differential Privacy

Differential privacy states that if there are two databases that differ by only one element, they are statistically indistinguishable from each other. In particular, if an observer cannot tell whether the element is in the dataset or not, she will not be able to determine anything else about the element either.

Definition 1. (ϵ-differential privacy [92]) For any two neighboring datasets $\mathcal{D}_1 \sim \mathcal{D}_2$ that differ by one element, a randomized mechanism $\mathcal{A}$: $\mathcal{D} \rightarrow \mathcal{O}$ preserves ϵ-differential privacy (ϵ-DP) when there exists $\epsilon > 0$ such that,

$$\Pr\left[\mathcal{A}(D_1) \in \mathcal{T}\right] \leq e^{\epsilon} \Pr\left[\mathcal{A}(D_2) \in \mathcal{T}\right] \tag{3.1}$$

holds for every subset $\mathcal{T} \subseteq \mathcal{O}$, where $\mathcal{D}$ is a dataset, $\mathcal{T}$ is the response set, and $\mathcal{O}$ depicts the set of all outcomes.

The value ϵ is used to determine how strict the privacy is. A smaller ϵ gives better privacy but worse accuracy. Depending on the application ϵ should be chosen to strike a balance between accuracy and privacy.

Definition 2. (Global Sensitivity [92]) For a real-valued query function $q : \mathcal{D} \rightarrow \mathbb{R}$, where $\mathcal{D}$ denotes the set of all possible datasets, the global sensitivity of q, denoted by Δ, is defined as

$$\Delta = \max_{\mathcal{D}_1 \sim \mathcal{D}_2} |q(\mathcal{D}_1) - q(\mathcal{D}_2)|, \tag{3.2}$$

for all $\mathcal{D}_1 \in \mathcal{D}$ and $\mathcal{D}_2 \in \mathcal{D}$.

The sensitivity is defined as the maximum effect of any single input of the function on the output, and should be concealed to preserve privacy.

Laplacian Mechanism

One of the most well-known techniques in differential privacy is the Laplacian mechanism, which uses random noise X drawn from the symmetric Laplacian distribution. The zero-mean Laplacian distribution has a symmetric probability density function $f(x)$ with a scale parameter λ defined as:

$$f(x) = \frac{1}{2\lambda} e^{-\frac{|x|}{\lambda}}. \tag{3.3}$$

Given the global sensitivity, Δ, of the query function q, and the privacy parameter ϵ, the *Laplacian mechanism* $\mathcal{A}$ uses random noise X drawn from the Laplacian distribution with scale $\lambda = \frac{\Delta}{\epsilon}$. The Laplacian mechanism preserves ϵ-differential privacy [14].

3.2.3 Training Local Logistic Regression Classifiers

Logistic regression is a machine learning algorithm used to solve the problem of binary linear classification.

Assume one of n parties is called P_i and has a local data set consisting of instances $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_m^{(i)})$, where m is the number of features, and their corresponding labels $y^{(i)}$.

Party P_i uses its training examples $(x^{(i)}, y^{(i)})$ to learn a logistic classifier with weights w_i. The weights are obtained by solving the following optimization problem:

$$w_i = \arg\min_w \frac{1}{t_i} \sum_{k=1}^{t_i} log(1 + e^{-y_k^{(i)} f(x_k^{(i)})}), \tag{3.4}$$

where $f(x_k^{(i)}) = w^T x_k^{(i)}$ and t_i is the number of training examples of P_i.

In order to minimize the loss function, we make use of gradient descent, an iterative optimization algorithm, calculating the optimal w iteratively as $w^{j+1} \leftarrow w^j - \alpha \nabla L(w^j)$, where α is the learning rate, j is the iteration, $w^0 = 0$, and ∇L is the gradient of the loss function. Our local logistic regression is a vector-based re-implementation of Jayaraman et al. [90]

3.2.4 Differentially Private Federated Logistic Regression using Output Perturbation by adding Laplace noise

Privacy-preserving federated learning allows a large number of parties to learn a model while keeping their local training data private. Parties first train local models

on their local data and coordinate with a server to obtain a global model. Given n parties, let w_i, for $i \in 1$ to n, represent the local model estimator after minimizing the objective function.

Then $W = \frac{1}{n} \sum_{i=1}^{n} w_i + \eta$, where η is the differentially private noise added to the cumulative model.

According to Jayaraman et al [90], for 1-*Lipschitz* the global sensitivity for a multi-party setting is $\frac{2}{n*k*\alpha}$, where k is the size of the smallest dataset amongst the n parties, and α is the regularization parameter. Hence, $\eta = Laplace(\frac{2}{n*k*\alpha*\epsilon})$, where ϵ is the privacy loss parameter.

In our protocol, each client will add noise to the weights of the trained local model.

3.3 Approach

We illustrate the application of federated learning with differential privacy and secure multi-party computation to a problem of collective interest in finance, that of accurately identifying fraudulent credit card transactions. This application typifies the case where multiple firms would individually and collectively profit from working together to eliminate the common problem of fraudulent purchases, as the occurrence of fraud benefits none of the lawful parties in the processing chain.

The current limitation to this type of cooperation is data sharing. The involved companies would not wish to share their local training data, that is their entire history of fraudulent and non-fraudulent transactions, including potentially sensitive customer and merchant information, and in many cases would be legally prohibited from doing so. A secure federated learning protocol could satisfy the firms and their regulators that data exposure risks have been sufficiently minimized to permit this mutually beneficial collaboration. Here we describe the key aspects of our approach.

As described in the Overview, federated learning is an iterative algorithm that follows a simple, repetitive process. The server chooses some users to produce an

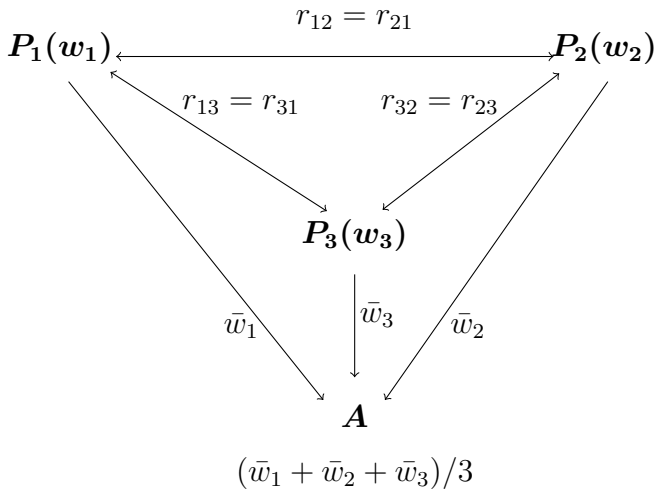


Figure 3.1: Secure 3-party weighted average protocol where $\bar{w}_1 = w_1 + r_{12} + r_{13}, \bar{w}_2 = w_2 - r_{21} + r_{23}, \bar{w}_3 = w_3 - r_{31} - r_{32}$.

updated model. Those users train a model on their individual data, then send the model updates to the server. The server aggregates the updates to construct a new global model and shares it with all users.

In this paper, we consider logistic regression as the local learning method, and each client update includes the weights of that logistic regression. The server receives the weights from all clients at each iteration and computes the new global model using the average of the client updates for each weight. Recall also from the Overview the literature demonstrating that the server can infer some private client data from the trained model weights, which is clearly undesirable.

3.3.1 Eliminating weight leakage

In order to hide each client's model weights from the server, we use the technique of secure multi-party computation (MPC), in which the clients work together to send their individual updates to the server in an encrypted manner. In particular, our secure weighted average protocol running across n clients is based on the protocol of Bonawitz et al [16].

Informally, this can be thought of as each pair of clients sharing a common source

of randomness known only to that pair. When communicating weights to the server, one client within each pair will add, and the other will subtract, the (same) next value in the shared randomness. In this way, the averaged model produced by the server will be identical to a model produced without MPC, but each weight arriving from each client has $n-1$ large random numbers added to or subtracted from it, removing the server's ability to accurately reconstruct any client's actual model weights. We describe the process formally in Section 3.3.3.

Our protocol includes the ability to reuse the common randomness for each iteration of logistic regression, so the clients will only require pairwise communication once via the server, at the start of the protocol. In subsequent iterations, each client only has to communicate with the server. A 3-party example is given in Figure 3.1. Note that the weights are encrypted via the use of the common randomness r. The values $\bar{w}$ reveal nothing about the weights w.

3.3.2 Eliminating weighted average leakage

Using our protocol, all information about every client's weights is completely hidden from the server. However, the shared learned model can still expose some information about individual client weights and subsequently a client's local data set.

Imagine a scenario in which $n-1$ out of the n clients collude or the server chooses $n-1$ adversarial parties. The shared computation result is simply the per-weight average of each client's encrypted weights. The $n-1$ colluding clients of course know their correct individual weights, but they also know *all* of the encryption added to the weights, because each component of the non-colluding client's encryption occurred with a colluding counterparty who knows the values added or subtracted. The colluding clients, working together, can thus recover the *exact* model weights of the "honest" client.

We would like to avoid client data exposure even in the face of such collusion. To

Protocol 1 Privacy-Preserving Federated Logistic Regression Protocol Π_{PPFL}

The protocol Π_{PPFL} runs with parties $P_1, \ldots, P_n$ and a server S. It proceeds as follows:
Inputs: For $i \in [n]$, party P_i holds input dataset D_i.
Public Parameters: $(\mathbb{G}, g, q)$ generated by $\mathcal{G}(1^\lambda)$ and modulo p.
$\boldsymbol{\Pi_{\mathsf{PPFL}}.\mathsf{Setup}(1^\lambda)}$:

Round 1: Each party P_i for $i \in [n]$ proceeds as follows:

- Choose n secrets $a_{i,1}, \ldots, a_{i,n}$ uniformly and independently at random from $\mathbb{Z}_q$ and compute $(pk_{i,1}, \ldots, pk_{i,n}) = (g^{a_{i,1}} \bmod p, \ldots, g^{a_{i,n}} \bmod p)$.
- Generate a Laplace random variable η_i from $Laplace(\frac{2}{n*len(DS_i)*\alpha*\epsilon})$.
- Each party P_i sends $pk_{i,j}$ to the Server who forwards $pk_{i,j}$ to party P_j.

Round 2: Each party P_j for $j \in [n]$ proceeds as follows:

- Upon receiving all values $(pk_{1,j}, \ldots, pk_{n,j})$, compute the shared common keys $r_{i,j}$ for all $i \in [n]$ as follows:
  (a) Using the secret $a_{j,i}$ compute $c_{i,j} = c_{j,i} = (pk_{i,j})^{a_{j,i}} = (g^{a_{i,j}})^{a_{j,i}} \bmod p$.
  (b) Let $c_{1,j}, \ldots, c_{n,j}$ be the set of all common keys. Use a key-derivation function and set $r_{i,j} = r_{j,i} = H(c_{i,j})$

Given the above setup, we can compute the federated logistic regression model:
$\boldsymbol{\Pi_{\mathsf{PPFL}}.\mathsf{WeightedAverage}(D_i, \{r_{i,j}\}_{j\in[n]})}$:

Round 1: Each party P_i proceeds as follows:

- Compute the weights W_i, using Equation (1), of the local logistic classifier obtained by implementing regularized logistic regression on input D_i.
  The next steps are repeated per weight. Without loss of generality we describe the algorithm for a single weight, denoted by w_i.
- Compute and send y_i to the server.

$$y_i := w_i + \sum_{j=i+1}^{n} r_{i,j} - \sum_{k=1}^{i-1} r_{k,i} + \eta_i \bmod p \ .$$

Round 2: The server sends $W = \Big(\sum_{i=1}^{n} y_i \bmod p\Big)/n$ to all parties.

$\boldsymbol{\Pi_{\mathsf{PPFL}}.Output(1^\lambda, W)}$: Each party P_i upon receiving W runs the next iteration of the logistic regression repeating $\Pi_{\mathsf{PPFL}}.WeightedAverage(1^\lambda)$.

this end, we can apply differential privacy within the MPC protocol. In addition to the pairwise encryption, each client will also independently generate and add random "noise" to each of its model weights. Now, even if $n-1$ clients collude, they will only be able to recover the differentially private "noisy" weights of the honest client, instead of the exact weights. The privacy loss parameter in differential privacy is typically called ϵ (epsilon) and is inversely proportional to the amount of noise added. Thus there is a trade-off: lower values of epsilon (more "noise") better prevent inference of private data during a collusion attack, but eventually interfere with accurate learning of the shared model.

3.3.3 Secure Weighted Average Protocol

In this section we formally describe our weighted average protocol Π_{PPFL}, depicted in Section 3.3.1 Protocol 1, for secure logistic regression performed by a set of clients $(P_1, \ldots, P_n)$ and a server S. All operations are performed modulo some bound p.

During setup, every pair of parties P_i and P_j will share some common randomness $r_{i,j} = r_{j,i}$. In the online weighted average phase, client P_i sends its weights masked with these common random strings, adding all $r_{i,j}$ for $j > i$ and subtracting all $r_{i,k}$ for $k < i$. That is, P_i sends to server S the following message for its data x_i:

$$\bar{w_i} := (w_i + \sum_{j=i+1}^{n} r_{ij} - \sum_{k=1}^{i-1} r_{ki}) \bmod p \tag{3.5}$$

To establish common randomness, each pair of client parties run the Diffie-Hellman Key agreement protocol [93] communicating via the server. The cryptographic primitives used in Protocol 1 include:

- An algorithm $\mathcal{G}(1^\lambda)$, where λ is the security parameter, that outputs a representation of a cyclic group $\mathbb{G}$ of order q (with $||q|| = \lambda$) for which the discrete logarithm problem is believed to be hard. Recall that a group $\mathbb{G}$ is cyclic if there

exists a generator g such that $\{g^0, g^1, \ldots, g^{q-1}\} = \mathbb{G}$. Moreover, the discrete logarithm problem is believed to be hard if for every probabilistic polynomial time adversary A, there exists a negligible function $\mathsf{negl}(\cdot)$ such that:

$$\Pr_{x \leftarrow \mathbb{Z}_q}[A(\mathbb{G}, g, q, g^x) = x] = \mathsf{negl}(\lambda)$$

In other words, it is hard to guess x given g^x for particular groups $\mathbb{G}$.

- A key derivation function $H : \mathbb{G} \to \{0,1\}^\lambda$. It is assumed that if h is distributed uniformly in $\mathbb{G}$, then $H(h)$ is distributed uniformly in $\{0,1\}^\lambda$.
- A pseudorandom generator with double expansion, i.e., $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$. It is assumed that for every distinguisher D there exists a negligible function $\mathsf{negl}(\cdot)$ such that:

$$\left| \Pr_{s \leftarrow \{0,1\}^\lambda}[D(G(s)) = 1] - \Pr_{r \leftarrow \{0,1\}^{2\lambda}}[D(r) = 1] \right| = \mathsf{negl}(\lambda)$$

Protocol 1 is described for a single iteration of the logistic regression. To perform the next iteration the algorithm $\Pi_{\mathsf{PPFL}}.\mathsf{Setup}$ is not repeated. Instead, the parties can use the common keys $r_{i,j}$ to generate different common keys for the next iteration. More specifically, in the first iteration of the logistic regression we use a pseudorandom generator $(r'_{i,j}, s) = G(r_{i,j})$ and update the common randomness $r_{i,j} := r'_{i,j}$. For the next iteration, we run $G(s)$ to obtain a new $r'_{i,j}$ and the seed for the next iteration and so on. Thus the parties need to run the exchange once at the onset of the training. Our secure weighted average protocol is based on the protocol of Bonawitz et al [16] and its security follows in the same way.

3.4 Experiments

In order to evaluate our method, we implement it in ABIDES, the agent-based interactive discrete event simulation framework described in Chapter 2. Many prior works on federated learning calculate the running time of their protocol ignoring the communication time to the server, but with ABIDES we are able to simulate the latency of the distributed clients' communication.

3.4.1 Experimental Dataset and Method

To evaluate the performance of the protocol on real-world data, we selected the Kaggle Credit Card Fraud (2013) dataset [91], which provides transformed features that represent the first 26 principal components of unknown original features. Two original features are provided without transformation: the elapsed time from the start time of the dataset and the amount of the transaction. We used the Amount column without transformation, but excluded the Time column because our learning method does not attempt to identify temporal clusters or patterns. We also added a constant intercept feature to permit greater flexibility in the regression. The dataset provides a categorical y variable identifying whether the transaction was judged fraudulent (True) or not (False). Of the 284,807 records, only 492 (less than 0.2%) are labelled fraudulent, representing an extremely unbalanced dataset.

We loaded the dataset once at the start of each complete simulation of the protocol and performed a randomized train-test split (75% vs 25%). At each protocol iteration, each client selected 1000 rows of training data at random as its "local" data for that iteration. The holdout test data was the same for all clients, and no client was ever permitted to train on it. The clients then implemented Protocol 1 as described in the Approach section, attempting to collaboratively learn a credit card fraud detector, despite each individual client having insufficient data (possibly even zero fraudulent

Table 3.1: Total protocol time, mean server time per iteration, and mean time per user per protocol iteration in milliseconds for 30 protocol iterations, within which each client runs 250 iterations of local regression training.

Users	Total	Server	User DH Setup	User Training	User Encrypt
100	4148.9	16.897	6.721	86.152	2.231
200	6371.7	33.573	13.388	85.836	4.429
300	9575.1	50.893	20.156	85.469	6.558
400	12931.0	67.682	27.018	86.207	8.745
500	17008.6	85.432	33.944	86.305	10.874

transactions) to do so. Under Protocol 1, the collaboration is performed in such a way as to not reveal any information about a client's data, using differential privacy within a secure multi-party computation.

3.4.2 Protocol Timing Results

The use of simulation to evaluate the protocol allowed us to construct an accurate model of how long it would take to run such a protocol in the real world. To accomplish this, each simulation client timed each section of its own part of the protocol, capturing the actual time taken to run the Diffie-Hellman setup (once), the encryption and privacy steps (every iteration), and the local model training step (every iteration). The service agent captured the actual time taken to receive and store each client's encrypted model each iteration and to combine the models once per iteration. Our simulation also handles variable communication latency, with each pair of agents having a minimum latency plus a cubic "jitter" component that is randomly generated per message.

In Table 3.1 we provide the timing results of the various steps of our protocol on the credit card fraud dataset with all clients in different areas of New York City, as well as the total (simulated) time required for all parties to complete the protocol and produce a final shared model. Experiments were run for 30 iterations of Protocol

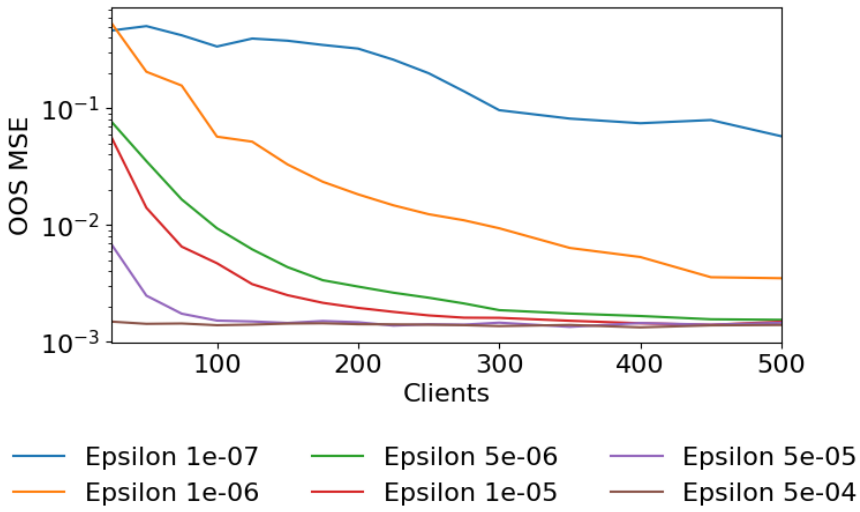


Figure 3.2: Out of sample loss function by number of parties for different values of ϵ privacy loss parameter.

1, within each of which each client runs 250 iterations of local regression training.

Each experiment was implemented in a single thread on a 24-core Intel Xeon X5650 at 2.6GHZ with 128GB RAM. Note that while the simulation is single-threaded, it does track a separate current time for each agent, uses the agent's current time when sending or receiving messages, and ensures that agents are not permitted to time travel or perform multiple activities in an overlapping manner. We assert that these times should therefore be a reasonable estimation of what would occur in a complete, distributed implementation of the protocol. The time required to perform each simulation, which is *not* the estimated real-world protocol time, ranged from 5 minutes for 100 parties to 28 minutes for 500 parties. We were able to run many simultaneous simulations (one per core) on the same system with only a slight increase to overall simulation time.

3.4.3 Protocol Accuracy Results

A key concern with the demonstrated protocol is that, while the secure multi-party computation (MPC) component introduces no accuracy loss to the collaborative train-

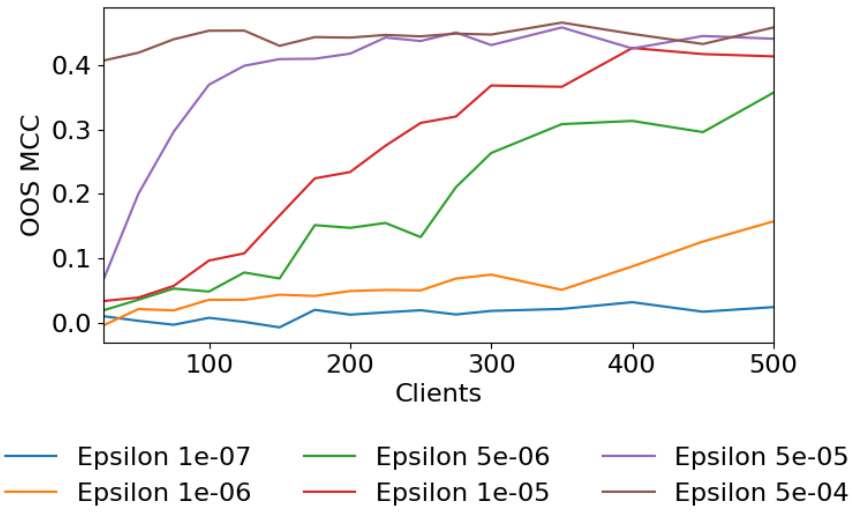


Figure 3.3: Out of sample Matthews Correlation Coefficient by number of parties for different values of ϵ privacy loss parameter.

ing effort, the differential privacy component does in inverse proportion to the ϵ privacy loss parameter. In Figure 3.2, we show the loss function value on the holdout test data using the securely-learned shared model at the end of the final protocol iteration. Smaller selections of the privacy loss parameter ϵ result in more uncertainty about a client's local weights when other parties collude to reveal them, but permit better loss minimization for the same number of parties and protocol iterations. For instance with 200 clients the training loss and model accuracy worsen dramatically once $\epsilon < 5e^{-5}$.

Matthews Correlation Coefficient: Because of the extreme class imbalance in the credit card fraud dataset, simple predictive accuracy would be a poor choice of measure. The proper classification for 99.8% of the examples is False (not fraudulent), therefore a naive classifier that always returns False would achieve a misleading 99.8% accuracy of prediction.

We instead assess our approach using the Matthews Correlation Coefficient (MCC).[94] MCC assesses binary classification performance even in the face of unbalanced output classes by accounting for the size of the true negative prediction set: Information not

captured by precision, recall, and the F-score.[95] MCC is a contingency method of calculating the Pearson product-moment correlation coefficient and therefore has the same interpretation. [96, 97, 98] For example, the MCC of the aforementioned naive classifier would be zero, indicating no correlation between the predicted and actual values.

Let $C(M, D)$ represent the confusion matrix between binary classification model M and data D, and recall that for classification output variable y, $True$ indicates fraud and $False$ indicates a non-fraudulent transaction. We can then define MCC for this problem as:

$$MCC(M, D) = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FN)(TP + FP)(TN + FP)(TN + FN)}} \tag{3.6}$$

where matrix entries $TP = C(True, True)$, $FP = C(True, False)$, $TN = C(False, False)$, and $FN = C(False, True)$.

In Figure 3.3 we show the MCC of our method's predictions with the correct values. Smaller selections of privacy loss parameter ϵ are better for privacy, but directly harm the accuracy of the learned model, so we cannot simply improve privacy by making ϵ arbitrarily small! We note that having more parties participate in the computation *does* permit lower values of ϵ while still producing an accurate shared model. For all evaluated client population sizes, models trained under Protocol 1 with $\epsilon \geq 5e^{-4}$ had similar mean accuracy to models trained with unsecured ("in the clear") federated learning.

3.4.4 Adversarial Data Recovery

We now consider that the untrusted server, or a group of other clients, may attempt to recover the unencrypted model weights of an honest client party.

Snooping Server

First suppose that the server, acting alone, attempts to infer the unencrypted weights of a particular client. For a single model weight, the value transmitted by the honest party h to the server is:

$$T = W_h + P_h + R_h \tag{3.7}$$

where W_h is the honest party's original weight, P_h is the differentially private noise added by the honest party, and $R_h = \sum_{c \in C} \pm R_{hc}$ for the set of all other clients C. Recall that for each client pair (h, c), one of them will add R_{hc} and the other will subtract it.

The server has a real problem! It does not know *any* of the pairwise client values that compose R_h. With 100 participating clients, a single client's R_h is a summation of 99 values randomly generated from the range $(0, 2^{32})$. The value of R_h is thus at least eight orders of magnitude greater than the range of W_h, leaving the server with no information about the private weights at all.

Colluding Clients

Now suppose that all the clients except one collude to recover the single honest client's locally trained model weights, so they might then make some inference about that client's private data. Let the honest client be h and the set of $n - 1$ colluding clients be C. For a single model weight, the final value contained in the shared model will be a known multiple of:

$$F = W_h + W_C + P_h + P_C + R - R \tag{3.8}$$

where W_h is the honest party's original weight, P_h is the honest party's differentially private noise, $W_C = \sum_{c \in C} W_c$, $P_C = \sum_{c \in C} P_c$, and R is the sum of all secure multi-

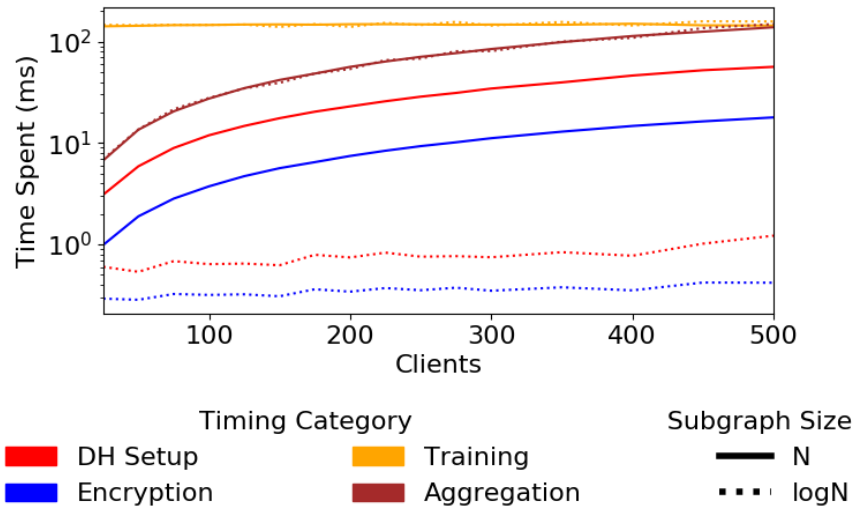


Figure 3.4: Protocol execution time by number of parties, contrasting two MPC peer group sizes.

party computation (MPC) values. Note that R is automatically removed when the computation is performed, because for each client pair i and j, one client added R_{ij} and the other subtracted R_{ij} from its transmitted values.

The conspirators precisely know terms W_C and P_C and can therefore recover the honest party's $W_h + P_h$. They cannot accurately infer the privacy noise P_h added by the honest party. For example the smallest privacy loss parameter value that does not prevent 100 clients from learning the shared model effectively is around $\epsilon = 5e^{-5}$. Using this, the mean absolute values for the first weight are $W_0 = 0.62$ and $P_0 = 0.38$. Thus there is still considerable uncertainty around the exact weight values of the honest party.

3.4.5 Peer Exchange Neighborhood

We have heretofore assumed that all N clients will partner with all other clients in the secure multi-party computation (MPC) peer exchange. That is, each client will modify their differentially private weights with $N - 1$ random large numbers, added or subtracted. This is ideal from a privacy perspective, because it means $N-1$ clients

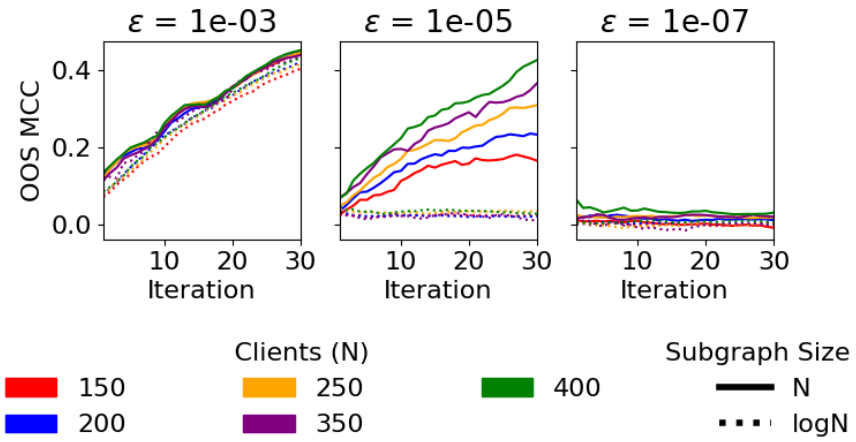


Figure 3.5: Out of sample Matthews Correlation Coefficient by protocol iteration for three values of ϵ privacy loss parameter, contrasting two MPC peer group sizes.

must collude to accurately remove the MPC encryption from one client's transmitted weights. However, for large client populations N, it imposes a growth factor of $O(N^2)$ to the overall protocol effort for these pairwise exchanges. It may therefore be acceptable to allow smaller "neighborhoods" (subgraphs) of MPC peer exchange. In Figures 3.4 and 3.5, we provide the mean execution time by task category, and the shared model accuracy, for peer exchange subgraphs of size N and $\log(N)$. Depending on the selection of privacy loss parameter ϵ, the use of a smaller peer group for key exchange provides a significant speed boost to certain elements of the protocol, but at the expense of either privacy or accuracy.

3.5 Conclusion

In this chapter, we have presented a differentially private secure multi-party computation protocol for federated learning intended to be accessible to an audience with some computation background, but without requiring prior knowledge of security, encryption, privacy, or distributed learning.

We have demonstrated the techniques on a common financial data set containing two days of anonymized credit card transactions, of which about 0.2% are labeled as

fraudulent, and shown how these techniques permit multiple parties (e.g. financial firms) to collaboratively learn a useful fraud detection model *without* sharing any of their client data or transactions, and with added layers of protection that make private data recovery difficult or impossible even with adversarial actors participating in the system.

Finally, we have implemented the full protocol on a real-world financial data set in an agent-based interactive discrete event simulation and conducted experiments to evaluate the accuracy and expected running time of the protocol for various numbers of participating parties, various values of the ϵ privacy loss parameter, and a multi-party computation neighborhood size of N vs $\log(N)$ peers. We hope that by enhancing ABIDES as an open source federated learning framework, we can reduce the cost and complexity of developing private learning implementations, and thereby encourage their commercial adoption by responsible practitioners.

In the next chapter, we leverage ABIDES in the cryptographic domain to develop a state-of-the-art improvement to PPFL, using a novel mechanism based on 1-2 oblivious transfer to protect against an extreme edge case: collusion of all other parties to reveal the sensitive data of a single "honest" party.

CHAPTER 4

OBLIVIOUS DISTRIBUTED DIFFERENTIAL PRIVACY SECURE AGAINST COLLUSION ATTACKS IN FEDERATED LEARNING

In this chapter, we use the extensions developed for ABIDES in Chapter 3 to protect federated learning against collusion attacks in which parties collaborate to expose an honest client's model parameters. For the sake of brevity, we refer the reader to Sections 1.2.1, 3.1 and 3.2 for comprehensive PPFL introduction and background, and present only new information here.

We build on a recent line of research that combines differential privacy and MPC to produce a secure federated learning protocol. [16, 90] These prior works provide strong protection against undesired inference by the server, but the collusion of enough clients can reveal the noisy weights of an honest client. This poses a problem because of the accuracy trade-off for differential privacy discussed in Chapter 3: the amount of random noise that can be added is limited by the need for acceptable model performance.

We propose a novel, efficient mechanism that protects against any attempt to undermine differential privacy by collusion of $n-1$ out of n total clients. Unlike prior works, we offer a protocol where the noise for each party is added in an oblivious way. Obliviousness *can* be achieved by running the noise generation inside the MPC, but such solutions are based on heavy cryptography machinery involving a significant amount of public key operations or incur increased communication complexity. [90, 99] In this work we focus on the concretely efficient aggregation protocol of Bonawitz et al which does not involve any public key operations in the learning phase. [16] We therefore provide the first practical protection against $n-1$ attacks by constructing an efficient oblivious distributed differentially private secure aggregation protocol.

4.1 Approach

Our approach combines secure multi-party aggregation with oblivious distributed differential privacy to better secure federated learning against $n-1$ collusion attacks. In this chapter, we again consider logistic regression as the local learning method, and each client update includes the weights of that logistic regression. The server receives the weights from all clients at each iteration and computes a new global model using the average of the client updates for each weight. Recall from Section 1.2.1 the literature demonstrating that private client data can be inferred from the trained model weights, which is clearly undesirable. The general task, then, is to secure each client's locally trained model weights against discovery while still learning an accurate shared model. We note that the collusion problem can be solved using generic MPC, but such generic solutions are impractical due to computational inefficiency. Our contribution is a practical and efficient solution to this problem using lightweight cryptographic tools.

Please review Sections 3.3.1 and 3.3.2 for discussion of our basic approach to eliminating per-weight and weighted average leakage during protocol execution.

We introduce a novel and efficient *oblivious distributed* differentially private mechanism. In prior works, each client picks its own local noise. By contrast, we offer a protocol where the noise for each party is added in an oblivious way. More specifically: For each weight, each client receives a tuple of encrypted noise terms from each other client and adds only a subset of them. Thus, a party P does not know the cleartext noise added to its weight and the other parties do not know which noise term is chosen by P.

We now show that the information leakage on the honest client's weights after the collusion attack is smaller than previous approaches. Our task is to enable the parties to calculate the sum of their inputs (i.e., $W = \sum_{i=1}^{n} w_i$), while ensuring privacy for

Protocol 2 Privacy-Preserving Federated Logistic Regression Protocol Π_{PPFL}

The protocol Π_{PPFL} runs with parties $P_1, \ldots, P_n$ and a server S. It proceeds as follows:
Inputs: For $i \in [n]$, party P_i holds input dataset D_i.
Public Parameters: $(\mathbb{G}, g, q)$ generated by $G(1^\lambda)$ and modulo p.
Π_{PPFL}.Setup(1^λ):

Round 1: Each party P_i for $i \in [n]$ proceeds as follows:

- Choose n secrets $a_{i,1}, \ldots, a_{i,n}$ uniformly and independently at random from $\mathbb{Z}_q$ and computes $(pk_{i,1}, \ldots, pk_{i,n}) = (g^{a_{i,1}} \bmod p, \ldots, g^{a_{i,n}} \bmod p)$.
- Generate random variables $\gamma_{i,j}^b$ and $\bar{\gamma}_{i,j}^b$ for $b \in \{0,1\}$ from the gamma $\mathcal{G}(1/n, scale)$ distribution with $scale = 2/(n * len(D_i) * \alpha * \epsilon)$.
- For all $j \in [n]$:
  (a) Generate random masks $s_{i,j} \in \mathbb{Z}_q$.
  (b) Compute masked noises $\eta_{i,j}^0 = s_{i,j} + \gamma_{i,j}^0 - \bar{\gamma}_{i,j}^0$ and $\eta_{i,j}^1 = s_{i,j} + \gamma_{i,j}^1 - \bar{\gamma}_{i,j}^1$.
- Each party P_i sends $pk_{i,j}$ and $\eta_{i,j}^0, \eta_{i,j}^1$ to S who permutes and randomizes them and forwards to party P_j.

Round 2: (Diffie-Hellman key exchange) Each party P_j for $j \in [n]$ proceeds as follows:

- Upon receiving all values $(pk_{1,j}, \ldots, pk_{n,j})$, compute the shared common keys $r_{i,j}$ for all $i \in [n]$ as follows:
  (a) Using the secret $a_{j,i}$ compute $c_{i,j} = c_{j,i} = (pk_{i,j})^{a_{j,i}} = (g^{a_{i,j}})^{a_{j,i}} \bmod p$.
  (b) Let $c_{1,j}, \ldots, c_{n,j}$ be the set of all common keys. Use a key-derivation function and set $r_{i,j} = r_{j,i} = H(c_{i,j})$

Π_{PPFL}.WeightedAverage($D_i, \{r_{i,j}\}_{j \in [n]}$):

Round 1: Each party P_i proceeds as follows:

- Compute the weights w_i, using Equation (1), of the local logistic classifier obtained by implementing regularized logistic regression on input D_i.
  We describe the algorithm for a single weight, denoted by w_i.
- Generate a random bit vector $b = (b_1, \ldots, b_n)$ and send y_i to the server S: $y_i := w_i + \sum_{j=i+1}^{n} r_{i,j} - \sum_{k=1}^{i-1} r_{k,i} + \sum_{j=1}^{n} \eta_{j,i}^{b_j} - \sum_{j=1}^{n} s_{i,j} \bmod p$.

Round 2: The server sends $W = (\sum_{i=1}^{n} y_i \bmod p)/n$ to all parties.

Π_{PPFL}.*Output*($1^\lambda, W$): Each party P_i upon receiving W repeats WeightedAverage for the next weight or iteration of the logistic regression with locally updated common keys r.

an honest party in the presence of a collusion attack given W. In prior works if $n-1$ parties collude, they subtract their weights and noise terms from W and then the final noise remaining in the transmitted weight of the honest party w_h is a single value chosen by the honest party. In our case, if $n-1$ parties collaborate then the final noise remaining in the transmitted weight of the honest party w_h is $n-1$ times larger, because the corrupted parties cannot subtract their noise terms.

At a high level, in our scheme, each client sends two encrypted noisy terms (permuted and randomized by the server) per model weight to the other clients, but each receiving client chooses only one of the two to add to each weight. Thus even if parties collude they cannot subtract a significant number of noise terms since they do not know which noise terms the honest client chose.

4.1.1 Network Topology & Threat Model

As is common in the federated learning setting, we opt for a star network topology, where there is one central party that is connected to all other parties. This central server can be distinct from the n original parties.

The protocols that we describe and compare against are secure in the semi-honest model. A semi-honest adversary follows the protocol correctly but tries to learn as much as possible about the inputs of the uncorrupted parties from the messages it receives. Furthermore, if there are multiple semi-honest corruptions, we allow the adversary to combine the views of the corrupted parties to potentially learn more information.

For a protocol of n parties, we offer security against any $n-1$ semi-honest corruptions or collusion of the server with $n-1$ semi-honest corruptions.

4.2 Secure Weighted Average Protocol

4.2.1 Our Protocol

In this section we formally describe our weighted average protocol Π_{PPFL}, depicted in Protocol 2, for secure logistic regression performed by a set of clients $(P_1, \ldots, P_n)$ and a server S.

During setup, every pair of parties P_i and P_j will share some common randomness $r_{i,j} = r_{j,i}$. In the online weighted average phase, client P_i sends its weights masked with these common random strings, adding all $r_{i,j}$ for $j > i$ and subtracting all $r_{i,k}$ for $k < i$. That is, P_i sends to server S the following message for its data w_i: $\bar{w_i} := (w_i + \sum_{j=i+1}^{n} r_{ij} - \sum_{k=1}^{i-1} r_{ki}) \bmod p$.

To establish common randomness, each pair of parties run the Diffie-Hellman Key exchange protocol [93] communicating via the server. The cryptographic primitives used in Protocol 2 include:

- An algorithm $G(1^\lambda)$, where λ is the security parameter, that outputs a representation of a cyclic group $\mathbb{G}$ of order q (with $||q|| = \lambda$) for which the discrete logarithm problem is believed to be hard (in other words, it is hard to guess x given g^x for particular groups $\mathbb{G}$).

- A key derivation function $H : \mathbb{G} \to \{0,1\}^\lambda$. It is assumed that if h is distributed uniformly in $\mathbb{G}$, then $H(h)$ is distributed uniformly in $\{0,1\}^\lambda$.

- A pseudorandom generator with double expansion, i.e., $PRG : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$.

The protocol is given for a single iteration of federated logistic regression. To handle the next iteration without the need to re-run the Diffie-Hellman key exchange we follow the same approach as in [16]: in the first iteration we use a pseudorandom generator $(r'_{i,j}, s) = PRG(r_{i,j})$ to both update the common randomness $r_{i,j} := r'_{i,j}$

and obtain a new random seed s. For subsequent iterations, instead of executing $\Pi_{\mathsf{PPFL}}.\mathsf{Setup}$, parties can run $PRG(s)$ to obtain a new $r'_{i,j}$ and the seed for the next iteration. Thus the parties need to run the exchange only once at the onset of the training. We generate the Laplacian noise in a distributed way by the use of gamma distributions $\mathcal{G}$ given that the Laplace distribution $\mathcal{L}$ can be constructed as the sum of differences of i.i.d. gamma distributions. To run machine learning algorithms and the DP mechanism which computes on rational values, we use field elements in a finite field $\mathbb{Z}_q$ to represent the *fixed-point values*. Concretely, for a fixed-point value $\bar{x}$ with k bits in the integer part and f bits in the decimal part, we use the field element $x := 2^f \cdot \bar{x} \mod q$ in $\mathbb{Z}_q$ to represent it.

4.2.2 Security of our Protocol

We prove that our protocol protects the privacy of honest users in the semi-honest setting given the topology in Section 4.1.1. Following the standard idea-real paradigm, we will show that when executing the protocol with a set of parties $\mathcal{U}$ of size n with threshold $t < n$, the joint view of the server S and any set of less than t users does not leak any information about the other users' inputs except what can be inferred from the output of the computation.

The view of a party P_i consists of its internal state (including its input w_i and randomness) and all messages this party received from other parties. The messages *sent* by this party do not need to be part of the view because they can be determined using the other elements of its view. Given any subset $\mathcal{C}$ of corrupted parties out of the n parties in $\mathcal{U}$, let $\mathsf{REAL}_{\mathcal{C}}^{\mathcal{U},t,k}(w_1, \ldots, w_n)$ be a random variable representing the combined views of all parties in $\mathcal{C}$ in the above protocol execution, where the randomness is over the internal randomness of all parties. We are going to show that there exists an efficient simulator that, for every choice of the honest clients' inputs, outputs a simulation of the adversarial participants' view of the protocol run

whose distribution is computationally indistinguishable from the distribution of the adversaries' view of the real protocol run.

The following theorem shows that the joint view of any subset of less than t users and the server can be simulated without knowing the secret input of any other users. In other words, the adversary controlling less than t users cannot learn anything other than the output of the computation. In our protocol we consider the leakage L learned from the difference of the noise terms η^0, η^1. Note that this leakage does not affect the error function given later in Definition 2.

Theorem 1 (Semi-Honest Security, against t clients)**.** *For security parameter λ, an n-party protocol for an aggregation function f and all leakage L of size $O(n)$ is L-secure if there exists a PPT simulator SIM such that for all $k, t, \mathcal{U}$ where $t < |\mathcal{U}|$, all corrupted parties $\mathcal{C} \subseteq \mathcal{U}$ and all $w_{\mathcal{U}}$, which denote all secret inputs of all users in all iterations k, letting $n = |\mathcal{U}|$, then the output of SIM is computationally indistinguishable from the output of $\mathsf{REAL}^{\mathcal{U},t,k}$:*

$$\mathsf{REAL}_{\mathcal{C}}^{\mathcal{U},t,k}(\lambda, w_{\mathcal{U}}, L) \approx_c \mathsf{SIM}^{\mathcal{U},t,k}(\lambda, w_{\mathcal{C}}, L)$$

Next we argue that the error term on the honest client's inputs after the collusion attack of t parties is larger than previous approaches. For this, we require the following additional property. Consider the case of $n-1$ collusion; we define collusion privacy as follows:

Collusion-Privacy: An n-party protocol provides *Collusion-Privacy*, for an aggregation function f and a probability distribution $\mathcal{D}$, if any adversary, who controls all parties except client P_h, learns no more than the honest party's values $w_h + \eta$ where $\eta \leftarrow \mathcal{D}$ and $f(w_1, \ldots, w_n)$.

In prior works if $n-1$ parties collude then the final noise left in the weight of the honest party w_h is a single value from $\mathcal{D}$. In our case, if $n-1$ parties collaborate then

the final noise left in the weight of the honest party w_h is $n-1$ times larger than $\mathcal{D}$ since the corrupted parties cannot subtract their exact noise terms.

To measure the error, we quantify the difference between $f(D)$ and its perturbed value $\hat{f}(D)$ which is the error introduced by the differential private mechanism of the secure aggregation protocol.

Definition 3. (Error function) Let $D \in \mathcal{D}$, $f : \mathcal{D} \to \mathbb{R}$, and let $\delta = \frac{|f(D)-\hat{f}(D)|}{|f(D)|+1}$ (i.e., the value of the error). The error function is defined as $\mu = \mathbb{E}(\delta)$. The expectation is taken on the randomness of $\hat{f}(D)$. The standard deviation of the error is $\sigma = \sqrt{Var(\delta)}$.

After the execution of Protocol 2, parties receive the noisy sum of their inputs, i.e., $W = \sum_{i=1}^{n} w_i$, In prior non-oblivious works if $n-1$ parties collaborate and remove their weights from w then the final noise added to the weight of the honest party w_h is a value from $\mathcal{L}(\lambda)$, and hence, the error is $\mu = \frac{1}{|W|+1}\mathbb{E}|\mathcal{L}(\lambda)| = \frac{\lambda}{|W|+1}$.

In our oblivious case, if $n-1$ parties collaborate then the final noise added to the weight of the honest party w_h is $n-1$ times larger than $\mathcal{L}(\lambda)$, and hence, the error is $\mu = \frac{1}{|W|+1}\mathbb{E}|\sum_{i=1}^{n-1}\mathcal{L}(\lambda)| = \frac{(n-1)\cdot\lambda}{|W|+1}$.

However, in practice even if the parties cannot subtract their exact noise terms they can still try to subtract the average of the noise terms, or one of the two noise terms, or use the leakage L to reduce the amount of error. In Section 4.3.4 and Figure 4.3 we empirically show that such an attack is little better than the attack of subtracting nothing.

Note that the output of the aggregation protocol, $W+\eta$, is generated such that η follows exactly the same distribution in both non-oblivious and oblivious cases, but the noise left after an $n-1$ attack against the oblivious case is higher.

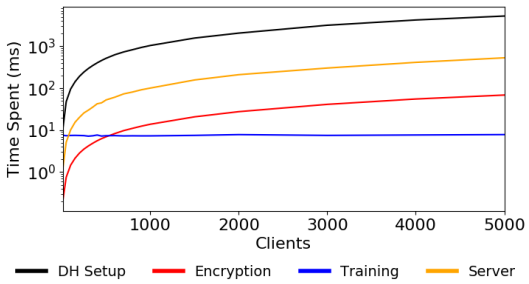
4.3 Experiments

We implemented our novel protocol for secure federated learning with oblivious distributed differential privacy within ABIDES, as presented in Chapter 2 and adapted to the domain of federated learning in Chapter 3. Following the same approach, we simulated our oblivious protocol for 5,000 distributed clients and analyzed the timing, accuracy, and privacy of the empirical results.

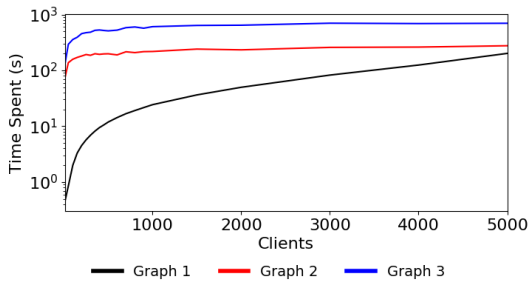
4.3.1 Experimental Dataset and Method

We evaluated our protocol's performance using the Adult Census Income dataset [100], which provides 14 input features such as age, marital status, and occupation, that can be used to predict a categorical output variable identifying whether (*True*) or not (*False*) an individual earns over $50K USD per year. We used a preprocessed version of the dataset from Jayaraman et al following the method of Chadhuri et al which transformed each categorical variable into a series of binary features, then normalized both features and examples, resulting in 104 features for consideration. [90, 101] We added a constant intercept feature to permit greater flexibility in the regression. Of the 45,222 records in our cleaned data set, there were 11,208 positive examples (about 25%), representing a moderately unbalanced dataset.

The dataset was loaded only once per complete simulation of the protocol, after which a randomized train-test split (75% vs 25%) was taken. Once per round of federated learning, each client randomly selected 200 rows from the training data as its "local" data. The holdout test data was the same for all clients, and no client ever trained on it. The clients implemented our described protocol using oblivious distributed differential privacy, such that no information about client data is revealed, even during an extreme $n-1$ collusion attack by all other parties.

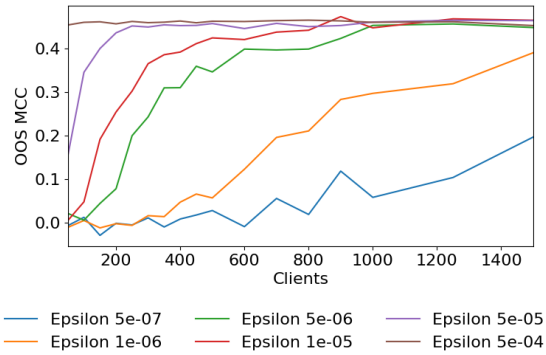


(a) Time spent per protocol component on Graph 1.

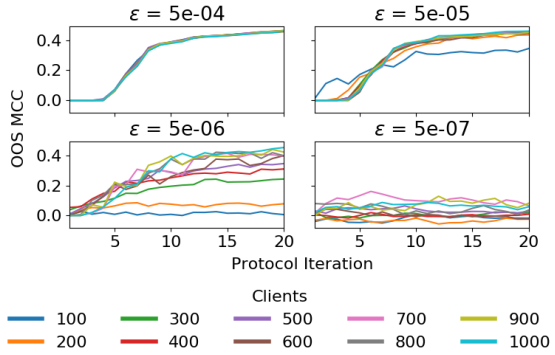


(b) Total protocol execution time for Graphs 1-3.

Figure 4.1: Timing results for our oblivious distributed differential privacy protocol.



(a) Performance of final shared model by ϵ privacy loss parameter.



(b) Performance per protocol iteration by ϵ and client count.

Figure 4.2: Out of sample performance (Matthews Correlation Coefficient) for our oblivious distributed differential privacy protocol.

4.3.2 Protocol Timing Results

To accurately estimate the time complexity of the protocol, we evaluated it within the ABIDES discrete event simulation environment. The simulation permits construction of an arbitrary network graph with defined pairwise connectivity, minimum latency, and parameters for randomly selected "jitter" with nanosecond resolution. It also captures the real elapsed runtime of each client activity and appropriately delays both sent messages and the earliest time at which a client may act again. Each experiment was implemented on a 24-core Intel Xeon X5650 at 2.6GHZ with 128GB RAM. The mean time required to run the protocol simulation on a single CPU core

ranged from 32 seconds for 100 parties to 12 hours for 5,000 parties.

Figure 4.1a summarizes the time spent performing each section of our protocol on the adult census income data set: **Diffie-Hellman Setup** one time per client, **Encryption** of the weights and local model, **Training** per client per protocol iteration, and **Server** aggregation time per protocol iteration. Figure 4.1b shows the estimated time required to run the full protocol (not the simulation) for three different network graphs: **Graph 1** places all participants around New York City, **Graph 2** places the server in New York City and clients around London, **Graph 3** places the server in New York City and clients all over the world. All experiments comprised 20 rounds of secure federated learning, with each client running 50 iterations of local regression training at each round. Latency is the most significant time component for small participant networks, but as the population size grows, computation effort surpasses it. Fortunately, the two largest components of computational time growth represent work performed only once per client for the entire protocol, and work performed only by the server, respectively.

In Table 4.1, we present categorized millisecond timing for Protocol 2. The table outlines four categories:

- *Server* includes all data reception, storage, and aggregation tasks by the server presented as mean time per protocol iteration.
- *DH Setup* includes the one-time Diffie-Hellman Key Exchange and noise generation presented as mean time per client party.
- *Training* includes the time spent performing logistic regression on local training data presented as mean time per client per protocol iteration.
- *Encryption* includes the time spent applying encryption randomness, generating the next encryption randomness, and applying privacy noise presented as mean time per client per protocol iteration.

Table 4.1: Categorized protocol time in milliseconds.

Users	Server	User DH Setup	User Training	User Encrypt
200	20.3	193.0	7.5	2.9
400	42.7	404.0	7.7	5.6
600	61.4	631.6	7.4	8.3
800	81.5	832.1	7.4	11.1
1000	101.1	1,046.5	7.3	13.9
3000	303.8	3,185.0	7.5	41.3
5000	532.5	5,277.9	7.9	68.9

The total number of messages sent by the parties grows quadratically with n for the one-time *DH Setup* phase, and linearly with n for each *Encryption* phase. No messages are sent for the *Training* phase.

These timings were generated using network graph 1, which is the case that places all parties including the server at random locations around New York City. We note, however, that choice of network graph should not affect these component times, only the communication latency for the full protocol running times shown previously.

4.3.3 Protocol Accuracy Results

The secure multi-party aggregation component of our protocol does not cause any loss of accuracy, because in each round of federated learning, the MPC encryption elements sum to zero in the final shared model. Differential privacy, whether in the context of our new protocol or a prior approach, *does* introduce final model accuracy loss inversely proportional to the ϵ privacy loss parameter. Smaller selections of ϵ result in more uncertainty about a client's local weights when other parties collude to reveal them, but increasingly confound learning. For example, in our protocol experiments with 200 clients, final model accuracy worsens dramatically once $\epsilon < 5e - 5$.

Matthews Correlation Coefficient: Because of the significant (3:1) class imbalance in the adult census income data set, simple predictive accuracy would be a

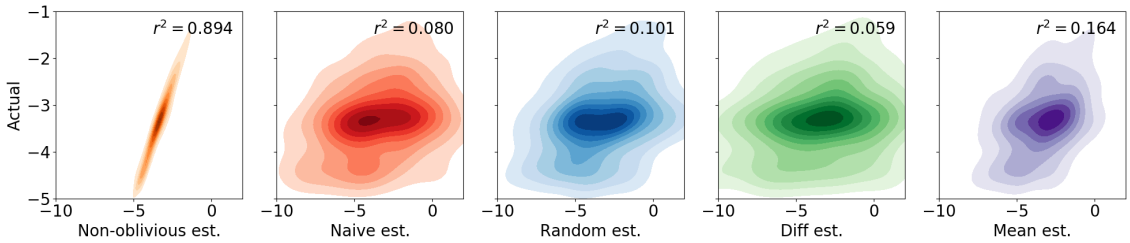


Figure 4.3: Density plot of actual versus estimated honest party weight over 1,000 iterations of the $n-1$ collusion attack.

poor choice of measure. We instead assess our approach using the Matthews Correlation Coefficient (MCC) [94], a contingency method of calculating the Pearson product-moment correlation coefficient (with the same interpretation), that is appropriate for imbalanced classification problems. [95, 96, 97, 98]

In Figure 4.2a we show the MCC of our protocol's final shared model predictions against the correct values for a range of ϵ. As expected, smaller ϵ harms the accuracy of the learned model. Thus there is a dynamic lower bound, varying with population size, on useful values of ϵ. For example when considering out of sample $MCC(n)$, with n being the client population size, in our experiments with $\epsilon = 1e-5$: $MCC(100) = 0.005$, $MCC(200) = 0.254$, and $MCC(500) = 0.423$. For all evaluated client population sizes (50 to 5,000), models trained under Protocol 2 with $\epsilon \geq 5e-4$ had similar accuracy to unsecured federated learning. Figure 4.2b shows the impact ϵ can have on each round of federated learning: with $\epsilon = 5e-4$ or $\epsilon = 5e-7$, population size does not matter because either all sizes succeed at learning or none do; but with $\epsilon = 5e-6$, varying client population sizes learn at vastly different rates.

In our simulated network environment, with 1000 clients implementing the described protocol for federated logistic regression using privacy loss parameter $\epsilon = 5e-4$, we found an out of sample final iteration relative accuracy loss (MSE verus learning in the clear) of $1.1e-6$ and an out of sample final iteration relative MCC loss of 0.0018.

4.3.4 Adversarial Data Recovery

Prior works like Bonawitz et al discuss attacks from a "snooping" server which attempts to infer the unencrypted weights of a particular client. [16] The attacks fail since the server does not have the common random values r.

Here we consider the security of the protocol against attacks from within the participant population. The first attack is defended well by previous non-oblivious protocols, but the second is not. Our oblivious protocol defends well from both attacks.

Snooping Server

First suppose that the untrusted server, acting alone, attempts to infer the unencrypted weights of a particular client. For a single model weight M, the value transmitted by the honest party h to the server is:

$$M_h = w_h + T_h + R_h \tag{4.1}$$

where w_h is the honest party's original weight, T_h is the total privacy noise added by h, and $R_h = \sum_{c \in C} \pm r_{ch}$ for the set of all other clients C. Recall that for each client pair (h, c), one of them will add R_{ch} and the other will subtract it. In the case of the server acting alone, we do not need to be concerned with the details of T_h, because the weights are already heavily encrypted against the server by MPC.

When the server tries to solve the problem of reconstructing w_h from M_h, it does not possess any of the pairwise client values r_{ch} that compose R_h. With just 100 participating clients, a single client's R_h is a summation of 99 values randomly generated (in our case) from the range $(0, 2^{32})$. The value of R_h is therefore orders of magnitude greater than the range of w_h, leaving the server with no meaningful information about the honest client's private model weights.

As noted, this relatively simple case is handled well by the secure aggregation protocol alone and successfully defended by prior works in our area. The next case is not.

Collusion attack

We consider the $n-1$ attack, in which *all other clients* conspire to recover the unencrypted model weights of a single "honest" client. Let the honest client be h and the set of $n-1$ colluding clients be C. For a single model weight, let $F = w_h + W_C + T_h + T_C$ be the output of the aggregation protocol at the end of each iteration, where w_h is the honest party's original weight, T_h is the honest party's noise sum, $W_C = \sum_{c \in C} w_c$, $T_C = \sum_{c \in C} T_c$. Note that the sum of the randomness r, s is removed by design from the output when the computation is performed, so MPC cannot defend against this type of attack.

Under prior non-oblivious protocols in which each party generates and adds its own noise locally, the colluding parties know W_C and T_C and can therefore recover the honest party's noisy weights $W_h + T_h$.

Under our oblivious protocol, the noise which cannot be subtracted in F has a more dispersed distribution that cannot be much narrowed by the colluding parties since h will have received from each colluding party c a choice of two difference of gamma privacy noises $\hat{\gamma}^0_{ch}$ and $\hat{\gamma}^1_{ch}$ and c will not know which was selected by h. Moreover, since the server permutes and randomizes the encrypted noise terms, T_C is also not precisely known to the colluding parties. For an extreme Sybil attack, the server will have to run a secure shuffle protocol which we have not implemented in this version.

We empirically illustrate the dramatic improvement in privacy against an $n-1$ distributed attack by considering five cases. In each case, the colluding parties attempt to recover W_h and always remove W_C. In the **Non-Oblivious** case followed

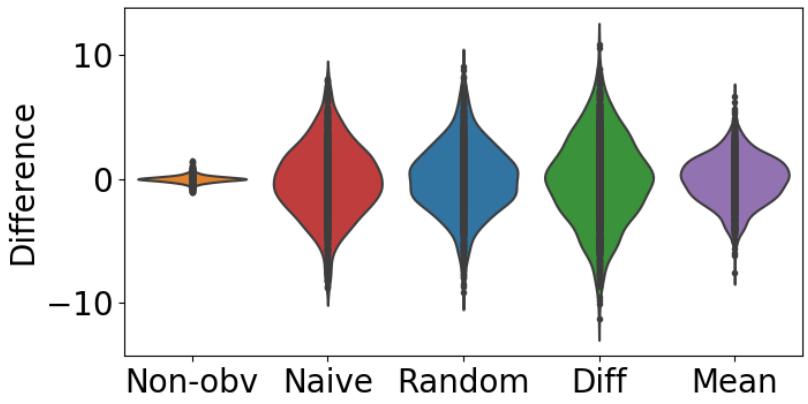


Figure 4.4: Distribution of difference between estimated and actual honest party weight over 1,000 iterations of $n-1$ collusion attack.

by prior works, the corrupted parties can accurately remove T_C. In the other four cases, Protocol 2 is attacked, and the corrupted parties must decide how to deal with the unknown noise choices by other parties i: under **Naive** they do nothing additional (i.e., they do not remove any noise terms); under **Random** each corrupted party c removes either $\hat{\gamma}_{ci}^0$ or $\hat{\gamma}_{ci}^1$ at random; and under **Diff** and **Mean** each corrupted party c removes the difference or mean of $\hat{\gamma}_{ci}^0$ and $\hat{\gamma}_{ci}^1$ respectively.

We consider the recovery attempt across 1,000 full iterations of Protocol 2 with 100 clients participating. At the end of every iteration, 99 clients share information in an attempt to recover the unencrypted model weights of the one honest client. Privacy loss parameter $\epsilon = 5e-4$ was selected because it did not cause significant shared model accuracy loss for any tested number of parties.

Figure 4.3 shows a density plot of the honest party's actual model weight versus the collaborators' estimate of that weight. Figure 4.4 summarizes the distribution of the difference between estimated and actual weights for each attack scenario. The $n-1$ attack is successful ($r^2 = 0.894$) against the prior non-oblivious protocol, but not successful ($r^2 = 0.164$ or worse) against our new oblivious protocol.

Additional Residual Noise: We have discussed a key aspect of our protocol's

security against an $n-1$ collusion attack, that given final model output for a single weight:

$$F = w_h + W_C + T_h + T_C \tag{4.2}$$

where w_h is the honest party's original weight, T_h is the honest party's noise sum, $W_C = \sum_{c \in C} w_c$, and $T_C = \sum_{c \in C} T_c$, the conspirators C cannot accurately infer T_h because each conspirator c does not know whether h added $\hat{\gamma}^0_{ch}$ or $\hat{\gamma}^1_{ch}$ to its weight.

However, we only briefly touched on a second aspect of our protocol's resistance to the $n-1$ attack. Not only can the conspirators not remove T_h, they actually cannot accurately remove the T_C either. This is because Protocol 2 also encrypts the oblivious distributed differential privacy noise choices. When h generates $\hat{\gamma}^0_{hc}$, it is a composition:

$$\hat{\gamma}^0_{hc} = \gamma^0_{hc} + s_{hc} \tag{4.3}$$

where γ^0_{hc} is the initial noise choice generated from a difference of gamma distributions and s_{hc} is a large value. Honest party h adds the same s_{hc} to $\hat{\gamma}^0_{hc}$ and $\hat{\gamma}^1_{hc}$. It later subtracts s_{hc} from its own transmitted weights to avoid disturbing the calculation. For the case where s_{hc} is generated based on additive secret sharing, the honest party uses a different s_{hc} for its own transmitted weights and this attack does not apply because the sum of all the s's is zero.

Thus conspirator c knows $\hat{\gamma}^0_{hc}$ and $\hat{\gamma}^1_{hc}$, and knows which one it selected. However the encryption randomness s_{hc} has already been removed from the final shared output weight, leaving only the unencrypted γ^0_{hc} or γ^1_{hc}. Conspirator c must therefore remove only the appropriate γ_{hc} and *not* $\hat{\gamma}_{hc}$, but it cannot do this, because it does not know what s_{hc} was.

We did not simulate this additional error in the current work. With n parties, this will cause the collaborators C in our attack cases to be *even more wrong* since they can only approximate the $n-1$ values of s's chosen by the honest party for each

weight of h, when c removes values based on encrypted $\hat{\gamma}_{hc}$ instead of unencrypted γ_{hc}.

Thus under our Protocol 2, for each collaborator c, the additional error (beyond basic differential privacy noise) in the estimate of w_h will compose two factors:

1. Not knowing whether h added $\hat{\gamma}^0_{ch}$ or $\hat{\gamma}^1_{ch}$.

2. Being off by large encryption value s_{hc} when subtracting $\hat{\gamma}^0_{hc}$ or $\hat{\gamma}^1_{hc}$.

We evaluated the first error in this work and left detailed analysis of the second for future work, although it is already part of the additional security of our protocol.

4.3.5 Additional Data Set

To further validate the new approach for oblivious distributed differential privacy, we also ran Protocol 2 and a previous non-oblivious protocol against the Kaggle Credit Card Fraud data set [91] and evaluated the success of the same cases of $n-1$ attacks. This data set provides transformed features that represent the first 26 principal components of unknown original features. Two original features are provided without transformation: the elapsed time from the start time of the dataset and the amount of the transaction. We used the Amount column without transformation, but excluded the Time column because our learning method does not attempt to identify temporal clusters or patterns. The dataset provides a categorical y variable identifying whether the transaction was judged fraudulent (True) or not (False). Of the 284,807 records, only 492 (less than 0.2%) are labelled fraudulent, representing an extremely unbalanced dataset. The data set was otherwise handled in exactly the same manner as the adult census data set.

In Figures 4.5 and 4.6, we show the result of 1,000 attempted $n-1$ attacks against an arbitrarily-selected honest party weight, using the credit card fraud data set. Because of the randomness involved, the magnitude of difference varies across

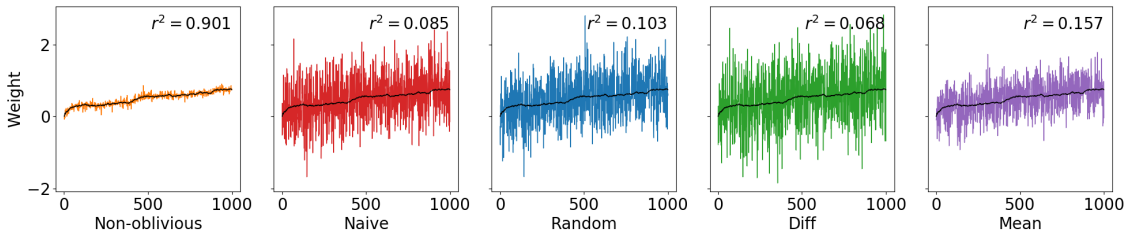


Figure 4.5: Actual (black) versus estimated (color) honest party weight over 1,000 iterations of $n - 1$ collusion attack.

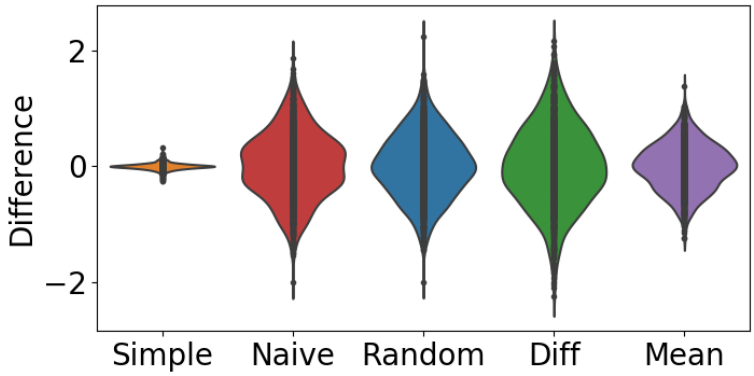


Figure 4.6: Violin plot showing distribution of difference between estimated and actual honest party weight.

the model weights.

4.4 Conclusion

In this chapter, we presented an efficient mechanism for oblivious distributed differential privacy that is the first to secure against $n - 1$ collusion attacks on the clients' model parameters, and leveraged that mechanism to construct a secure federated learning protocol. We also detailed the protocol and proved its security against a semi-honest adversary. We have left the case where clients drop off during the protocol as future work. Our mechanism is therefore well suited to cross-silo federated learning applications where clients are different organizations (e.g. medical or financial) or geodistributed datacenters, as opposed to mobile or IoT devices which can go

offline.

To empirically evaluate the protocol in a practical setting, we again leveraged ABIDES to simulate its performance with 5,000 parties on two common data sets, and estimated its accuracy and running time for various client counts and values of the ϵ privacy loss parameter. We also conducted an $n-1$ attack and showed that it is effective against prior non-oblivious protocols, but not against our new protocol. In the next chapter, I extend the ABIDES financial market simulation with a new set of spoofing strategies, use them to generate synthetic spoofing examples, and learn to detect spoofing.

CHAPTER 5
DETECTION OF ILLEGAL SPOOFING IN FINANCIAL MARKETS

As I discussed in the Introduction, a particular focus of my work with ABIDES has been to enable studies of intelligent agent behavior that cannot or should not be otherwise performed, for example due to unavailable data or illegal activities. I believe simulation will play an important role in the future regulation of artificial intelligence, and an important first step is the ability to reliably detect that an agent has learned to violate some particular legal boundary.

In Section 1.3, I discussed the literature on financial market simulation, and shared the relevant statute and guidance that controls spoofing in financial markets. In this chapter, I construct an approach to automated spoofing detection. This is a difficult problem for several reasons:

- Each individual action taken by a trading agent, for example buying stock or canceling a previous order, is entirely normative; only in certain combinations might a questionable behavior arise.
- Publicly-available stock market data (as shown in Figure 2.1) is anonymized; identifying information is lost as orders flow through the financial system.
- Spoofing is not well-defined and is similar to necessary market making behaviors; as discussed in Section 1.3.3, legal and regulatory behavior relies on intent, requiring that automated detection focus on behaviors that "look like" spoofing.

To approach this complex area, I adapt the ABIDES simulation to generate synthetic examples of potentially illegal spoofing behaviors and explore how best to distinguish them from non-spoofing behaviors.

5.1 Related Work

I am not the first investigator to consider the detection of spoofing in financial markets. In the 2014 work most closely aligned with my own, Cao et al. described two forms of deceptive electronic price manipulation. [102] In the first, called *spoofing trading*, an actor generates the illusion of increased demand over a relatively long time horizon (perhaps 30 minutes) by maintaining a large order on the opposite side of the order book. In the second, *quote stuffing*, there is a quick, aggressive effort to generate the illusion of a bidding war (or actual price movement) by placing a sequence of orders inside the bid-ask spread. The researchers smooth temporal order data and transform it to relative price offsets, then investigate the use of K-Nearest Neighbors and Support Vector Machines to detect potential spoofing sequences with good results. Cao et al. rely on a similar transformation to mine and do inject synthetic data, but where I use an agent-based model for the entire market, they use historical data for four stocks, augmented by synthetic data constructed as random perturbations around the mean historical behavior of real actors as disclosed through regulatory actions. Without an agent-based model, other market participants obviously cannot "react" to the injected spoofing sequences.

Detection of spoofing has become an area of increasing worldwide academic interest. Leangarun et al. investigated whether a feedforward neural network with access to only Level 1 data (i.e. successful transactions) could identify market manipulation. [103] They found that it could detect the more classic "pump and dump" style of manipulation, but could not detect spoofing. Mendonça et al. obtained private data from a Brazilian brokerage firm with a consistent (but anonymous) identifier for each trader and used a decision tree constructed from expert domain knowledge to demonstrate that likely spoofing behavior could be identified, recommending (as I do) that "brokerage firms might prevent reoccurrences of this practice, showing

greater diligence, which is important to promote ethics in the capital markets". [104] Li et al. applied a variety of classification methods to labeled daily and tick spoofing data obtained from Chinese regulators after publicized enforcement actions. [105] Surprisingly, they were able to use the daily data to reliably detect days during which spoofing had occurred, but were not able to detect spoofing with the fine-resolution tick data. This may be explained by a lack of transformation of the tick data or the nature of the classifiers employed. Wang et al. used Generative Adversarial Networks (GAN) to train both a spoofing agent which learns to avoid detection by disguising its activity as market making, and a detector that attempts to defeat this evasion. [106] The quite interesting converged result is that the spoofer can learn to evade detection, but must eventually "hide" its behavior so well that it cannot profit from it.

5.2 Spoofing and AI Regulation

Spoofing is a form of price manipulation through placement of false orders, intended to deceive other investors (or their trading algorithms) into raising or lowering a stock price in response to the appearance of false demand. The "spoofer" will profit from this temporary and erroneous pricing in the usual ways: selling previously-acquired stock above true market value, or perhaps purchasing desired stock below true market value. A typical spoofing sequence follows:

1. Spoofer buys, or already owns, shares of ABC stock.

2. Over time, spoofer places limit orders to buy ABC stock for a little *less* than the current market value. Because the orders do not represent the current best price, they will not be transacted.

3. Other traders observe the order book, note the increasing demand to own the stock, and predict the price will rise. They buy the stock, driving up the price.

4. Spoofer updates the price of their false orders to keep them near the best price, while ensuring they never *become* the best price.

5. When the price has risen sufficiently, spoofer sells their position in ABC stock and cancels the false orders, allowing the price to fall as the false demand evaporates.

Spoofing is distinguished from the better-known "pump and dump" scheme by its mechanism. In a classic pump and dump, the manipulator affects stock prices through actions outside the stock market: calling into a radio show, writing a newspaper editorial, or tweeting about the stock. When spoofing, the manipulator acts inside the stock market, placing and cancelling orders to give a technical impression of rising demand.

The regulations surrounding spoofing, and some other forms of manipulation, present a particular challenge for intelligent agents because they rely on the *intent of the actor*. This is important because, for example, the cancellation of orders is both legal and necessary when new information arrives, a client's situation changes, or one is undertaking a market maker role which requires the maintenance of positions at certain relative price levels. It only becomes abusive when a market participant *already* intends to cancel an order *at the time it is originally placed*. This raises an interesting question beyond the scope of my current work:

> What is the intent of a learning algorithm that displays behavior its creator did not expect?

The answer to this question will not likely come from the field of computer science, but it is clear regardless that laws and regulations will have to adapt to the presence of intelligent agents.

Here I propose a first step: training a learning algorithm to identify (this chapter) and avoid (Chapter 6) action sequences that *look like* known spoofing behaviors,

rather than referencing intent. If successful, this could offer a new direction for regulators to explore: encouraging or requiring best practices that include the use of such trained detectors to avoid behaviors that are measurably similar to spoofing or other disruptive trading practices.

5.3 Simulation of Spoofing Behavior

The aggregated and anonymous nature of public market data complicates any study of spoofing, because it is a sequential behavior rather than a singular action, and using public data no two orders can be traced to the same actor. One could in theory obtain known examples of the behavior by spoofing a real market and observing the effects, but of course this is illegal, and conducting research in this manner would be unethical.

Instead I experiment within the ABIDES simulation, starting with a population of existing agents that pursue various greedy but legal strategies, and to that introduce a new agent that profits from an illegal spoofing strategy. There are two high level goals: the spoofing agent must be profitable, and must be identifiable using only its publicly observable actions.

As a baseline population for this exercise, I instantiate: 500 Zero Intelligence (ZI) agents, which were previously described; 500 Value agents, which arbitrage the current market price against a private extrinsic valuation; and 10 Order Book Imbalance (OBI) agents, which represent high-frequency liquidity traders predicting short term price moves by studying the order book.

To this, I introduce a spoofing agent, following a strategy similar to that outlined in the previous section. The spoofing agent observes the simulated market for a time to understand current prices, then purchases some stock at a "fair" price:

$$\mathbb{E}[p_e] \leq \min\left[\frac{a_w^L + a_w^H}{2}, p_e'\right] \tag{5.1}$$

Table 5.1: Effect of spoofing activity on agent profit. OBI* is the high frequency agent nearest the exchange. It is expected to outperform slower OBI traders.

Depth	Quantity	OBI	OBI*	Spoofer	Value	ZI
honest		-1,811	2,158	283	-39	67
3	1000	-128,695	22,897	45,419	572	1,606
	1500	-261,974	-13,720	85,339	983	3,587
	2000	-261,608	-21,402	84,777	974	3,608
	2500	-257,309	-19,480	83,491	947	3,557
4	1000	-126,323	24,520	47,253	598	1,531
	1500	-266,288	-14,477	88,450	993	3,650
	2000	-272,594	-17,817	89,629	1,007	3,756
	2500	-270,460	-19,416	88,728	992	3,738
5	1000	-127,505	23,970	47,164	599	1,553
	1500	-262,329	-14,210	87,024	973	3,601
	2000	-267,877	-15,057	89,179	1,004	3,668
	2500	-267,769	-15,494	88,985	1,011	3,661
6	1000	-119,003	23,125	45,236	553	1,451
	1500	-261,368	-8,634	86,783	971	3,576
	2000	-272,012	-17,184	91,477	1,022	3,724
	2500	-273,144	-16,386	90,670	1,009	3,757
7	1000	-125,853	24,119	45,996	584	1,539
	1500	-259,638	-11,992	86,288	973	3,549
	2000	-283,719	-21,091	91,876	1,057	3,908
	2500	-280,505	-19,628	91,125	1,041	3,865
8	1000	-127,719	24,664	45,271	571	1,588
	1500	-270,863	-15,252	89,794	980	3,745
	2000	-289,257	-21,261	92,789	1,093	3,971
	2500	-286,847	-20,755	91,499	1,078	3,944

where $\mathbb{E}[p_e]$ is the expected entry price for its desired position, p'_e is the prior entry price, and a_w^L and a_w^H are respectively the lowest and highest observed *best ask* prices during the warmup period. It then begins to stimulate demand by placing limit orders to buy stock at a configurable quantity (number of shares) and depth (worse than the best price). It cancels and repositions these orders to ensure they remain near the best price but are never transacted. Once a configurable profit target has been reached, the agent sells its stock, cancels the spoofing orders, and waits for the price to settle down using a configurable "cooldown" period. It will then repeat the full behavior again.

5.4 Effect of Spoofing on Profitability

To test the efficacy of the spoofer, and its effect on other agents, I simulated a total of 1,160 full market days with 49 agent configurations using the method of common random numbers to reduce variance across configurations. Forty-eight configurations included spoofing behavior: each pairwise combination of $depth \in \{3, 4, 5, 6, 7, 8\}$ and $quantity \in \{1000, 1500, 2000, 2500, 3000, 3500, 4000, 5000\}$. The final configuration still included the spoofing agent, but configured in an "honest" mode that omits the manipulative portion of its strategy. Each spoofing configuration was simulated for 20 market days; the non-spoofing configuration was simulated for 200 days. The mean profit per class of agent is presented in Table 5.1. The effect of varying spoofer quote depth and quote size on the profit distribution of each agent is presented in Figures 5.1 and 5.2. From these, I note concretely that:

- The order book aware agents (OBI) are harmed by spoofing. OBI* is harmed least. Harms increase with spoofing order quantity up to a limit. Quote depth is irrelevant within the tested range.

- Value and ZI agents are indirectly helped by spoofing, presumably because the

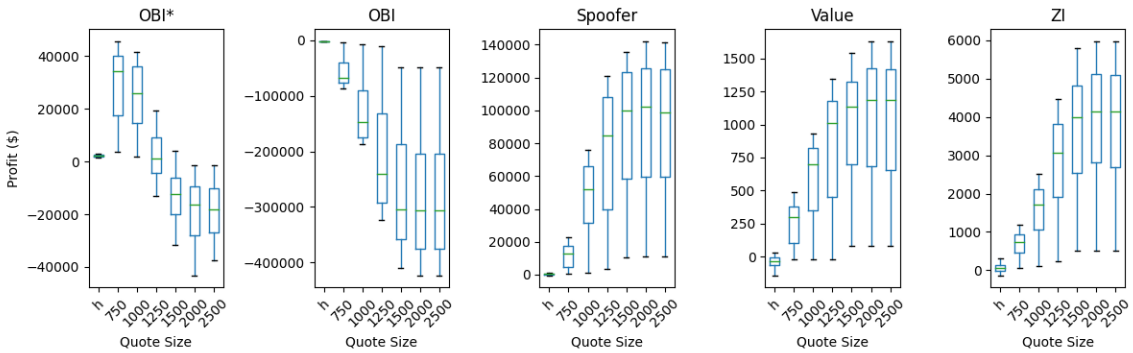


Figure 5.1: Boxplot showing effect of spoofer quote size on profitability of each agent class. Includes all quote depths. Quote size h indicates "honest" spoofer.

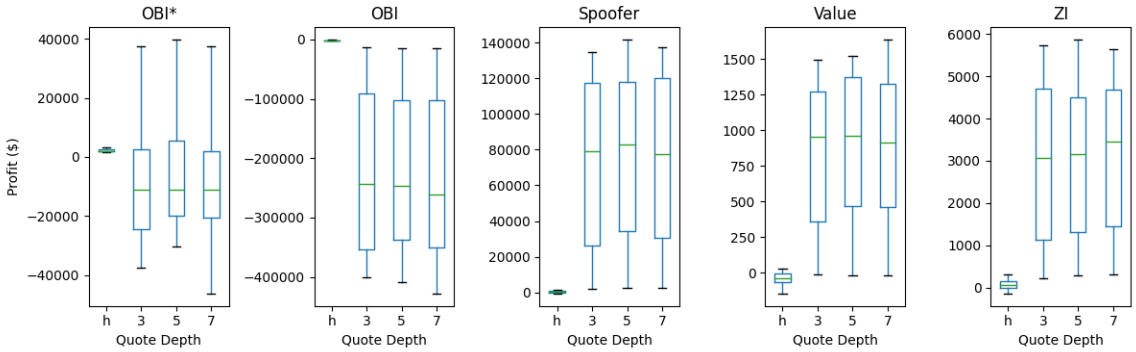


Figure 5.2: Boxplot showing effect of spoofer quote depth on profitability of each agent class. Includes all quote sizes. Quote size h indicates "honest" spoofer.

OBI agents had been exploiting *them* and now cannot.

- The spoofing strategy is clearly effective regardless of precise parameter selection, but lower quote sizes engender less response from the OBI agents, and hence produce lower spoofing profits.

Spoofing order quantities over 2,500 do not affect profitability and are omitted from this section. I hypothesize that the buy-sell ratios observed by the order book imbalance agents reach all meaningful reaction thresholds at that level of spoofing. The generated activity sequences for quantities over 2,500 are still used for spoofing detection.

5.5 Spoofing Data Synthesis

To train a supervised learning algorithm, I will require many example sequences of spoofing and non-spoofing behavior. Since I cannot obtain these from historical or live data, I use the previously described market simulation configurations to produce synthetic examples. With forty-eight different detailed spoofing approaches, I hope to achieve generality of detection. Within each simulated market day, I independently record for every order-related action:

- Initiating agent id,
- Initiating agent type,
- Simulated timestamp of the action,
- Action type (placing vs canceling a limit order),
- Order direction (buy vs sell),
- Limit price relative to current best bid or ask,
- Order quantity in shares,
- Is this action in support of a spoofing strategy,

where the recorded price is relative to the same side of the limit order book as the new activity. That is, on a limit order to buy, a relative price of zero equals the current best bid, positive prices indicate placement within the bid order book, and negative prices represent crossing the spread. Actions taken by the spoofing agent in "honest" mode are recorded as non-spoofing behavior.

After all simulations are completed, an action sequence is reconstructed for each agent for each market day. These action sequences are divided into non-overlapping

Table 5.2: Examples of spoofing and non-spoofing behavior. M^+: market buy, L^+: limit buy under best bid, C: cancel one L^+, M^-: market sell.

Possible spoofing sequence	M^+	L^+	L^+	L^+	M^-	C	C	C
Impossible spoofing sequence	L^+	C	M^-	L^+	L^+	M^+	C	C

subsequences of twenty actions each. Extra actions at the end of the day are discarded. This subsequence length was selected to balance the likelihood of an agent manifesting "spoofing" behavior within most windows against the desire to have many different behavior examples. Four features are retained in the training examples: action type, order direction, relative limit price, and order quantity. The remaining information is discarded. Examples are labeled True when they arise from a spoofing agent not in its "honest" mode, and False otherwise. I thus obtain 2,611,707 examples of sequenced agent behavior, of which 84,666 (approx. 3.2%) represent spoofing behavior and 2,527,041 non-spoofing behavior. As described, each training example has dimensionality $(20, 4)$.

5.6 Spoofing Detection Strategy

I have an input training data set of size $(2611707, 20, 4)$, with each labeled as spoofing or non-spoofing behavior. The detection task will be challenging for two primary reasons: the data set is highly imbalanced, because not spoofing is much more common than spoofing; and the temporal element of spoofing means that rearranging the same action primitives can render a spoofing explanation likely or impossible. This is illustrated in Table 5.2, where M^+ indicates a market buy order, L^+ indicates a limit buy order priced below the best bid, C indicates canceling one L^+ order, and M^- represents a market sell order.

This characteristic suggests an *activity recognition* problem and the need for a learning method that assigns value to the temporal ordering of the data. Fortunately

there is a rich body of research to draw from: Lara et al. describe the use of decision trees, instance-based learners like k-Nearest Neighbors, artificial neural networks, or ensembles of these [107]; and Wang et al. describe deep learning approaches including convolutional neural networks (CNN), autoencoders, recurrent neural networks (RNN), and hybrid models. [108]

Given the nature of the recognition problem, I selected Bidirectional Long Short-Term Memory (Bi-LSTM), a neural network architecture designed to capture complex temporal relationships in the training data, as the candidate most likely to succeed. For comparison and completeness, however, I also evaluated five other neural network architectures: standard LSTM, bidirectional and standard Gated Recurrent Units (Bi-GRU, GRU), a CNN convolving over time, and a simple feed-forward neural network (FFNN).

Before training, I preprocess the synthetic data. The training examples are shuffled and divided into a training (80%) and holdout test (20%) set. The training examples are further divided into a training (80%) and validation (20%) set. Each set is required to maintain the overall class balance of labels. The action type and order direction features are converted from boolean to 0-1 categorical with the usual mapping. The price and quantity features are normalized to $N(0, 1)$. To correct for the class imbalance, the positive class is oversampled to be equal in size to the negative class.

The candidate detection network contains a Bidirectional LSTM accepting input shape $(20, 4)$ with 64 hidden units and ReLu activation, connected to a dense layer with sigmoid activation and a single output neuron. The comparison networks are similarly structured. The network is trained with batch size 64 using the adam optimizer and binary crossentropy loss function. Given the large synthetic data set, a single training epoch proves sufficient for convergence. All detection experiments were conducted on a 2.4 GHz Quad-Core Intel Core i5 with 16 GB 2133 MHz LPDDR3

RAM. Training the network took approximately three minutes.

5.7 Spoofing Detection Results

Using the synthetic spoofing data and the given detection strategy, I empirically tested every combination of the available features against each of the proposed network architectures for a total of $15 \times 6 = 90$ detection experiments. For each experiment, I performed five-fold cross validation, training five separate detectors with different held out data, for a total of 450 trials. Here I analyze the results from two different perspectives: feature ablation and model selection.

5.7.1 Feature ablation

I evaluated the experimental results as a feature ablation study to determine which combinations of inputs were most informative of spoofing behavior. In Table 5.3 I report and interpret out-of-sample results for the Bi-LSTM architecture as a representative model.

I note that accuracy is not particularly informative for an extremely imbalanced data set like this one. Instead I focus on the raw confusion matrix, and measures derived from it that are appropriate for imbalanced boolean classification problems:

- The number of **true positive** predictions.
- The number of **false positive** predictions, also called Type I Error.
  - Innocent behavior flagged as spoofing.
- The number of **true negative** predictions.
- The number of **false negative** predictions, also called Type II Error.
  - Spoofing behavior that was overlooked.

Table 5.3: Feature ablation study of spoofing detection accuracy. Candidate features: order (D)irection, relative (P)rice, order (Q)uantity, action (T)ype.

Features	TP	FP	TN	FN	Precision	Recall	MCC
D	16,914	3,411	501,997	18	0.832	0.999	0.909
P	9,877	294,816	210,592	7,055	0.019	0.583	0.000
Q	1,396	0	505,409	15,537	1.000	0.082	0.283
T	16,914	556	504,852	18	0.968	0.999	0.983
DP	16,913	3,247	502,161	20	0.839	0.999	0.912
DQ	16,912	3,699	501,709	20	0.824	0.999	0.903
DT	16,928	88	505,321	4	0.995	1.000	0.997
PQ	1,421	0	505,408	15,511	1.000	0.084	0.285
PT	16,914	567	504,841	18	0.968	0.999	0.983
QT	16,907	554	504,854	25	0.969	0.999	0.983
DPQ	16,909	3,250	502,158	23	0.839	0.999	0.912
DPT	16,930	104	505,304	2	0.994	1.000	0.997
DQT	16,929	77	505,331	3	0.995	1.000	0.998
PQT	16,916	563	504,846	16	0.968	0.999	0.983
DPQT	16,930	77	505,332	2	0.995	1.000	0.998

- The **precision** or ratio of true positives to all positive predictions.
  - What portion of identified spoofing was actually spoofing?
- The **recall** or ratio of true positives to all positive examples.
  - What portion of all spoofing behavior was detected?
- The **Matthews Correlation Coefficient**, a measure of binary classification performance.

I rely on MCC as the primary measure of success, which has an interpretation consistent with the familiar Pearson correlation coefficient: zero indicates no correlation, -1 strong anti-correlation, and +1 strong correlation. Considering single feature models: action type produces a model with predictions very strongly correlated to ground truth (MCC 0.983); order direction is also very strongly correlated (MCC

0.909); order quantity is weakly correlated (MCC 0.283); and relative price is uncorrelated (MCC 0.000). Unsurprisingly, combining the two best features (direction and type) produces an even better model with MCC 0.997. Further adding quantity results in marginal improvement at best (MCC 0.998), and adding price does not improve the combined model. The ablation study would thus seem to indicate that a model with type (place/cancel order) and direction (buy/sell) contains the most useful information to detect spoofing action sequences.

5.7.2 Model selection

I also evaluated the six different model architectures presented in Section 5.6 to see if there were significant differences in performance when using the most successful combinations of features from the ablation study in the previous subsection. For this, I tested order direction alone, action type alone, and these two features together. The results are presented in Table 5.4.

The interpretation of the table columns is the same as given in Section 5.7.1. There is little difference among the tested models. The fully-connected feed-forward model is slightly less capable with single features. Perhaps most interestingly, a CNN with temporal convolution consistently achieves the same performance as much more complex architectures. This may be due to the length 20 windowing of the data, incorporating time within each example behavior. Regardless, based on this result, I conclude that given informative features like action type and order direction, most any modern ML architecture is sufficient for the problem, and I should select a simple model like temporally-convolved CNN.

5.8 Conclusion

I introduced many variations of a spoofing agent to a simulated agent-based stock market to study the impact of spoofing on agent profit, and more significantly to

Table 5.4: Architecture comparison of spoofing detection accuracy. Candidate features: order (D)irection, relative (P)rice, order (Q)uantity, action (T)ype.

Feat.	Arch.	TP	FP	TN	FN	Precision	Recall	MCC
D	FFFC	16,900	550	504,858	32	0.969	0.998	0.983
	CNN	16,907	423	504,985	25	0.976	0.999	0.987
	GRU	16,919	499	504,909	13	0.971	0.999	0.985
	LSTM	16,918	544	504,864	14	0.969	0.999	0.983
	Bi-GRU	16,920	537	504,871	12	0.969	0.999	0.984
	Bi-LSTM	16,914	556	504,852	18	0.968	0.999	0.983
T	FFFC	16,908	3,447	501,961	25	0.831	0.999	0.908
	CNN	16,913	3,165	502,243	19	0.842	0.999	0.914
	GRU	16,900	3,171	502,237	32	0.842	0.998	0.914
	LSTM	16,916	3,797	501,612	16	0.819	0.999	0.901
	Bi-GRU	16,910	3,237	502,171	22	0.839	0.999	0.913
	Bi-LSTM	16,914	3,411	501,997	18	0.832	0.999	0.909
DT	FFFC	16,926	125	505,283	6	0.993	1.000	0.996
	CNN	16,931	36	505,372	1	0.998	1.000	0.999
	GRU	16,932	69	505,339	0	0.996	1.000	0.998
	LSTM	16,928	97	505,311	4	0.994	1.000	0.997
	Bi-GRU	16,932	32	505,376	0	0.998	1.000	0.999
	Bi-LSTM	16,928	88	505,321	4	0.995	1.000	0.997

learn a spoofing detector. Based on Table 5.1 and Figure 5.1, it is clear the spoofing strategy profits across a variety of parameter selections and primarily takes money from the order book imbalance strategy. This is expected, as the OBI is a form of high-frequency trading that explicitly predicts short-term price moves by studying the volume balance of the order book – the exact feature the spoofer is manipulating. I note with interest that the spoofing agent causes the OBI traders to lose money well in excess of that captured by the spoofer. In their efforts to (legally) exploit the less intelligent agents, the OBI are now placing bad bets that feed money to those agents.

On the question of spoofing detection, I can interpret the results of Table 5.3. I used an oversampling technique to balance the training data set, so the detector does not "know" that spoofing should be rare. It must consider it just as likely as

non-spoofing behavior. So long as action type or order direction are available as features, the detector exhibits near-perfect recall, indicating that very few incidents of spoofing are not captured. When both are present, it also exhibits near-perfect precision, indicating low incidence of falsely identified spoofing, which would increase the workload of a human verifying suspect behavior. With a Matthews correlation greater than 0.9 in every case where at least one of action type or order direction are available, indicating very strong correlation between predictions and ground truth, I assert that the problem of spoofing detection given a large set of synthetic data from simulation is eminently tractable. The results of Chapter 6 will suggest the detector is not yet sufficiently general, but this work still offers a new direction for the regulation of AI in financial markets: the requirement that institutional participants and intermediaries monitor identifiable activity streams, using an ensemble of all available recognizers, in a best effort to identify and curtail non-normative trading practices.

In this chapter, I explored the promise of financial market simulation more deeply by synthesizing and then detecting activity sequences representing efforts to illegally spoof a financial market. This is a problem of significant current relevance, with $920 million in related fines in the year 2020 alone. I found that simulation is a safe, effective way to generate synthetic examples of spoofing behavior, and that spoofing can be accurately detected within action sequences of length twenty using four or fewer simple features. In the next chapter, I use the learned detector to train a normative trading agent that learns *not* to spoof the market in which it participates.

CHAPTER 6
NORMATIVE REINFORCEMENT: LEARNING NOT TO SPOOF

In Chapter 5, I introduced a method for training a neural spoofing detector using synthetic data from MAS of a stock market. In this chapter, I construct an RL-based stock market trader that has sufficient capacity to discover and adopt a spoofing-based strategy to boost its profitability, then leverage the detector as a normative guide during training to discourage the agent from adopting such a strategy. With this work, I hope to both illustrate a looming problem and provide a potential solution combining financial regulation and industry best practices. I provided an introductory overview to RL in Section 1.3.1, described background material related to Safe RL in Section 1.3.2, and discussed the particulars of financial market spoofing and the specific problem we are trying to solve in Section 1.3.3.

6.1 Related Work

In contrast with Safe RL (Section 1.3.2), which is concerned with the avoidance of critical failure cases like bankruptcy or physical collision, the pursuit of normative behavior in RL agents targets activity patterns humans will find acceptable and appropriate to a situation. Soares et al. discuss what they call the *value alignment problem*, which they define for an intelligent agent as learning and acting according to the preferences of its operators. [109] Informally, we might think of this as adding an implicit "without doing anything bad" to the end of any goal statement. Of course, this is an oversimplification: "bad" actions might *be* value-aligned if undertaken to avoid even worse outcomes.

What we might now call normative learning is substantially based on several foundational areas. Perhaps the earliest is Stuart Russell's 1998 introduction of *inverse*

reinforcement learning, under which one observes an agent's inputs, actions, and environment, and attempts to discern the agent's reward function. [110] In 1999, Ng et al. formalized a method for *reward shaping* intended to guide a learning algorithm to discovering an optimal policy more quickly through modification of the reward signal with additional feedback. [111] This concept was extended by Griffith et al. in 2013 to *policy shaping*, a form of interactive machine learning, where instead of external feedback numerically altering the reward signal, the agent attempts to directly learn the feedback as a separate policy. [112] The internal and external policies are combined as part of the decision-making process.

A significant line of work in normative learning has been undertaken in the field of robotics, especially navigation. Okal et al. have taken a Bayesian approach to socially normative navigation, training robotic agents to respect personal space, follow the flow of pedestrian traffic, and avoid breaking up conversational groups. [113] Coscia et al. extend the method using point-wise circular distributions based on prior observations to make long-term path predictions for the bicycles and pedestrians in the scene. [114] Shimosaka et al. considered the possibility of a normative driving agent that uses expert and inexpert demonstration to both learn that it should slow down at uncontrolled crossroads, and that inexperienced drivers may not do so. [115] Triebel et al. present an airport passenger assistance robot which learns to navigate a chaotic space, identify an appropriate spokesperson within a human group, and approach the spokesperson with an appropriate speed and direction. [116]

Another consistent research trajectory in normative learning has been explored in narrative intelligence. In 2015, Riedl. et al explored the idea of intelligent agents learning human values by reading stories, in particular crowdsourced stories concerning a specific environment and goal. [117] In addition to goal-oriented rewards, the agent receives additional rewards or penalties based on the frequency with which its activities appear in the corpus of human stories. Frazier et al. leverage the effectively

pre-labeled children's stories of *Goofus & Gallant* (GG), in which Gallant always does the "right" thing and Goofus the "wrong" thing, to learn a value-aligned prior. [118] This normative model is then successfully zero shot transferred to two new story data sets, with further improvement after limited fine tuning. Nahian et al. empirically evaluate methods of using the GG normative prior to induce a mixutre of normative and goal-oriented behavior in a series of interactive fiction game worlds, finding that an approach based on policy shaping and action reranking provides the most agent flexibility in selecting normative actions even when this lowers environmental rewards. [119] I explore both reward shaping and policy shaping (action reranking) as part of the current effort to teach an RL agent to profitably trade the stock market while avoiding a particular non-normative behavior: spoofing.

6.2 Approach

For this investigation, I assume a financial practitioner acting in good faith, wishing to leverage RL (specifically Q-learning) in the construction of an automated stock trading algorithm. The practitioner is aware, perhaps from Nick Bostrom's paperclip maximization thought experiment, that unconstrained objectives (e.g. "construct as many paperclips as possible") represent a risk to be thoughtfully managed, and wants to avoid non-normative behaviors like spoofing. [120]

For each set of experiments, I construct a tabular Q-learning agent to interact with a simulated multi-agent stock market in which other strategic traders are also participating. The agent is long-only, meaning it cannot sell stock it does not own, and is constrained to hold a set quantity of stock to avoid conflation of intelligent behavior with mere leverage. The base reward function for the agent includes the realized gain/loss in dollars when a stock position is closed, a small penalty for nonsensical actions, and a small transaction cost for any request sent to the stock exchange. The agent always observes the market for some *warmup* period before acting, during which

it learns what might be a "reasonable" price for the stock relative to the lowest and highest observed asking prices. The RL agents rely either on ϵ-greedy exploration, in which $\epsilon \in [0, 1]$ controls the proportion of random vs expected-optimal actions selected, or Boltzmann (softmax) exploration, in which actions are stochastically selected from a Boltzmann distribution [121]:

$$P(a|s) = \frac{e^{Q_{sa}}}{\sum_j e^{Q_{sj}}} \tag{6.1}$$

I make extensive use of spoofing detector Θ as learned in Chapter 5. The average activation level of Θ will be noted for each experimental agent, and in the normative experiments, the output of Θ will be used to influence the agent's actions. All price and quantity inputs to Θ are scaled according to the detector training data. Because Θ requires action sequences of length 20, and an agent may learn to trade less than 20 times per day, action sequences are carried forward across market days for the same instance of an agent, and the first 20 total actions of an agent are assumed to be normative.

The current state of the environment, including the agent, is represented as a set of boolean conditions drawn from the following candidates:

- **Holdings** (H): is the agent in a long position?
- **Holdings Advantage** (HA): did the agent have entry advantage when the current position was acquired?
- **Entry Advantage** (EA): is the estimated cost to enter a position less than the observed reasonable price?
- **Exit Advantage** (XA): does the estimated return from exiting a position exceed the agent's profit target?

- **Stop Loss** (SL): does the estimated loss from exiting a position exceed the agent's maximum permitted loss?
- **Open Orders** (OP): does the agent have open, unfilled orders?
- **Depth Orders** (DP): has the market moved since the agent placed an unfilled order?

When all candidates are included, the agent state space is therefore of size $2^7 = 128$. This state space is suitable only for a "ranging" market, but in simulation I can guarantee this condition.

Each experimental agent is given an action space drawn from the following:

- **Aggressive Entry** (AG): Aggressively enter a long position by purchasing a set quantity of stock at whatever price is immediately available.
- **Passive Entry** (PS): Passively enter a long position by offering to buy a set quantity of stock at slightly below the current price.
- **Exit** (EX): Exit a long position by selling whatever quantity of stock is owned.
- **Cancel** (CN): Cancel all unfilled open orders.
- **Update Passive Entry** (UP): Refresh a passive entry attempt by cancelling open orders and reissuing a passive entry order. This effectively combines action CN with PS.
- **Do Nothing** (DN): Do nothing.

With the exception of this experimental RL trading agent, the market environment and actors remain as described in Chapter 5. In the following sections, I build the argument that normative learning is both necessary and effective for this problem domain through a series of experiments within my simulated environment.

6.3 Fixed Policies

To ensure the possibility of a meaningful result in subsequent experiments, I first *design* a pair of fixed policies using the complete state-action space described in Section 6.2: π^s, which will attempt to profit by spoofing the simulated market, and π^h, which will attempt to profit without spoofing the simulated market. If both policies are profitable, then there is proof by demonstration that the RL agent *could* learn to profit either through spoofing or honest participation.

I construct honest policy $\pi^h(s) \rightarrow a$ as follows:

$$\pi^h(s) = \begin{cases} \text{AG} & \text{if } \neg \text{ H} \wedge \text{EA} \wedge \neg \text{ OP} \\ \text{EX} & \text{if H} \wedge \text{XA} \wedge \neg \text{ OP} \\ \text{CN} & \text{if OP} \\ \text{DN} & \text{otherwise} \end{cases} \tag{6.2}$$

I construct spoofing policy $\pi^s(s) \rightarrow a$ as follows:

$$\pi^s(s) = \begin{cases} \text{AG} & \text{if } \neg \text{ H} \wedge \text{EA} \wedge \neg \text{ OP} \\ \text{PS} & \text{if H} \wedge \neg \text{ (XA} \vee \text{SL)} \wedge \neg \text{ OP} \\ \text{EX} & \text{if H} \wedge \text{ (XA} \vee \text{SL)} \\ \text{CN} & \text{if } \neg \text{ H} \wedge \text{OP} \\ \text{UP} & \text{if H} \wedge \neg \text{ (XA} \vee \text{SL)} \wedge \text{DP} \\ \text{DN} & \text{otherwise} \end{cases} \tag{6.3}$$

As a baseline population for this experiment, I again instantiate 500 Zero Intelligence (ZI) agents, 500 Value agents, and 10 Order Book Imbalance (OBI) agents, all as described in Chapter 5. The exploitative high-frequency trader (OBI*) is positioned near the exchange at $21\mu s$ latency. The experimental fixed policy agent

Table 6.1: Average profit across 100 full market days comparing ad-hoc versus policy experimental agent in "honest" mode.

	Depth	Quantity	Exper.	OBI	OBI*	Value	ZI
ad hoc		*honest*	283	-1,811	2,158	-39	67
policy		*honest*	517	-1,654	1,985	-30	55

Table 6.2: Average profit across 100 full market days comparing experimental agent performance in "spoofing" mode while varying passive order depth. Ad-hoc agent from Table 5.1 for reference.

	Quantity	Depth	Exper.	OBI	OBI*	Value	ZI
ad hoc	2500	5	88,985	-267,769	-15,494	1,011	3,661
policy	2500	3	42,586	-140,664	-13,214	482	1,991
		4	40,485	-141,528	-17,832	477	2,033
		5	48,627	-158,853	-14,827	531	2,261
		6	50,003	-165,913	-16,635	553	2,368
		7	42,022	-139,738	-12,837	480	1,977

Table 6.3: Average profit across 100 full market days comparing experimental agent performance in "spoofing" mode while varying passive order quantity. Ad-hoc agent from Table 5.1 for reference.

	Depth	Quantity	Exper.	OBI	OBI*	Value	ZI
ad hoc	5	2500	88,985	-267,769	-15,494	1,011	3,661
policy	5	750	6,019	-37,563	17,828	122	507
		1000	19,377	-93,435	-2,937	425	1,224
		1250	34,703	-112,876	-5,108	407	1,565
		1500	43,216	-145,732	-13,354	479	2,084
		2000	47,622	-154,682	-13,421	520	2,195
		2500	48,627	-158,853	-14,827	531	2,261

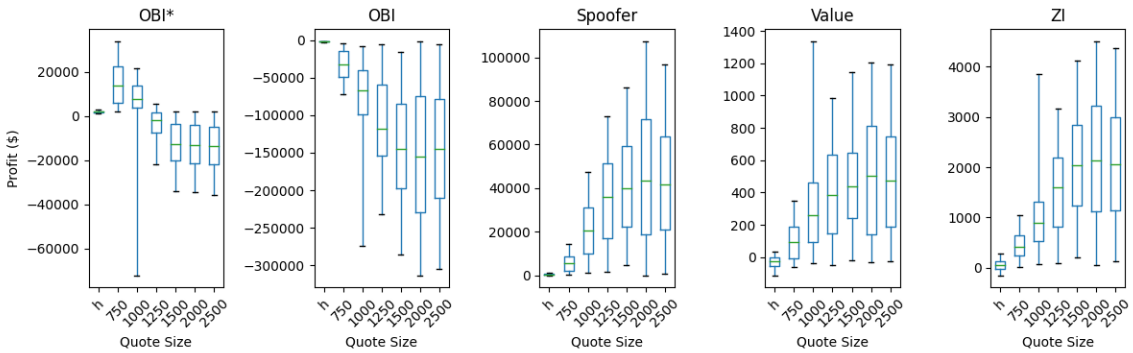


Figure 6.1: Boxplot showing effect of fixed policy agent quote size on profitability of each agent class. Includes all quote depths. Quote size h indicates "honest" policy agent.

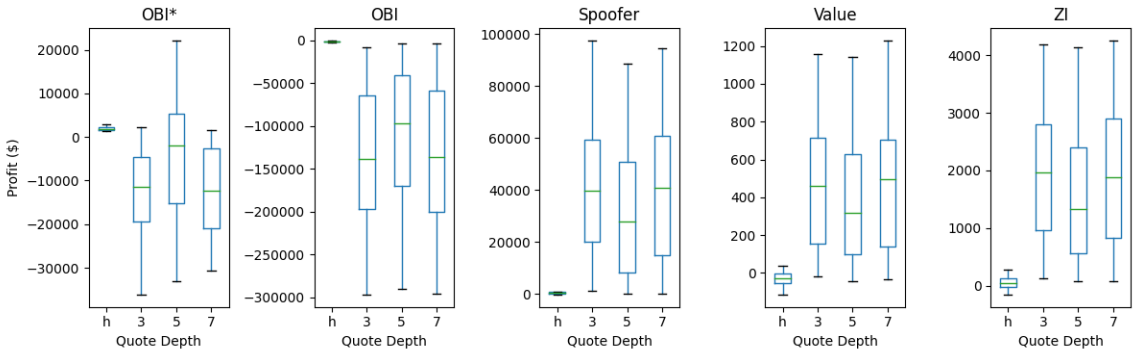


Figure 6.2: Boxplot showing effect of fixed policy agent quote depth on profitability of each agent class. Includes all quote sizes. Quote size h indicates "honest" policy agent.

replaces the ad-hoc experimental trader at a co-location latency of $33ns$. Note that the ad-hoc cases also appear in Table 5.1.

Tables 6.1, 6.2 and 6.3 summarize the average daily profit of each strategy across 100 simulated full market days. The returns of policy $\pi^h(s)$ approximately double those of the honest ad-hoc strategy, while the returns of the ad-hoc spoofing strategy approximately double those of policy $\pi^s(s)$. The positive or negative impact on the profitability of other agent classes proportionally follows. This is expected, as over the short term, stock trading is a zero-sum game. While the ad-hoc versus fixed policy outcomes differ by a scalar multiple from those presented in Section 5.4, they are of

the same order of magnitude in each case, and always positive for the experimental agent.

Figures 6.1 and 6.2 represent the box plot distribution of returns for each strategy as the quote size and depth of the fixed policy agent are varied. The findings are very similar to the ad-hoc agent from Chapter 5:

- Compared with policy $\pi^h(s)$, policy $\pi^s(s)$ harms the order-book aware OBI strategy and helps the other strategies.
- Passive quote depth has no discernible effect on the performance of any strategy.
- Up to a limit of about 1500 shares, positive and negative strategy outcomes scale proportionally with passive quote size.
- The exploitative OBI* agent is assisted by the spoofing behavior up to about quote size 1250, but then overwhelmed by it.

I also find that the average $\Theta(a_0 : a_{19})$ activation for fixed honest policy $\pi^h(s) = 0$ (order of $1e - 23$) and for fixed spoofing policy $\pi^s(s) = 0.842$.

Having demonstrated consistent profit for both $\pi^h(s)$ and $\pi^s(s)$, I conclude that it should be *possible* for a policy-based agent to learn to profit in this simulated market without adopting a non-normative spoofing behavior, but that adopting such a spoofing behavior would significantly enhance profitability.

6.4 Restricted Actions: Learning at a Disadvantage

I first attempt to enforce behavioral norms on the RL trading agent by restricting its action space, thus directly constraining its capacity to adopt non-normative behaviors. The restricted agent Q^R is given the complete state space described in Section 6.2 but limited action space {AG, EX, DN}. Given a more comprehensive state space, non-normative behaviors like insider trading would still be possible, but this state space

Table 6.4: Average profit across 100 full market days comparing restricted trained experimental agent approaches to exploration.

Exploration	Exper.	OBI	OBI*	Value	ZI
ϵ-greedy 1.0	467	-1,761	2,112	-30	57
ϵ-greedy step	382	-1,787	2,118	-27	54
ϵ-greedy decay	482	-1,825	2,138	-30	58
Boltzmann r-raw	-58	-1,791	2,117	-34	62
Boltzmann r-scaled	405	-1,776	2,123	-32	59

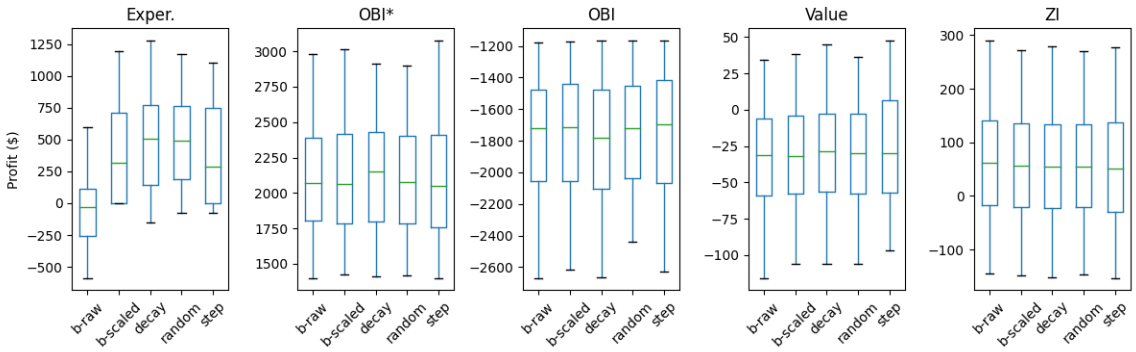


Figure 6.3: Boxplot showing effect of exploration method on profitability of each agent class.

is limited to internal market information. I therefore hypothesize that Q^R should not be able to learn technical non-normative behaviors like spoofing.

The agent population and configuration for this series of experiments follows the previous section except that the experimental agent now adopts online Q-learning to discover an optimal trading policy. I instantiate 10 differently-seeded agents in 10 separate simulation sequences. Each agent is permitted to train for 10 full market days and then evaluated (without updates) for another 10 full market days. All agents adopt either Boltzmann or ϵ-greedy exploration as described in Section 6.2. For ϵ-greedy, we consider three approaches: a fully random ($\epsilon = 1$), a scheduled step decline per day, or a more typical geometric decay. For Boltzmann, I consider raw and linearly scaled environmental rewards. At the first day of training, Q values are initialized to 0.0001 for action DN and to zero for all other actions, giving the agent

a slight initial bias to do nothing under Boltzmann exploration or in the $1 - \epsilon$ case for ϵ-greedy. In all cases, I apply a learning rate of $\alpha = \max(0.1, \frac{1}{c(s,a)})$, where c is a counter function.

In Table 6.4 and Figure 6.3 I report the average and distribution of profit across 100 market days for each trained agent configuration by exploration approach. Regardless of exact strategy, the ϵ-greedy traders and the scaled-r Boltzmann trader find a profitable policy in the mean, while the unscaled-r Boltzmann trader does not (quite). The exploration method selected by the RL trader makes little difference to the other agent classes. I note that the Boltzmann trader consistently converged in fewer market days of training, presumably by quickly learning to reject unrewarding actions. I postulate that the unscaled-r Boltzmann trader converged to a sub-optimal policy because the selection of actions in proportion to large scale rewards caused it to prefer any action that presented an early reward opportunity, without sufficient exploration of alternatives. For future experiments, I will prefer a scaled-r Boltzmann approach for its combination of fast convergence, near-optimal profit, and lack of significant downward deviations.

I additionally experimented with an actor-critic style target table, with varying update frequency, but found that it did not improve performance, and so do not include detailed results. I suspect that the non-stationary nature of our environment caused any artificial delay in utility estimation to be detrimental.

Finally, every action sequence produced by the experimental agents was evaluated using trained spoofing detector Θ from Chapter 5. As expected, the activation responses are uniformly low ($< 1e - 17$), confirming the hypothesis that the restricted agent lacked the capacity to spoof the simulated market. The restricted agent did consistently learn to turn a profit in the simulated market: in its scaled-r Boltzmann configuration, only 5% of market days ended in a loss, and the average loss on those days was only \$195. This confirms the hypothesis that a well-constructed RL trader

can learn to profit in the simulated market without spoofing.

6.5 Normative Alignment: Learning Not to Spoof

In prior sections, I laid out several important preliminaries: I constructed both an honest and spoofing policy that earn profits in the simulated market. I showed that normative behavior can be profitable, but non-normative behavior is more profitable. I successfully *learned* an honest policy, constrained through action space restrictions, that performs similarly to the designed honest policy. Each policy has demonstrated a reliable distribution of daily profit and has activated spoofing detector Θ to the expected degree. I must now attempt to answer the question:

> Can the experimental RL agent be given an unrestricted action space without learning to spoof the simulated market?

For this series of experiments, I remove the action space restrictions on the experimental agent. I would expect this to present a problem similar to the evocative example given in the ubiquitous Russell and Norvig textbook: "Maximize dirt collected" is an acceptable goal for an intelligent vacuum until it gains the "dump dirt" action, at which point the goal must be refined to "maximize floor cleanliness" to avoid an unwanted exploit. [41] I therefore also augment the training process with two of the approaches described in Section 6.1: reward shaping and policy shaping (with action reranking). I hypothesize that normatively-constrained trader Q^N will use the additional flexibility of its action space to earn profit beyond restricted agent Q^R, but with lower Θ scores than the fixed spoofing policy due to the inclusion of a normative prior.

Both normative training scenarios require that trained spoofing detector Θ be incorporated directly into the experimental agent. Ideally, I could evaluate individual proposed or selected actions for normativity, but as established in Figure 5.2, all of

the agent's individual actions are normative, and it is only sequences of actions that take on characteristics of non-normativity. In particular, Θ requires action sequences of exactly length 20 to make an evaluation, and was trained on data arising from the exchange agent. I therefore maintain a length 20 history only of those actions that would be visible to the exchange: this means that one agent policy action may translate to zero or many exchange-visible actions for determining normativity.

For the reward shaping approach, I stochastically select an action from the Boltzmann distribution as described in Section 6.2. When the agent is to receive an immediate, positive reward for closing a profitable position, I pass the action history through Θ to obtain an activation level: $\Theta(a_0 : a_{19}) \in [0, 1]$. The activation level is high for spoofing behavior, which I want to discourage, so I transform the reward:

$$r' = r \times [1 - \Theta(a_0 : a_{19})] \tag{6.4}$$

For the action reranking approach, I transform the Boltzmann-derived probability vector prior to action selection. Each time the agent must act, I tentatively add each potential action to the recent history and pass the "proposed" histories through Θ to estimate the normativity of each action given the current history. I use this to update the probability of selecting each action:

$$P(a|s) = \frac{e^{Q_{sa} \cdot [1 - \Theta(a, a_0 : a_{18})]}}{\sum_j e^{Q_{sj} \cdot [1 - \Theta(j, a_0 : a_{18})]}} \tag{6.5}$$

In Table 6.5 and Figure 6.4, I report the average and distribution of profit across 100 market days for each trained experimental agent configuration using passive depth 5 and quantity 2500, comparing the four normative approaches with three of the earlier configurations for reference. The table also shows the average detector activation for the agent in the "Norm." column, and Figure 6.5 shows the activation distribution.

Examining Table 6.5 suggests that all four approaches to normative guidance

Table 6.5: Average profit across 100 full market days comparing trained experimental agent approaches to normative guidance: action reranking or reward shaping with unscaled or linearly scaled environmental rewards.

Config.	Scaled?	Norm.	Exper.	OBI	OBI*	Value	ZI
Fixed Honest	-	0.000	517	-1,654	1,985	-30	55
Fixed Spoofing	-	0.842	48,627	-158,853	-14,827	531	2,261
Restricted	Y	0.000	405	-1,776	2,123	-32	59
Reranking	N	0.213	67,615	-376,759	-73,187	1,386	5,407
Reranking	Y	0.043	21,871	-83,964	-12,070	227	1,263
Shaping	N	0.492	93,863	-636,593	-171,145	2,734	8,880
Shaping	Y	0.072	19,987	-86,130	-10,882	263	1,268

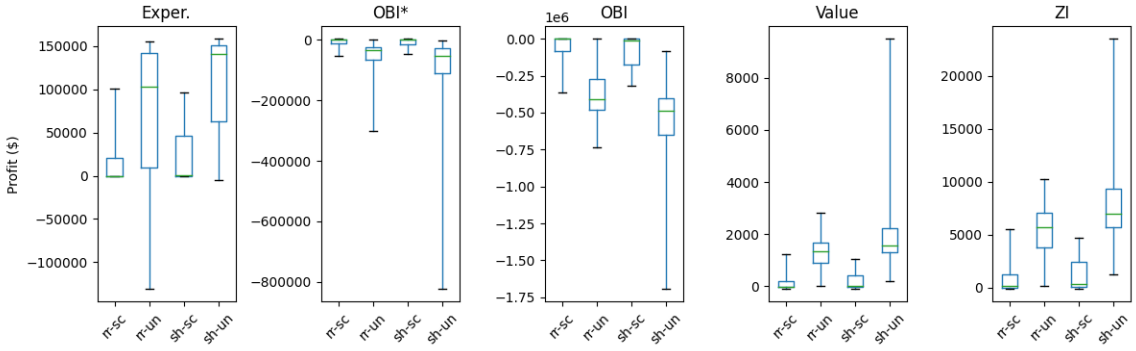


Figure 6.4: Boxplot showing effect of normative guidance method on profitability of each agent class.

reduce the average activation of the spoofing detector, however one may rightly be suspicious that both configurations with unscaled environmental rewards achieve not only greater profit than the "honest" traders, but greater than the fixed spoofing policy! The configurations with scaled environmental rewards are more reasonable, at least being lower than the known spoofing strategy.

The distribution of profits shown in Figure 6.4 is more revealing. Both unscaled approaches to normative guidance earn spoofing-level profits most of the time, with occasional catastrophic failures for the reranking approach. Both scaled approaches seem promising, rarely or never losing money, but also rarely exceeding expected levels of profit from normative behavior.

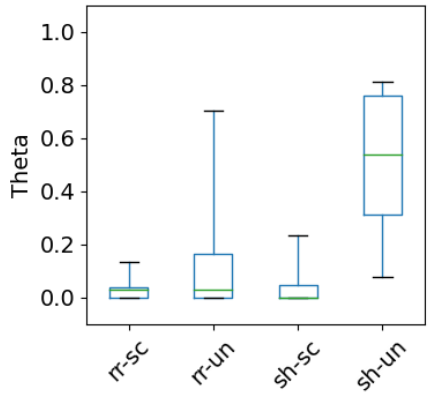


Figure 6.5: Boxplot showing effect of normative guidance method on normativity of each agent class.

The scaled action reranking approach has by far the lowest overall detector activation at $\Theta = 0.043$ and a tight distribution on the profit boxplot, demonstrating the best overall ability to guide the experimental trading agent to normative behavior, even at the expense of increased profit. However, the long upward shadow for scaled action reranking profit in Figure 6.4 combined with the *short* upward shadow for scaled action reranking Θ in Figure 6.5 suggests that some agents are evading their normative guidance.

Extracting individual agents trained under the same approach but in different stochastic market conditions illuminates the potential issue. Considering both profit and detector activation, there are two clusters of scaled action reranking agents with different characteristics:

$$C_1 : p \in [-200, 800], \Theta \in [0, 0.040]$$

$$C_2 : p \in [67\text{k}, 109\text{k}], \Theta \in [0, 0.133]$$

The larger cluster C_1 has characteristics similar to known honest agents. The smaller cluster C_2 has profits similar to known spoofing agents, but with far lower Θ activation levels. With a complex, continuous state-action space, I might assume the C_2 agents

have simply learned to cleverly trade without spoofing, but the seven boolean state features in my current environment do not permit much freedom. It is more likely that these agents have learned to evade the spoofing detector.

Detailed activity traces were captured during the simulation of all experimental agents. At the time of each RL agent decision, a record is made of the boolean state components, the selected action, the Θ activation and selection likelihood of all actions, the immediate reward, and the estimated portfolio value change, if any. Further analysis of these traces results in several interesting observations:

1. Profit for C_1 and C_2 agents are similar during training, suggesting that specific experiences separate development of the agent clusters, rather than overall training success.

2. During training, C_2 agents experience state combination {H 1, XA 1, OP 1} twice as often as C_1 agents, effectively receiving more opportunities to discover that open orders drive higher rewards. They also stochastically select action EX 50% more often in response, clearly learning to close their position when this occurs.

3. During training, C_1 agents enter a position with entry advantage (EA $\rightarrow$ EN) more than twice as often as C_2 agents. This may indicate that C_1 agents are experiencing better early buying opportunities.

4. During training, all agents select actions that register $\Theta = 1.0$ about 67 times per day, showing that the normative guidance does not completely prohibit taking known non-normative actions. Every time this occurs in a trace, it is because *all* actions have estimated $\Theta = 1.0$.

5. During test evaluation, C_2 agents close a profitable position while having an open order at the market an average of 189 times per day, while C_1 agents

never do this. This confirms that C_2 agents have almost certainly learned to spoof (despite low Θ levels) and that C_1 agents have not.

Taken together, these observations reveal promising avenues for future work.

Observations 2 and 3 suggest that eventual assignment to cluster C_1 (apparently honest) or C_2 (apparently spoofing) may depend substantially on the stochastic market properties experienced by the agent during its early training period. Agents that experience frequent, early "fair" prices to buy in are more likely to end up in the apparently honest cluster, compared with agents that do not. This can be addressed with a less artificial state space for the RL trader. In particular, calculating entry advantage on a rolling basis, or providing generally more granular features, could allow more flexibility to find apparently honest strategies on market days when prices rarely fall back to the midpoint of the warm-up period's price range.

Observation 4 suggests a limitation with the current mechanism to evaluate normative trajectory through recent action history. These occurrences represent periods where the agent's recent history, exclusive of the proposed action, is so non-normative that no single selected action can lower the detector activation, effectively removing any normative guidance for that selection. To address this in future work, I can attempt to apply both policy shaping and reward shaping together, so when a non-normative action is selected despite action reranking, the shaped reward penalty can lower the estimated Q value to discourage repetition of the behavior. It may also be possible to make the normative guidance more responsive to the most recent actions, or to adjust its threshold, so it will be less likely to rate all proposed actions as equally non-normative.

Overall, cluster C_1 (apparently honest) is roughly three times the size of cluster C_2 (apparently spoofing). Given this, and the low Θ activations shown in Table 6.5 and Figure 6.5, I interpret the current experimental results as: the normative guidance works about 3 out of 4 times. While imperfect, I accept this as a promising start.

6.6 Conclusion

I have introduced and tested honest and spoofing fixed policy trading agents, an honest (action restricted) Q-learning trading agent, and an unrestricted Q-trading agent with two forms of normative guidance. The fixed policy traders show similar performance to the ad-hoc agents introduced in Chapter 5 both in terms of profitability and normativity and, as expected, the action-restricted agent lacked the ability to spoof the simulated market under any exploration regime.

As discussed in Section 6.5, the experiments with normative guidance produced promising but mixed results: for most random simulation starts, the unrestricted agent appeared not to pursue spoofing strategies and displayed profits and norm values similar to the honest fixed policy and restricted agent. However, about one quarter of the unrestricted agents earned *far higher* profit while still registering essentially zero on the spoofing detector, and exploration of their action traces confirms the development of apparent spoofing behaviors that evade normative guidance.

I consider this a successful result: In each case, the normative guidance prevented development of behaviors that would trigger the spoofing detector. With respect to the edge case failures, the previously unexplored research problem would appear to be reduced to some combination of "build a better spoofing detector" and "allow an apparently honest RL agent to better adapt to a wide variety of stochastic market conditions". Specific suggestions for the latter were presented in Section 6.5, while the former can be approached in at least two ways: add variety to the designed spoofing sequences for training data generation, or directly feed any apparent "stealth spoofing" behaviors as additional training data to the detector.

Reflecting on Chapters 5 and 6 together, this work demonstrates that unintentional legal and regulatory violations by intelligent algorithms are a serious concern requiring industry and regulator attention. The spoofing detector presented here can

be improved and included as one component of an ensemble alongside decision tree (or random forest) models, hidden Markov models, and others, with new models added to the ensemble as new non-normative behaviors are identified over time. Regulatory agencies could recommend or mandate that firms with a brokerage or trading desk deploy the ensemble as a best faith effort to detect and discourage market manipulation at a position in the order flow where identifying information is still present. The use of market simulation to understand how other traders may react to an agent's behavior can ease the development component models for the ensemble.

In this chapter, I developed a method to inject normative guidance information to an RL trading agent in simulation and observed the effects on its behavior. In the final chapter, I reflect on the overall scope of the dissertation, highlight some contributions, and suggest additional avenues for future work.

CHAPTER 7
CONCLUSIONS AND CONTRIBUTIONS

In this chapter I discuss how the presented work supports my thesis, consider the contributions made in this dissertation, and propose several avenues of future work. I began this multi-domain empirical investigation with the assertion that:

> ***Multi-agent simulation is an effective tool to solve problems in responsible machine learning.***

I applied MAS to two primary problem domains within responsible ML: user data privacy preservation and normative reinforcement learning. In Chapter 2, I introduced a novel MAS for financial markets with important new capabilities and used it to study the impact high-frequency trading agents may have on other market participants. In Chapters 3 and 4, I extended ABIDES to the domain of federated learning and presented a novel privacy-preserving protocol which prevents user data from being exposed by semi-honest participant collusion during collaborative learning. In Chapters 5 and 6, I used ABIDES to generate a corpus of spoofing examples, with which a detector was trained to serve as normative guidance for an RL trading agent that should not learn to spoof the stock market.

The application of MAS to responsible ML was not without challenges. When simulating private federated learning, I encountered substantial cryptographic overhead in both time and space complexity. This limited the maximum participant count on several occasions, and was difficult to circumvent given the single-threaded nature of my simulation. Despite this, MAS allowed the development, iteration, and realistic evaluation of a novel privacy extension, that would otherwise require worldwide cloud resources, on a single local CPU. The training of a spoofing detector presented an

obvious problem: no real spoofing data was available, so the effort would be reliant on the verisimilitude of simulated spoofing strategies for the efficacy of the detector. Applying the detector as normative guidance via action reranking proved very slow, as it involved a trading agent with potential action frequency 1μs querying a neural network at least six times per required decision. Nevertheless, it was demonstrated that both a normative and non-normative agent could profit in the simulated market, and to show that an agent with the capacity to spoof could be discouraged from doing so without artificially restricting actions that have normative application.

The success of each of those efforts, leading to the contributions summarized in the next section, effectively validate the thesis.

7.1 Contributions

The work presented in this dissertation makes a number of concrete and novel contributions to the research space of responsible machine learning.

- ABIDES, an open source multi-domain MAS with important new capabilities, plus built-in agents and functionality, for complex electronic trading markets, privacy-preserving federated learning (PPFL), and safe evaluation of the (non-) normativity of learning agents.

- A method for inexpensive empirical analysis of PPFL in simulation, with the ability to vary latency, participant count, privacy loss parameter ϵ, and participant connectivity, and observe the effect on protocol accuracy, temporal load, and privacy.

- An improved PPFL mechanism based on 1-2 oblivious transfer that protects sensitive user data even when all other collaborative learning participants collude to reveal the data.

- Construction and evaluation of a financial market spoofing detector from synthetic behavior examples.
- An approach to reward shaping and policy shaping (action reranking) for complex financial environments in which only sequences of actions may be judged as non-normative, requiring novel use of forward-simulated action histories as guidance input.
- Empirical analysis of a normative RL trading agent learning not to spoof a financial market.

7.2 Future Work

In this concluding section, I describe promising extensions or new applications of the work presented in this dissertation.

7.2.1 AI Market Regulation

Given the recent enforcement actions discussed in Sections 1.3.3 and 5.1 [35, 36, 39, 105], the necessity to regulate the use of automated trading agents in financial markets has become clear. Current laws incorporating the intent of a trader or programmer likely apply even to automated agents deliberately designed to contravene statute or guidance, but it appears this is not the case for inadvertent abuse arising from a carelessly trained agent without proper constraints. The detection and normative trading work in Chapters 5 and 6 can be reinforced with additional approaches to spoofing and other prohibited activities, refined to more reliably inhibit non-normative behavior, and become the basis of "best practices" for the financial sector to follow when training automated traders. Regulatory bodies like CFTC, FINRA, and SEC can encourage adoption of such best practices through a safe harbor clause, offering indemnity against accidental violation if accepted best practices have been followed.

Because the presented detector uses only order information that would be visible to the first upstream service provider, firms could also better monitor their clients for prohibited activities.

7.2.2 Responsible Simulation of AI Attacks

As complex intelligent systems are given greater responsibility, we can expect consequential failures to arise from attacks against, or employing, intelligent agents. Understanding and mitigating these risks will be a high priority for academics and industry practitioners in the medium term. The work presented in Chapter 4 can be extended from attacks against federated learning to a range of other questions. Can we use simulation as a form of data augmentation to guard against inappropriate behavior during "black swan" events? Could a malign intelligent agent attack the stock market by learning to place orders that maximize volatility or cause an outright market crash? These and similar questions would be both interesting and possible to explore using the developed ABIDES platform.

7.2.3 High Performance Computing for Secure ML Collaboration

The nascent field of federated learning [16, 19, 13, 12] is attracting increasing attention for both its promise to allow more accurate learned inferences from distributed devices while maintaining user data privacy, and for the potential of safe cooperation across departments, firms, or agencies to learn shared models for the common good. With additional performance improvements, the presented work in accurate local simulation of federated learning protocols can boost privacy and research productivity in this emerging area. In particular, the current privacy protocol comes at a cubic computational cost per client, and the pure Python simulation could be much faster. Theoretic and practical advances on either front can encourage better, more resilient AI "in the wild" by lowering barriers to safe, efficient collaboration.

REFERENCES

[1] J. Angwin, J. Larson, S. Mattu, and L. Kirchner, "Machine bias: There's software used across the country to predict future criminals and it's biased against blacks.," *URL https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing*, 2016.

[2] ——, "How we analyzed the compas recidivism algorithm," *URL https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm*, 2016.

[3] A. Chouldechova, "Fair prediction with disparate impact: A study of bias in recidivism prediction instruments," *Big data*, vol. 5, no. 2, pp. 153–163, 2017.

[4] J. Buolamwini and T. Gebru, "Gender shades: Intersectional accuracy disparities in commercial gender classification," in *Conference on fairness, accountability and transparency*, 2018, pp. 77–91.

[5] S. Verma and J. Rubin, "Fairness definitions explained," in *2018 IEEE/ACM International Workshop on Software Fairness (FairWare)*, IEEE, 2018, pp. 1–7.

[6] S. Corbett-Davies and S. Goel, "The measure and mismeasure of fairness: A critical review of fair machine learning," *arXiv preprint arXiv:1808.00023*, 2018.

[7] White House, "Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy," *White House, Washington, DC*, pp. 1–62, 2012.

[8] R. Sen and S. Borle, "Estimating the contextual risk of data breach: An empirical approach," *Journal of Management Information Systems*, vol. 32, no. 2, pp. 314–341, 2015.

[9] Electronic Privacy Information Center, "Equifax data breach," *https://www.epic.org/privacy/data-breach/equifax/*, 2018.

[10] Kromtech Security Center, "Virtual keyboard developer leaked 31 million of client records," *https://kromtech.com/blog/security-center/virtual-keyboard-developer-leaked-31-million-of-client-records*, 2018.

[11] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, " communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, PMLR, 2017, pp. 1273–1282.

[12] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 739–753.

[13] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2017, pp. 3–18.

[14] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our data, ourselves: Privacy via distributed noise generation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2006, pp. 486–503.

[15] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, 1987, pp. 218–229.

[16] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2017, pp. 1175–1191.

[17] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, and B. Thorne, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," *arXiv preprint arXiv:1711.10677*, 2017.

[18] V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," *arXiv preprint arXiv:1705.10467*, 2017.

[19] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, *et al.*, "Advances and open problems in federated learning," *arXiv preprint arXiv:1912.04977*, 2019.

[20] J. Kilian, "Founding crytpography on oblivious transfer," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, 1988, pp. 20–31.

[21] R. Bellman, "A markov decision process," *Journal of Mathematical Mechanics*, 1957.

[22] S. P. Meyn, "The policy iteration algorithm for average reward markov decision processes with general state space," *IEEE Transactions on Automatic Control*, vol. 42, no. 12, pp. 1663–1680, 1997.

[23] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[24] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[25] M. Minsky, E. Feigenbaum, and J. Feldman, "Computers and thought," 1963.

[26] S. Thrun, "Exploration in active learning," *Handbook of Brain Science and Neural Networks*, pp. 381–384, 1995.

[27] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[28] J. Garcia and F. Fernandez, "A comprehensive survey on safe reinforcement learning," *J. Mach. Learn. Res.*, vol. 16, pp. 1437–1480, 2015.

[29] M. Hutter, "Self-optimizing and pareto-optimal policies in general environments based on bayes-mixtures," *CoRR*, vol. cs.AI/0204040, 2002.

[30] D. Ryabko and M. Hutter, "On the possibility of learning in reactive environments with arbitrary dependence," *CoRR*, vol. abs/0810.5636, 2008. arXiv: 0810.5636.

[31] A. Gosavi, "Reinforcement learning for model building and variance-penalized control," in *Winter Simulation Conference*, ser. WSC '09, Austin, Texas: Winter Simulation Conference, 2009, pp. 373–379, ISBN: 9781424457717.

[32] P. Geibel and F. Wysotzki, "Risk-sensitive reinforcement learning applied to control under constraints," *Journal of Artificial Intelligence Research*, vol. 24, pp. 81–108, 2005.

[33] Commodity Exchange Act: Antidisruptive Practices Authority, *7 U.S.C. § 6(c)(a)(5)(C). 2021*,

[34] Commodity Futures Trading Commission, "Fact sheet: Interpretive guidance and policy statement on disruptive practices," *https://www.cftc.gov*, 2021.

[35] E. J. Lee, K. S. Eom, and K. S. Park, "Microstructure-based manipulation: Strategic behavior and performance of spoofing traders," *Journal of Financial Markets*, vol. 16, no. 2, pp. 227–252, 2013.

[36] CFA Institute, "Global market sentiment survey," *URL https://www.cfainstitute.org/en/research/survey-reports/global-market-sentiment-survey-2015*, 2015.

[37] Dodd-Frank Wall Street Reform and Consumer Protection Act of 2010, *URL https://www.govinfo.gov/content/pkg/PLAW-111publ203/pdf/PLAW-111publ203.pdf*,

[38] T. J. Putniņš, "Market manipulation: A survey," *Journal of Economic Surveys*, vol. 26, no. 5, pp. 952–967, 2012.

[39] Commodity Futures Trading Commission, "CFTC division of enforcement issues annual report," *URL https://www.cftc.gov/PressRoom/PressReleases/8323-20*, 2020.

[40] ——, "CFTC orders jpmorgan to pay record $920 million for spoofing and manipulation," *URL https://www.cftc.gov/PressRoom/PressReleases/8260-20*, 2020.

[41] S. Russell and P. Norvig, "Artificial intelligence: A modern approach, fourth edition," 2021.

[42] M. Wooldridge, *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[43] E. Bonabeau, "Agent-based modeling: Methods and techniques for simulating human systems," *Proceedings of the national academy of sciences*, vol. 99, no. suppl 3, pp. 7280–7287, 2002.

[44] P. Bratley, B. L. Fox, and L. E. Schrage, *A guide to simulation*. Springer Science & Business Media, 2011.

[45] C. M. Macal and M. J. North, "Agent-based modeling and simulation," in *Proceedings of the 2009 Winter Simulation Conference (WSC)*, Dec. 2009, pp. 86–98.

[46] C. Macal and M. North, "Introductory tutorial: Agent-based modeling and simulation," in *Proceedings of the Winter Simulation Conference 2011*, IEEE, 2011, pp. 1456–1469.

[47] R. Axelrod, "Advancing the art of simulation in the social sciences," in *Simulating social phenomena*, Springer, 1997, pp. 21–40.

[48] L. Tesfatsion, "Agent-based computational economics: Growing economies from the bottom up," *Artificial life*, vol. 8, no. 1, pp. 55–82, 2002.

[49] A. Negahban and L. Yilmaz, "Agent-based simulation applications in marketing research: An integrated review," *Journal of Simulation*, vol. 8, no. 2, pp. 129–142, 2014.

[50] G. Weiss, *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 1999.

[51] P.-O. Siebers and U. Aickelin, "Introduction to multi-agent simulation," in *Encyclopedia of decision making and decision support technologies*, IGI Global, 2008, pp. 554–564.

[52] M. Niazi and A. Hussain, "Agent-based computing from multi-agent systems to agent-based models: A visual survey," *Scientometrics*, vol. 89, no. 2, pp. 479–499, 2011.

[53] D. Friedman, "The double auction market institution: A survey," *The double auction market: Institutions, theories, and evidence*, vol. 14, pp. 3–25, 1993.

[54] D. K. Gode and S. Sunder, "Allocative efficiency of markets with zero-intelligence traders: Market as a partial substitute for individual rationality," *Journal of political economy*, vol. 101, no. 1, pp. 119–137, 1993.

[55] D. Cliff and J. Bruten, "Zero is not enough: On the lower limit of agent intelligence for continuous double auction markets," *Proceedings of the Symposium on Computation in Economics, Finance and Engineering: Economic Systems*, 1998.

[56] S. Gjerstad and J. Dickhaut, "Price formation in double auctions," *Games and economic behavior*, vol. 22, no. 1, pp. 1–29, 1998.

[57] E. Wah, M. Wright, and M. P. Wellman, "Welfare effects of market making in continuous double auctions," *Journal of Artificial Intelligence Research*, vol. 59, pp. 613–650, 2017.

[58] S. Vyetrenko, D. Byrd, N. Petosa, M. Mahfouz, D. Dervovic, M. Veloso, and T. H. Balch, "Get real: Realism metrics for robust limit order book market simulations," *2020 ACM International Conference on AI in Finance*, 2020.

[59] T. H. Balch, M. Mahfouz, J. Lockhart, M. Hybinette, and D. Byrd, "How to evaluate trading strategies: Single agent market replay or multiple agent interactive simulation?" *ICML 2019 Workshop on AI in Finance*, 2019.

[60] M. Mahfouz, A. Filos, C. Chtourou, J. Lockhart, S. Assefa, M. Veloso, D. Mandic, and T. Balch, "On the importance of opponent modeling in auction markets," *arXiv preprint arXiv:1911.12816*, 2019.

[61] M. Karpe, J. Fang, Z. Ma, and C. Wang, "Multi-agent reinforcement learning in a realistic limit order book market simulation," *arXiv preprint arXiv:2006.05574*, 2020.

[62] D. Freidman, "The double auction market institution: A survey," *The Double Auction Market Institutions, Theories and Evidence, Addison Wesley*, 1993.

[63] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *Journal of political economy*, vol. 81, no. 3, pp. 637–654, 1973.

[64] R. C. Merton, "Theory of rational option pricing," *The Bell Journal of economics and management science*, pp. 141–183, 1973.

[65] R. K. Nance, "On time flow mechanisms for discrete system simulation," *Management Science*, vol. 18, no. 1, pp. 59–73, 1971.

[66] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[67] D. R. Jefferson and H. Sowizral, "Fast concurrent simulation using the time warp mechanism," in *Distributed Simulation 1985*, ser. Simulation Council Proceedings 2, vol. 15, TUDD-IfN-TK: Li-Ord.Le: Society for Computer Simulation (SCS), 1985, pp. 63–69.

[68] K. M. Chandy and J. Misra, "Asynchronous distributed simulation via a sequence of parallel computations," *Communications of the ACM*, vol. 24, no. 4, pp. 198–205, Apr. 1981.

[69] C. M. Macal and M. J. North, "Agent-based modeling and simulation," in *Proceedings of the 2009 Winter Simulation Conference (WSC)*, Dec. 2009, pp. 86–98.

[70] N. Minar, R. Burkhart, C. Langton, and M. Askenazi, *The swarm simulation system, a toolkit for building multi-agent simulations*, 1996.

[71] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "MASON: A multiagent simulation environment," *SIMULATION*, vol. 81, pp. 517–527, 2005.

[72] B. LeBaron, "A builder's guide to agent-based financial markets," *Quantitative Finance*, vol. 1, no. 2, pp. 254–261, 2001. eprint: https://doi.org/10.1088/1469-7688/1/2/307.

[73] B. I. Jacobs, K. N. Levy, and H. M. Markowitz, "Financial market simulation," *The Journal of Portfolio Management*, vol. 30, no. 5, pp. 142–152, 2004. eprint: https://jpm.pm-research.com/content/30/5/142.full.pdf.

[74] M. Levy, H. Levy, and S. Solomon, "A microscopic model of the stock market: Cycles, booms, and crashes," *Economics Letters*, vol. 45, no. 1, pp. 103–111, 1994.

[75] B. I. Jacobs, K. N. Levy, and H. M. Markowitz, "Simulating security markets in dynamic and equilibrium modes," *Financial Analysts Journal*, vol. 66, no. 5, pp. 42–53, 2010. eprint: https://doi.org/10.2469/faj.v66.n5.7.

[76] J. Wang, V. George, T. Balch, and M. Hybinette, "Stockyard: A discrete event-based stock market exchange simulator," in *2017 Winter Simulation Conference (WSC)*, IEEE, 2017, pp. 1193–1203.

[77] X. Wang and M. P. Wellman, "Spoofing the limit order book: An agent-based model," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 651–659.

[78] M. P. Wellman, "Methods for empirical game-theoretic analysis," in *AAAI*, 2006, pp. 1552–1556.

[79] M. Zook and M. H. Grote, "The microgeographies of global finance: High-frequency trading and the construction of information inequality," *Environment and Planning A: Economy and Space*, vol. 49, no. 1, pp. 121–140, 2017.

[80] J. Banks, *Handbook of simulation: principles, methodology, advances, applications, and practice*. John Wiley & Sons, 1998.

[81] NASDAQ OMX Group, *NASDAQ TotalView - ITCH 5.0*, URL http://www.nasdaqtrader.com/content/technicalsupport/specifications/dataproducts/NQTVITCHSpecification.pdf, Accessed: 2018-10-25.

[82] ——, *O*U*C*H Version 4.2*, URL http://www.nasdaqtrader.com/content/technicalsupport/specifications/TradingProducts/OUCH4.2.pdf, Accessed: 2018-10-25.

[83] T. E. Oliphant, *A guide to NumPy*. Trelgol Publishing USA, 2006, vol. 1.

[84] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, Austin, TX, vol. 445, 2010, pp. 51–56.

[85] S. Gjerstad, "The competitive market paradox," *Journal of Economic Dynamics and Control*, vol. 31, no. 5, pp. 1753–1780, 2007.

[86] L. Harris, "Optimal dynamic order submission strategies in some stylized trading problems," *Financial Markets, Institutions & Instruments*, vol. 7, no. 2, pp. 1–76, 1998.

[87] D. Byrd, "Explaining agent-based financial market simulation," *arXiv preprint arXiv:1909.11650*, 2019.

[88] F. McSherry and K. Talwar, "Mechanism design via differential privacy.," in *FOCS*, vol. 7, 2007, pp. 94–103.

[89] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, 1987, pp. 218–229.

[90] B. Jayaraman, L. Wang, D. Evans, and Q. Gu, "Distributed learning without distress: Privacy-preserving empirical risk minimization," in *Advances in Neural Information Processing Systems*, 2018, pp. 6343–6354.

[91] A. Dal Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi, "Calibrating probability with undersampling for unbalanced classification," in *2015 IEEE Symposium Series on Computational Intelligence*, IEEE, 2015, pp. 159–166.

[92] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography conference*, Springer, 2006, pp. 265–284.

[93] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[94] B. W. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.

[95] P. Baldi, S. Brunak, Y. Chauvin, C. A. Andersen, and H. Nielsen, "Assessing the accuracy of prediction algorithms for classification: An overview," *Bioinformatics*, vol. 16, no. 5, pp. 412–424, 2000.

[96] K. Pearson, "Note on regression and inheritance in the case of two parents," *Proceedings of the Royal Society of London*, vol. 58, pp. 240–242, 1895.

[97] D. M. Powers, "Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation," *Journal of Machine Learning Technologies*, 2011.

[98] J. D. Evans, *Straightforward statistics for the behavioral sciences*. Brooks/Cole, 1996.

[99] J. Champion, A. Shelat, and J. Ullman, "Securely sampling biased coins with applications to differential privacy," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, London, United Kingdom: Association for Computing Machinery, 2019, pp. 603–614, ISBN: 9781450367479.

[100] D. Dua and C. Graff, "UCI machine learning repository," *URL http://archive.ics.uci.edu/ml*, 2017.

[101] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, "Differentially private empirical risk minimization," *Journal of Machine Learning Research*, vol. 12, no. Mar, pp. 1069–1109, 2011.

[102] Y. Cao, Y. Li, S. Coleman, A. Belatreche, and T. M. McGinnity, "Detecting price manipulation in the financial market," in *2014 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr)*, IEEE, 2014, pp. 77–84.

[103] T. Leangarun, P. Tangamchit, and S. Thajchayapong, "Stock price manipulation detection using a computational neural network model," in *2016 Eighth International Conference on Advanced Computational Intelligence (ICACI)*, IEEE, 2016, pp. 337–341.

[104] L. Mendonça and A. De Genaro, "Detection and analysis of occurrences of spoofing in the brazilian capital market," *Journal of Financial Regulation and Compliance*, 2020.

[105] A. Li, J. Wu, and Z. Liu, "Market manipulation detection based on classification methods," *Procedia Computer Science*, vol. 122, pp. 788–795, 2017.

[106] X. Wang and M. P. Wellman, "Market manipulation: An adversarial learning framework for detection and evasion," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, C. Bessiere, Ed., Special Track on AI in FinTech, International Joint Conferences on Artificial Intelligence Organization, Jul. 2020, pp. 4626–4632.

[107] O. D. Lara and M. A. Labrador, "A survey on human activity recognition using wearable sensors," *IEEE communications surveys & tutorials*, vol. 15, no. 3, pp. 1192–1209, 2012.

[108] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep learning for sensor-based activity recognition: A survey," *Pattern Recognition Letters*, vol. 119, pp. 3–11, 2019.

[109] N. Soares and B. Fallenstein, "Aligning superintelligence with human interests: A technical research agenda," *Machine Intelligence Research Institute (MIRI) technical report*, vol. 8, 2014.

[110] S. Russell, "Learning agents for uncertain environments," in *Proceedings of the eleventh annual conference on Computational learning theory*, 1998, pp. 101–103.

[111] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the Sixteenth International Conference on Machine Learning*, 1999, pp. 278–287.

[112] S. Griffith, K. Subramanian, J. Scholz, C. L. Isbell, and A. Thomaz, "Policy shaping: Integrating human feedback with reinforcement learning," in *Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2*, 2013, pp. 2625–2633.

[113] B. Okal and K. O. Arras, "Learning socially normative robot navigation behaviors with bayesian inverse reinforcement learning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 2889–2895.

[114] P. Coscia, F. Castaldo, F. A. Palmieri, A. Alahi, S. Savarese, and L. Ballan, "Long-term path prediction in urban scenarios using circular distributions," *Image and Vision Computing*, vol. 69, pp. 81–91, 2018.

[115] M. Shimosaka, T. Kaneko, and K. Nishi, "Modeling risk anticipation and defensive driving on residential roads with inverse reinforcement learning," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2014, pp. 1694–1700.

[116] R. Triebel, K. Arras, R. Alami, L. Beyer, S. Breuers, R. Chatila, M. Chetouani, D. Cremers, V. Evers, M. Fiore, *et al.*, "Spencer: A socially aware service robot for passenger guidance and help in busy airports," in *Field and service robotics*, Springer, 2016, pp. 607–622.

[117] M. O. Riedl and B. Harrison, "Using stories to teach human values to artificial agents," in *AAAI Workshop: AI, Ethics, and Society*, 2016.

[118] S. Frazier, M. S. A. Nahian, M. Riedl, and B. Harrison, "Learning norms from stories: A prior for value aligned agents," *arXiv preprint arXiv:1912.03553*, 2019.

[119] M. S. A. Nahian, S. Frazier, B. Harrison, and M. Riedl, "Training value-aligned reinforcement learning agents using a normative prior," *arXiv preprint arXiv:2104.09469*, 2021.

[120] N. Bostrom, "Ethical issues in advanced artificial intelligence," *Cognitive, Emotive and Ethical Aspects of Decision Making in Humans and in Artificial Intelligence*, vol. 2, pp. 12–17, 2003.

[121] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.