Leveraging Domain Knowledge in Deep Learning Systems

A Dissertation Presented

by

Colin M. Van Oort

to

The Faculty of the Graduate College

of

The University of Vermont

In Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
Specializing in Complex Systems & Data Science

August, 2021

Defense Date: June 14th, 2021
Dissertation Examination Committee:

Safwan Wshah, Ph.D., Advisor
Jianing Li, Ph.D., Chairperson
Nicholas Cheney, Ph.D.
Christopher Danforth, Ph.D.
Brian Tivnan, Ph.D.
Cynthia J. Forehand, Ph.D., Dean of Graduate College

Abstract

Machine learning, and the sub-field of deep learning in particular, has experienced an explosion in research interest and practical applications over the past few decades. Deep learning approaches seem to have become the preferred approach in many domains, outpacing the use of more traditional machine learning methods. This transition has also coincided with a shift away from feature engineering based on domain knowledge. Instead, the common deep learning philosophy is to learn relevant features through the combination of expressive models and large datasets.

Some have interpreted this paradigm shift as the death of domain knowledge. I argue that domain knowledge is still broadly used in deep learning systems, and even critically important, but where and how domain knowledge is used has evolved. To support this argument I present three recent deep learning applications in disparate domains that each heavily rely on domain knowledge. Based on these three applications I discuss strategies for where and how domain knowledge is being effectively incorporated into newer deep learning systems.

Material for this dissertation has been published, or submitted for publication, in the following forms:

Van Oort, Colin M., Xu, Duo, Offner, Stella S., & Gutermuth, Robert A.. (2019). Casi: A convolutional neural network approach for shell identification. The Astrophysical Journal, 880(2), 83.

Van Oort, Colin M., Ferrell, Jonathon B., Remington, Jacob M., Wshah, Safwan, & Li, Jianing. (2021). AMPGAN v2: Machine Learning Guided Discovery of Anti-Microbial Peptides. Journal of Chemical Information and Modeling, 61(5), 2198–2207.

Van Oort, Colin M., Tivnan, Brian F., & Wshah, Safwan. (2021). Adaptive Agents and Data Quality in Agent-Based Financial Markets. Imminent submission to ACM Transactions on Intelligent Systems and Technology.

Acknowledgements

It takes a village to complete a doctorate. There are many people who have supported and guided me down the path to this dissertation, probably too many for me to mention all of them here. First, I'd like to thank my family, Nancy, Randall, Zach, Janine, and everyone else, for their support over the years. John, Dave, Thayer, thank you for your friendship and collaboration. Chris, Peter, Brian, Safwan, thank you for giving me a chance and all of your advice along the way. Robert Snapp, thank you for sparking my interest in machine learning. To all the students and faculty of the UVM Complex Systems Center, thank you for contributing to an amazing learning and research environment.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Domain knowledge is knowledge associated with a specific problem, system, or topic. The counterpart to domain knowledge is general knowledge, which is problem agnostic, or at least applicable to multiple problems. In the context of machine learning applications, domain knowledge comes from the application rather than from machine learning literature.

Domain knowledge is useful for many machine learning applications since it can be used to create strong biases before a model encounters data. Strong biases can improve a model in many ways, including reducing the amount of data required to reach desired performance. This improvement in data efficiency is extremely useful for many applications, since collecting training data can be difficult and costly. Supervised learning, one of the most common forms of machine learning, exacerbates the need for data efficiency since it requires the construction of a set of labels to accompany the training data. Human annotators are the most common source for training labels, though some applications may be able to automatically create labels using problem-specific metadata.

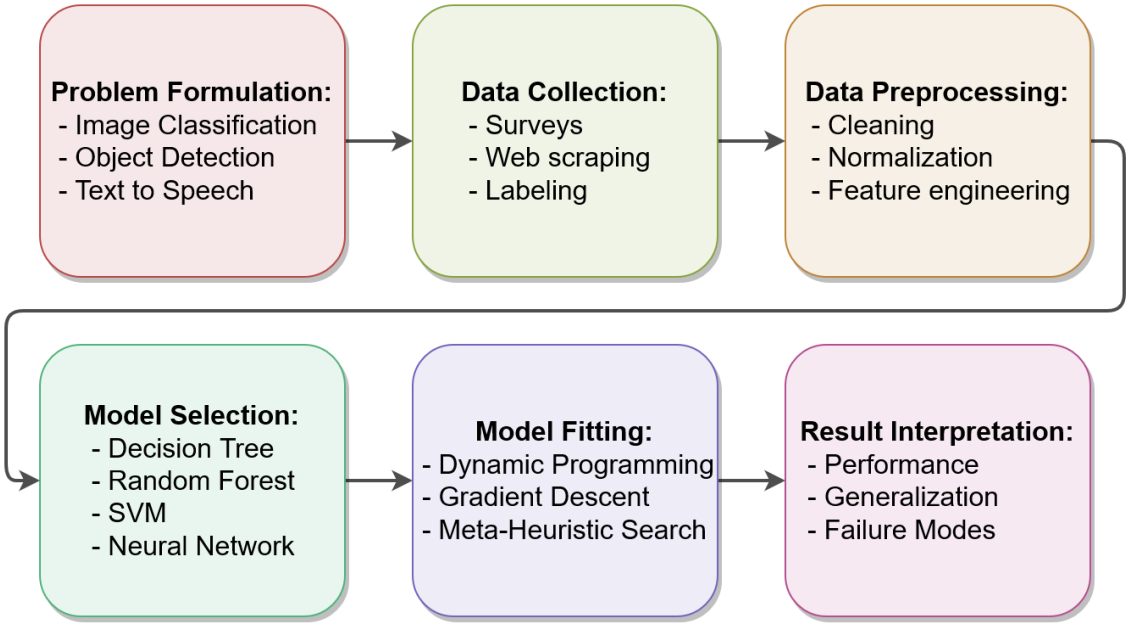

Figure 1.1: The development pipeline of a typical machine learning application, split into six steps.

The development pipeline of a typical machine learning application consists of roughly six steps: problem formulation, data collection, data preprocessing, model selection, model fitting, result interpretation. During problem formulation, a problem must be selected and formalized. After problem formulation, relevant training data must be collected, and labels must be constructed if supervised learning is to be used. Once training data has been collected, it needs to be processed so that it can be fed to a machine learning model. When the data has been properly cleaned and encoded, then an appropriate model needs to be selected. After a candidate model has been selected, it is fit to the training data. Finally, evaluation procedures allow the performance of the model, and any results produced during evaluation need to be interpreted. Figure 1.1 summarizes this pipeline.

In traditional machine learning pipelines, domain knowledge is extremely impor-

tant during problem formulation, data collection, data pre-processing, model selection, and result interpretation [162, 5, 267]. Domain knowledge plays an important role in problem formulation, where it influences which problems get investigated and serves as an important filter for the machine learning community. Additionally, domain knowledge often shapes the problem formalisms that are considered, so that the application can interface with existing tools and infrastructure if successful. Effective data collection requires domain knowledge to identify the features that should be gathered.

Problem formulation and data collection are critical since they provide the foundation for any successful machine learning application, but the machine learning research community has focussed more on the remaining steps of the pipeline in recent years. In particular, the data pre-processing step has received much attention, usually under the framing of feature engineering [141, 61, 270, 129].

Model selection can involve direct use of existing models, modification or extension of existing models, or the development of completely new models. However, most applications avoid creating new model types due to development costs and instead opt for direct use or modification of existing state-of-the-art models.

Model fitting tends to be impacted less by domain knowledge since it is primarily driven by mechanisms that are endogenous to the model. Result interpretation consists almost entirely of model performance evaluation, with the goal of understanding different aspects of performance such as generalization properties and failure modes. Domain knowledge is particularly useful for providing a domain-specific framing for performance evaluation, which makes it easier for domain experts to understand the effectiveness of new applications [173]. Additionally, many applications are difficult to

evaluate using general performance metrics, thus domain-specific metrics can provide relative and absolute frames of reference for model performance.

Newer applications built with deep learning tend to spend fewer resources on data preprocessing, and instead use a combination of larger datasets and more expressive models that can effectively learn how to extract salient features from the data [141, 117]. By learning to extract relevant features from relatively raw data, deep learning approaches can be easier to apply to new datasets, as long as enough labeled training examples are available. The field of deep learning has greatly expanded the effectiveness and ease of transfer learning, which involves training a model on a large and fairly general dataset, before fine-tuning on a smaller problem-specific dataset. This has been made possible by the emphasis that the deep learning community has placed on building models for classes of data (vectors, sequences, images, videos, etc.), rather than specific datasets.

The reduced role of domain knowledge has been a cause of concern for some in the machine learning community. The ability of deep learning to function largely without domain knowledge has lead to a proliferation of studies that consist of "mindless comparisons among the performance of algorithms that reveal little about the sources of power, or the effects of domain characteristics" [133]. Reliance on deep learning without domain knowledge can lead to poor performance on applications where costly data collection results in small datasets, which may not have enough samples to train effective feature extractors.

Despite the concerns of some, others have continued to investigate how best to use domain knowledge in deep learning applications [246, 172, 207, 264, 265, 260]. It seems that a consensus has been reached that the importance of domain knowledge

increases as the amount of available training data decreases. Domain knowledge is also critical when applications involve new data sources or problem formalisms, in which case it can reduce the amount of exploratory analysis needed to identify or construct an effective initial feature set.

In the following chapters, I present three recent deep learning applications as a context to examine these changes in the use of domain knowledge. The first application, a Convolutional Approach to Shell Identification (CASI), comes from the domain of astrophysics, where astronomers are interested in automatically detecting structures of interest in vast quantities of imaging data captured by telescopes. The second application, AntiMicrobial Peptide Generative Adversarial Network (AMPGAN), comes from the domain of molecular biochemistry, where chemists are interested in developing peptides with specific properties. The third and final application, Agent-Based Market Microstructure Simulation, comes from the domain of financial market modeling, where modelers wish to develop Agent-Based Financial Markets that are better able to inform policy and system design.

These applications come from disparate domains, but share at least two features: the use of deep learning to solve challenging real-world problems and a reliance on domain knowledge. By understanding how and where each of these applications leverages domain knowledge, we gain a better understanding of effective strategies for future applications. Though this coverage is far from complete, it provides evidence that domain knowledge remains critical for problem formulation, data collection, model selection, and result interpretation.

Chapter 2

CASI: A Convolutional Neural Network Approach for Shell Identification

This Chapter is derived from Van Oort et al. [238].

2.1 Abstract

We utilize techniques from deep learning to identify signatures of stellar feedback in simulated molecular clouds. Specifically, we implement a deep neural network with an architecture similar to U-Net and apply it to the problem of identifying wind-driven shells and bubbles using data from magneto-hydrodynamic simulations of turbulent molecular clouds with embedded stellar sources. The network is applied to two tasks, dense regression and segmentation, on two varieties of data, simulated density and synthetic ^{12}CO observations. Our Convolutional Approach for Shell Identification

(CASI) is able to obtain a true positive rate greater than 90%, while maintaining a false positive rate of 1%, on two segmentation tasks and also performs well on related regression tasks. The source code for CASI is available on GitLab.

2.2 Introduction

Forming stars influence their environment by injecting energy over a large dynamic range with different sources contributing at different times and characteristic length scales. Stellar feedback has been invoked to explain a host of phenomena including the relation between dense cores and the stellar Initial Mass Function [IMF, 4, 182], the longevity of turbulence within molecular clouds [46, 252, 185], the properties of multiple star systems [183] and the global efficiency of star formation [128, 143, 70]. Nevertheless, the energetics and impact of feedback remains poorly constrained.

Identifying feedback signatures and quantitatively disentangling the interaction with the environment are notoriously difficult. For decades, astronomers have studied the distribution of gas in the interstellar medium by making 2D dust emission and absorption maps and 3D atomic and molecular spectral cubes. A variety of algorithms have been developed to identify peaks in the data, namely cores and filaments, including CLUMPFIND, DENDROGRAMS and GETFILAMENTS [254, 90, 164]. However, simple structure identification algorithms like these fail to identify feedback signatures, which exhibit a variety of complex morphologies. Statistical approaches, such as principal component analysis and the spectral correlation function provide a means to quantify the underlying impact of feedback on the turbulent cloud structure; however, many statistics commonly applied to spectral line cubes are relatively insen-

sitive [22]. Consequently, the imprint of feedback is usually identified "by eye" [36, 8, 7, 144].

The human brain is a superb tool for parsing complex images [31], and a variety of papers have used visual identification to study feedback in surveys of individual regions [e.g., 127, 8, 7, 176, 144]. Features produced by stellar winds and outflows resemble shells, bubbles or cones in intensity maps, which is one way they can be visually identified [36, 219, 184]. In spectral line data, such as CO observations, feedback often appears connected over a range of velocities (frequencies), so astronomers often identify feedback by searching for coherent three-dimensional structures [8, 7, 144]. Meanwhile, the explosion of data over the last decade and production of large surveys, such as those covering the entire Galactic plane, have outstripped the analysis capacity of professional astronomers. This has led to a variety of "citizen science" efforts, in which interested members of the public visually inspect and characterize the data. Galaxy Zoo, which has undergone a number of iterations, involved millions of people, and produced dozens of papers to date, is the highest-profile of these initiatives [e.g., 148]. Recently, the Milky Way Project applied the power of citizen science to the identification of stellar feedback in the Spitzer Galactic plane surveys GLIMPSE and MIPSGAL. This effort yielded a catalog containing the locations and sizes of thousands of new bubbles in the Milky Way [219].

However, human classification, while formidable, has several disadvantages. Although numerous people devote significant time to data parsing, citizen hours are finite and only certain problems can be formulated into simple pattern searches for non-experts. Moreover, classifications are subjective and differ between people. This can produce different catalogs and conclusions for the same data even between experts

(compare Narayanan, Snell, and Bemis [176] with Li et al. [144], for example).

The very nature of feedback ensures that human identification will be ambiguous. Since stellar feedback acts on the interstellar medium, which by nature has a strongly inhomogeneous density and velocity distribution, signatures are usually asymmetric and often blend into the turbulent background [8, 7]. Voids, low density regions that are produced by supersonic turbulence, may also mascaraed as feedback-driven bubbles, causing false positives. Although stellar feedback can accelerate cloud gas to velocities above the mean cloud turbulent velocity, the peak velocity of the feedback is sensitive to the source orientation with respect to the line of sight and its location relative to the cloud boundary, where gas changes phase from molecular to atomic [8, 184, 144]. These complications mean that even experts have trouble unambiguously and accurately identifying feedback.

Algorithmic approaches to identifying bubbles have been utilized to reduce subjectivity of bubble identification, while also relieving the burden of human identifiers [80]. However, more traditional algorithmic approaches tend to lack the flexibility required for widespread application.

One alternative approach is machine learning, a sub-field of computer science in which algorithms adapt to patterns and correlations in data. Machine learning is now a mature field, and is commonly applied to pattern recognition problems, including topics ranging from genome sequencing to face recognition to drug discovery [141]. Machine learning can automate the process of feature identification, scale efficiently to large data sets, and produce repeatable catalogs. However, to date it has been applied relatively sparsely to problems in astrophysics.

In this work we present CASI, a convolutional approach for shell identifica-

tion [240][1]. CASI is a convolutional neural network, a variety of artificial neural network (ANN) where the primary unit of computation is the convolution operation rather than simple matrix multiplication. For an overview of convolution arithmetic in the context of machine learning, see Appendix 2.A for a brief overview and Dumoulin and Visin [66] for a more comprehensive guide[2]. ANNs are a computational model that is loosely inspired by biological neural networks, where the fundamental unit of computation is a single neuron that receives one or more stimuli and provides one or more output signals. See section 3 in Lieu et al. [147] for a brief overview of ANNs that targets the astronomy audience.

CASI is designed to identify feedback signatures in molecular clouds, with a focus on wind-driven bubbles created by intermediate-mass stars. This is motivated by the observation that such shells identified in nearby star-forming regions, like the Perseus molecular cloud, have a huge impact on the cloud energetics and evolution [7]. Magneto-hydrodynamic (MHD) simulations with embedded sources are used to train our method and investigate its efficacy.

In the remainder of §2.2, we summarize relevant machine learning applications in the literature. We describe our method in §2.3 and present results in §2.4. Finally, conclusions and discussion are provided in §2.5.

[1]The source code for CASI is available on GitLab: https://gitlab.com/casi-project/casi-2d

[2]An associated GitHub page provides helpful animations: https://github.com/vdumoulin/conv_arithmetic.

2.2.1 Machine Learning for Image Tasks

Hubel and Wiesel [113] identified specialized neurons in the visual cortices of cats and monkeys that process small, partially overlapping regions of their visual field. This pattern of local, overlapping connectivity inspired the design of the Neocognitron [76], a neural network based approach to character and digit recognition. However, difficulties encountered when training networks with more layers and a lack of sufficient training data led to a decline in the popularity of ANNs, with alternative methods such as support vector machines (SVMs) receiving more attention.

More than a decade later LeCun et al. [142] introduced LeNet-5, a Convolutional Neural Network (CNN) that broke the record for character recognition performance and became a baseline architecture for many applications of CNNs that followed, contributing to a resurgence the popularity of ANNs. This resurgence ushered in a wave of research and targeted hardware improvements that allowed ANNs to overtake many competing machine learning algorithms and attain state-of-the-art results on a variety of tasks, rivaling human performance in some cases [177].

In addition to character recognition, CNNs have been successful at image classification, object detection, semantic segmentation, and image denoising/artifact removal to name a few. For a broad overview of the techniques involved in CNNs and their applications see Gu et al. [96].

2.2.2 Previous Applications to Astronomical Data Analysis

Machine learning techniques have been applied to structure detection in astronomical data several times with varying degrees of success.

Beaumont, Williams, and Goodman [15] used SVMs to segment ^{12}CO data containing a supernova remnant partially obscured by a molecular cloud, reaching >90% accuracy when classifying hand-labeled pixels as belonging to the supernova remnant or molecular cloud.

SVMs are a supervised learning method that classifies data by finding a decision boundary that simultaneously minimizes classification error and maximizes the distance between the boundary and closest samples of any class. SVMs may also be applied to regression problems. Such applications are often referred to as support vector regression. Since SVMs attempt to maximize the margins about the decision boundary they tend to generalize well and feature robustness to minor perturbations of input data. Interested readers should refer to Bennett and Campbell [16] for an overview of SVMs.

Beaumont et al. [14] developed Brut, a method that utilizes Random Forest classifiers, to identify bubbles and similar structures in color-composite images from the Spitzer Space Telescope. However, Brut is sensitive to the position of the bubble in the image, making wide-field searches computationally expensive [261].

Deep learning is a relatively new and rapidly evolving sub-field of machine learning that features ANNs with sophisticated architectures and greater numbers of layers. Relatively few astrophysical applications utilize deep learning techniques, which may

be partly due to the age and the rapid research pace of the field. Daigle et al. [48] utilized a Multi-Layer Perceptron (MLP), a simple neural network architecture that features consecutive layers of densely connected artificial neurons, to identify expanding shells in the Canadian Galactic Plane Survey, obtaining a 0.6% false positive rate. Later Daigle, Joncas, and Parizeau [47] compared the performance of the MLP against two alternative network architectures, the competitive network and the growing neural gas network, on similar data. There was no clear winner in this comparison, since all three networks were able to correctly identify 10 out of the 11 bubbles considered when evaluated using a leave-one-out cross-validation method.

Dieleman, Willett, and Dambre [59] applied a CNN to the morphological classification of annotated images from the Galaxy Zoo project, attaining an accuracy $> 99\%$ for images where human annotators strongly agreed upon the classification label. The authors suggest that a machine learning system could be used to classify the "easy" images, leaving the more difficult cases for human annotators. Filtering the images in this way could lead to a reduced workload for human annotators when processing large surveys.

Lanusse et al. [134] trained a CNN to identify the existence of gravitational lensing in simulated data that was constructed to resemble Large Synoptic Survey Telescope (LSST) observations. This approach reached a true positive rate $\geq 80\%$ while maintaining a false positive rate of 1% on samples with varying signal-to-noise ratio.

The network employed in Lanusse et al. [134] utilizes residual connections, a network architecture feature introduced by He et al. [102] where identity connections combine the input and output data of a block of operations. Residual connections effectively change the underlying model of a network, or network component, from

$y = f(x)$ to $y = f(x)+x$, and encourage the network to learn iterative transformations of the input rather than a direct mapping [118]. Networks and network components that incorporate residual connections can easily learn the identity function, which allows them to mitigate the effects of harmful or under-performing components during learning. These properties allow architectures with residual connections to effectively utilize a greater number of layers and a larger number of model weights than architectures that do not include residual connections. See Section 2.A.6 for more details on residual architectures.

Primack et al. [194] utilized a simple CNN to classify images from the CANDELS survey into one of three phases of galaxy evolution. The network is trained using simulated CANDELS-like observations and then applied to real data, reaching so-called "Blue Nugget" phase galaxy identification accuracy of around 80%. This application involves a relatively small data set, thus the authors implemented several measures to keep the network from over-fitting, including data augmentation and dropout.

Lieu et al. [147] trained a CNN to classify solar system objects from other astronomical sources in simulated data. The network is initialized with weights that were trained on the ImageNet data set and then fine-tuned on 7512 simulated Euclid images. Similar to Primack et al. [194], this work utilizes various techniques to mitigate over-fitting, including batch normalization (see Appendix 2.A), dropout and data augmentation. After testing several modern CNN architectures, Lieu et al. [147] are able to reach an accuracy of 95.6% when distinguishing between four classes of stellar objects.

Most recently, Diaz et al. [58] investigated the classification of simulated galaxies into three classes. CNNs were applied to this task, using data generated from N-body

simulations as training data, and they were able to obtain an accuracy exceeding 99%.

The extent of previous work in this area, as well as the lack of a comprehensive and automated solution, motivates further application of machine learning techniques to structure identification in studies of star formation and the interstellar medium. In this study we apply the U-Net architecture, which is described in the following section, to several tasks derived from MHD simulation data.

2.3 Method Overview

2.3.1 Neural Network Architecture

In this work we employ a Residual U-Net, a variant of the U-Net architecture developed by Ronneberger, Fischer, and Brox [201] where the fundamental unit of construction is a residual block [102], rather than a single convolution. A residual block is simply a sequence of consecutively applied convolution operations that are spanned by a residual connection. See Appendix 2.A.6 for more details.

The U-Net architecture and its derivatives have grown in popularity since their introduction, and Residual U-Nets in particular have been applied to a wide variety of problems including road segmentation [268], detection of pulmonary nodules [132], segmentation of optic nerve tissue [55], and several other medical segmentation tasks [272].

Figure 2.1 displays our Residual U-Net architecture and provides details on the structure of each sub-component. Beyond the addition of residual connections, we also make a few other small alterations to the original U-Net architecture.

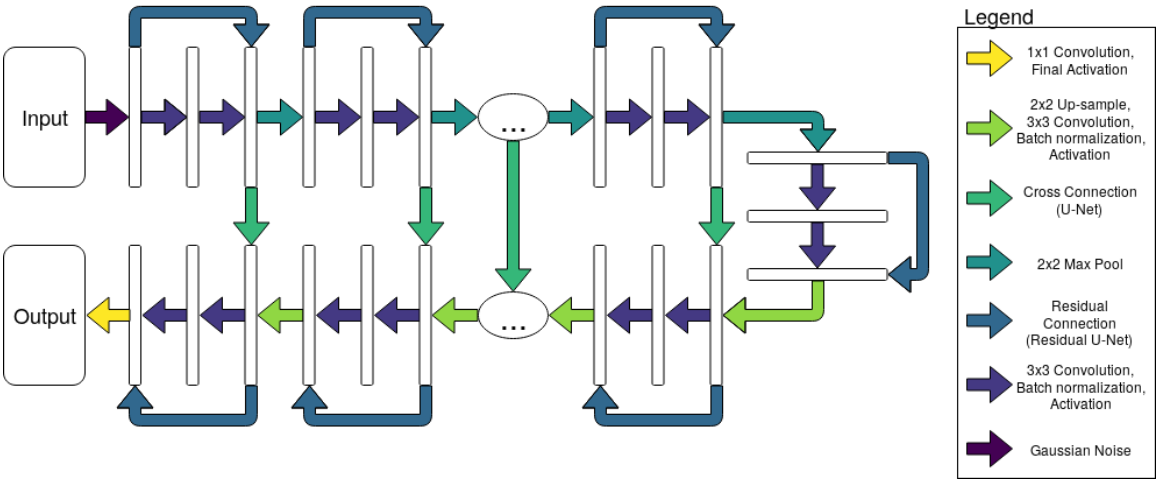

Figure 2.1: A flow-chart style depiction of our Residual U-Net architecture, which maps a 4D stack of images with dimensions (number of images, height of images, width of images, number of image channels) to a 4D stack of images (number of images, height of images, width of images, number of output channels). In the context of astronomy data, the "images" in question may be slices of observational data volumes and the height/width are literally the height and width of the observational data (in pixels/voxels). In this work we only utilize a single input image channel (gas density in a voxel, CO intensity in a voxel, etc.), however, it is possible to supply the network with multiple varieties of data using multiple input image channels. The first half of the network uses down-sampling operations to compress input features and construct higher-level representations, while the second half of the network uses up-sampling operations to reconstitute the abstract representations. Cross connections allow information from the down-sampling path to be utilized in the up-sampling path, leading to mappings that benefit from the combination of coarse and fine grained features. The ellipsis-within-oval graphics indicate that the depth of the architecture is variable and may be modified by the user.

In particular, we utilize padded convolutions[3] in place of unpadded convolutions, which results in feature maps with identical spatial dimensions at corresponding levels of the down-sampling and up-sampling paths. This removes the need to apply cropping to the cross connections and allows the depth of the network to be modified more easily when a particular problem benefits from the use of higher-level features. We make use of batch normalization prior to each activation function, which was not used by Ronneberger, Fischer, and Brox [201], since it can stabilize training and act as a light regularizer [116]. Note that batch normalization is not strictly necessary and may have a negative effect on performance for some tasks, thus it may be useful to re-evaluate its use when applying this architecture to new problem domains.

2.3.2 Training

We utilize stochastic gradient descent (SGD) with momentum to train our networks, following results from Wilson et al. [255] that suggest SGD may provide better generalization properties than adaptive step size methods, such as ADAGRAD [63] and ADAM [124]. Ruder [204] provides an excellent overview of gradient descent algorithms, with a focus on variants used in deep learning research and applications.

SGD is an optimization algorithm where the parameters of a function, such as the weights of a neural network, are adjusted using the gradient of a loss function with respect to those parameters. The loss function provides a performance criterion, and the gradient of the loss with respect to the model parameters indicates how the

[3]Padded convolutions augment the convolution operation by extending the spatial dimensions, e.g. height and width, of the input with generated data. One common padding scheme is to apply a band of zeros that is half as wide as the spatial dimensions of the convolution filter in that direction. This scheme results in a convolution whose input volume and output volume have identical dimensions when the convolution filters have odd spatial dimensions (e.g. 3, 5, 7, ...).

parameters should be adjusted in order to reduce the loss. The backpropagation algorithm, an application of the chain rule from differential calculus, distributes the gradients backwards through the network starting from the final layer.

The behavior of SGD can be controlled via the use of several parameters including the learning rate, batch size, and momentum intensity. The learning rate scales the magnitude of weight updates applied to the network in each step of SGD. Utilizing learning rates that are too high can lead to divergence, where the loss increases after each update and the network fails to learn, while learning rates that are too low may lead to premature convergence and extended training times.

Batch size determines how many training samples will be used to calculate the gradient at each step of SGD. Utilizing a batch size of one results in what is usually referred to as online SGD, while a batch size equal to the size of the training set results in batch SGD, and the use of batch sizes that fall between these two extremes results in mini-batch SGD. The batch size parameter features a trade-off between calculation speed and gradient accuracy when considering smaller vs. larger batch sizes. Nearly all modern applications of ANNs use mini-batch SGD for training, since online SGD can introduce too much noise into the training process and batch SGD tends to take too long to converge, though there is not a strong consensus on the optimal batch size setting. Masters and Luschi [159] indicate that smaller batch sizes, between 2 and 32, tend to work well in many cases, on the other hand, Hoffer, Hubara, and Soudry [106] suggest that larger batch sizes may also be effective, as long as the training duration is extended accordingly.

Momentum is an extension to SGD where each weight update is a linear combination of the current gradient and the previous weight update, which can reduce

oscillations in weight updates and speed up training convergence [85]. The momentum intensity parameter usually falls in the range $[0, 1)$ and controls the fraction of each weight updated that comes from the previous update. For example, setting the momentum intensity to 0.9 will result in each update consisting of the previous update multiplied by 0.9 plus the current gradient multiplied by 0.1. Alternatively, the momentum intensity may simply act as a learning rate applied to the previous weight update, rather than also scaling the current update.

We train our networks for 200 epochs[4] of SGD with the momentum parameter set to 0.9 and a batch size of 8.

The networks are initialized with random weights using the Glorot initialization scheme [82]. We utilize the uniform distribution variant of this scheme, which draws samples from a uniform distribution over the interval [-x, x], where

$$x = \sqrt{6/(fan\_in + fan\_out)},$$

fan_in is the number of input units for a weight tensor, and fan_out is the number of output units. Note, fewer training iterations may be required if the models are initialized with weights that have been previously trained on a similar data set and task, a process that is usually referred to as transfer learning [188]. During training the learning rate is adjusted using the cyclic learning rate schedule described in Huang et al. [112], with a maximum learning rate of 0.2 and 5 cycles of 40 epochs. Additionally, the training samples are shuffled at the end of each epoch, which effectively adds a small amount of noise to the gradient updates and can reduce the chance of getting

[4] A single training epoch involves several gradient updates, such that the network is exposed to each sample of the training set exactly once. In our case a single epoch consists of $\lceil N_T/B \rceil$ gradient updates, where N_T is the number of training samples and B is the batch size.

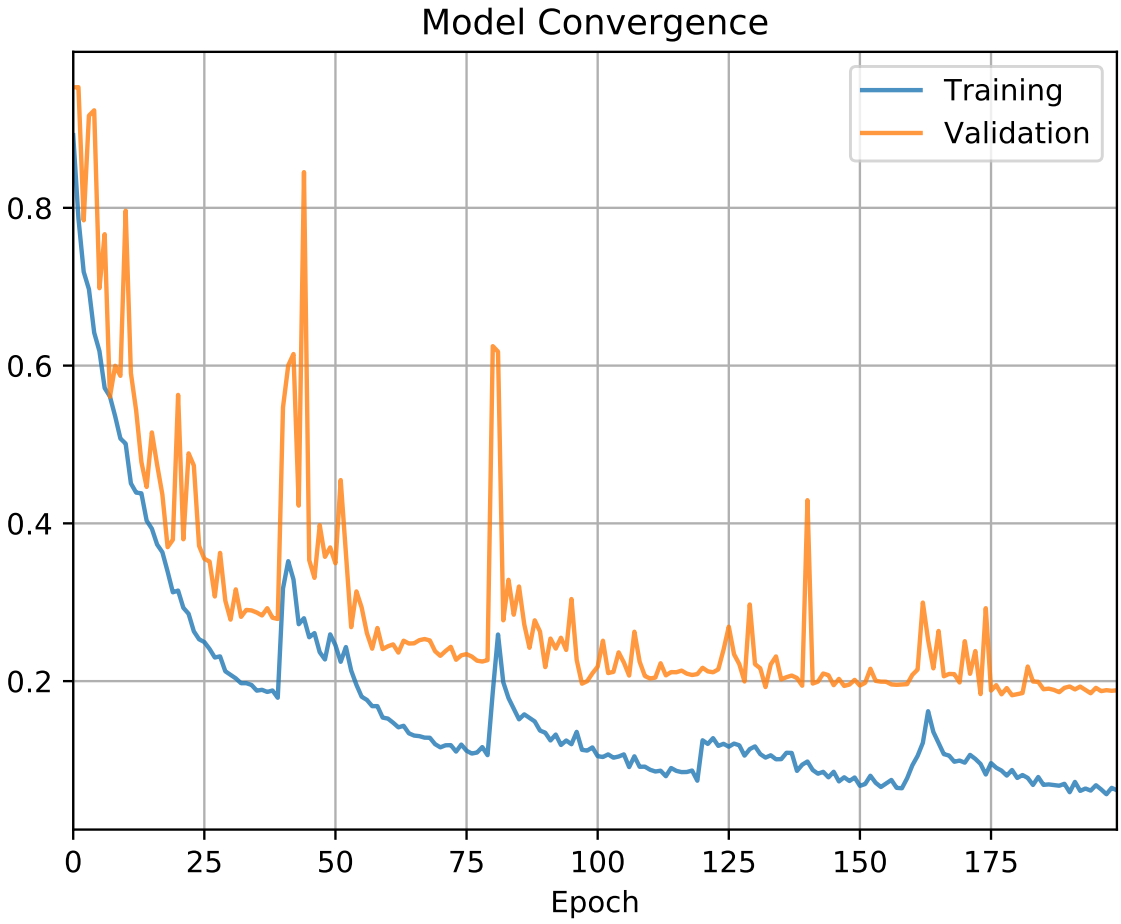

Figure 2.2: An example of the training and validation loss curves for a Residual U-Net trained on the CO segmentation task using the Intersection over Union loss function. The spikes that occur every 40 epochs are a feature of the cyclic learning rate schedule. Note that by the 200th epoch, network performance appears to have converged, with little improvement in the validation loss for 50 to 75 epochs.

Name	Definition	Symbol	Value
Batch Size	Samples provided during each training iteration	B	8
Depth	Number of blocks used in network construction	D	4
Filter Count	Filters allotted for each convolution operation	F	16
Noise Strength	Std dev of the noise applied to network inputs	σ	0.003

Table 2.1: Descriptions of network Hyper-Parameters and their selected values.

stuck in a local optimum. Finally, the model state is saved, via a check-pointing utility, each time a new minimum error is observed on the validation set.

2.3.3 Model Hyper-parameters

This section provides a detailed description of relevant hyper-parameters and how they influence performance of the model discussed in Section 2.3.1. Table 2.1 provides a brief summary of these hyper-parameters and the values utilized in our experiments.

The batch size of the network, which controls how many images are provided to the model simultaneously during training and inference, is determined by B. As mentioned in Section 2.3.2, there are trade-offs to be considered when selecting the batch size parameter. Larger batch sizes allow samples to be processed in parallel and may reduce training and inference times at the cost of additional memory overhead. Batch sizes greater than one allow for the aggregation of gradients over several data samples, providing more accurate gradient estimates and potentially reducing the number of training iterations required for the loss function to converge. Small batch sizes have been shown to have a beneficial regularizing effect on deep neural networks that may improve generalization [122, 106, 159]. Though progress has been made towards improving the effectiveness of networks trained with large batch sizes [106, 222], we tended to use small batch sizes due to memory limitations of graphics processing

units (GPUs) used to accelerate training.

The depth parameter, D, determines the number of fundamental blocks that are used in the construction of a particular network. For the Residual U-Net this is the number of convolution blocks, e.g. pairs of convolutions and associated operations such as batch normalization and residual connections, present in both the compressive and decompressive paths.

Each block in the Residual U-Net contains a spatial resampling operation, max pooling in the compressive path and nearest-neighbor upsampling in the decompressive path. See Section 2.A.3 for a brief overview of the mechanics and benefits of max pooling.

Thus, D governs the amount of dimension manipulation present in these architectures as well as the ability of the network to interact with the data at different spatial resolutions.

The depth parameter also contributes to the expressiveness of the network since each fundamental block includes one or more convolution operations. The expressiveness of a particular network refers to its ability to accurately approximate various functions. When considering two competing networks, X and Y, network X is more expressive than network Y if the set of functions that X is able to accurately approximate is a super-set of the set of functions that Y is able to accurately approximate. With this in mind, increasing D, and thus the number of model parameters, tends to improve the ability of the model to approximate functions and therefore increases its expressiveness.

The number of filters, F, indicates how many filters are allotted for each convolution operation, see Section 2.A.2 for more information about how the filters are used

in the model. Each down-sampling operation increases the number of filters allotted to down-stream convolution operations by a factor proportional to the dimension reduction, and similarly, each up-sampling operation decreases the number of filters provided for down-stream convolutions in proportion to the increase in spatial dimensions.

Additive Gaussian noise may be applied to the network inputs during training in order to avoid over-fitting, and the standard deviation of this noise is controlled by the σ parameter. The application of random noise to training samples can improve the robustness of the resulting method to small data perturbations and reduce the chances of over-fitting to the training data.

ANNs often require a computationally expensive hyper-parameter search process in order to reach desired performance levels. Some factors that contribute to the computational cost of this search are the number of hyper-parameters to be optimized, resources required to train the network, and complex non-linear relationships between various hyper-parameters and final model performance. We did not utilize a comprehensive hyper-parameter optimization method in this work, since hand-tuning alone provided adequate performance to demonstrate the effectiveness and flexibility of the method. Instead, we refer interested readers to relevant literature.

The simplest, and arguably least efficient, hyper-parameter optimization algorithms are grid-search and random-search. Bergstra and Bengio [17] investigates the relationship between these two methods and suggests that random-search may be a better choice.

Bayesian methods may offer a more intelligent method for exploring the space of network hyper-parameters, leading to a lower computational cost. Bayesian methods

are generally more complicated than random-search or grid search, thus there is a trade-off between compute time spent on the optimization and human time spent implementing more advance methods. Snoek, Larochelle, and Adams [223] provides an overview of Bayesian parameter optimization in the context of machine learning.

Evolutionary algorithms have also been successfully applied to hyper-parameter tuning. Examples include optimization of CNN hyper-parameters with a simple population-based evolutionary algorithm [12], optimization of SVM hyper-parameters with particle swarm optimization [97], and optimization of ANN hyper-parameters with co-variance matrix adaptation evolution strategies [CMA-ES, 153].

2.4 Validation

2.4.1 Simulation Training Set

Our study uses outputs from the simulations presented in Offner and Arce [180] as a training set. These calculations are performed with the ORION2 adaptive mesh refinement (AMR) code and follow the evolution of a 5 pc turbulent piece of a molecular cloud with five randomly distributed embedded sources. The stellar sources are represented by sink particles coupled to a sub-grid model for isotropic main-sequence stellar winds. See Offner and Arce [180] for additional details.

As a training set, the simulations have one essential advantage over observational data: they have complete information, including density, velocity, gas temperature and magnetic field at every point in the 3D volume. Of particular importance ORION2 has the capability to "tag" the gas launched in winds and follow its progress across

the domain [e.g., 181, 185]. The wind tracer field is a passive scalar, advected with the gas density, which tracks the amount of wind material in each cell. This field allows us to distinguish wind material from pristine cloud material and provides an exact map of the shells and bubbles created by the feedback (see §2.4.2).

For our training sets, we adopt outputs at different times from simulations with two different stellar distributions and two different initial magnetic field strengths as listed in Table 2.2. For training, we use only the 256^3 basegrid, thereby neglecting the information at higher "adaptive" resolution. This corresponds to a spatial resolution of ~ 0.02 pc.

2.4.2 Gas Density Training Set

We train our method with two different types of data. The first training set is constructed using the simulation gas density, ρ. We define the wind fraction as $f_t = \rho_t/\rho$, where ρ_t is the density of the wind material as tracked by the tracer field. Pixels with values of $f_t > 0.02$ are considered to be part of the feedback [e.g., 185]. These pixels define the target regions to be identified during training, testing and validation.

Due to the high expansion velocity of the wind shells, $v \gtrsim 1$ km s^{-1}, little mixing occurs outside the boundary of the swept-up material. Consequently, the feedback signatures are roughly spherical but are modulated by local density and magnetic field variations. Thus, in 2D image slices, the target training regions resemble irregular bubbles.

2.4.3 Synthetic CO Emission Training Set

The second training set is constructed from a suite of synthetic molecular line observations. We post-process each simulation output using the radiative transfer code RADMC3D[5] to obtain a spectral cube for the ^{12}CO(1-0) emission line. Following Offner and Arce [180], we adopt the Large Velocity Gradient (LVG) approximation, which calculates the level populations by solving the equations for local radiative statistical equilibrium. We use the gas densities and velocities on the 256^3 basegrid as inputs, where we convert from total mass density to molecular number density using $n_{\rm H_2} = \rho/(2.8 m_p)$ and $^{12}{\rm CO/H_2} = 10^{-4}$ [74]. Gas with temperatures exceeding 1000 K or with $n_{\rm H_2} < 50$ cm^{-3}, where all of the CO is likely dissociated, are assigned a CO abundance of 0. In addition, CO molecules freeze-out onto dust grains in cold gas with densities $n_{\rm H_2} > 10^4$ cm^{-3},

and CO molecules are dissociated by strong shocks, e.g., where the gas velocity exceeds 10 km/s, so we also set the CO abundance to 0 in these regions. We include turbulent line broadening below the grid resolution by adding a constant micro-turbulence of 0.25 km s^{-1}, which is consistent with the linewidth-size relation on this scale [135]. The spectral cube resolution is $\Delta v = 0.156$ km s^{-1}.

The tracer field, which tracks the stellar winds, records the amount of wind material in a given voxel (3D pixel). In order to use these data to define the positive and negative detections, we combine it with the gas velocity information and construct a spectral cube (position-position-velocity) that complements the synthetic CO emission. The approach we adopt is to map the tracer field to a density regime where $50 < n_{\rm H_2} < 10^4$ cm^{-3}. We then carry out the radiative post-processing described

[5] http://www.ita.uni-heidelberg.de/ dullemond/software/radmc-3d/

Model Properties			
Model	$t_{\rm run}$(Myr)	$\dot{M}_{\rm tot}(10^{-6}M_{\odot}{\rm yr}^{-1})$	$B(\mu G)$
W1_T2_0	0.0	0	13.5
W1_T2_0.1	0.1	41.7	13.5
W1_T2_0.2	0.2	41.7	13.5
W2_T2_0.1	0.1	4.5	13.5
W2_T2_0.2	0.2	4.5	13.5
W2_T3_0.1	0.1	4.5	5.6

Table 2.2: Model name, output time and the total stellar mass-loss rate. All models have $L = 5pc$, $M = 3762M_{\odot}$, initial gas temperature $T_i = 10K$, $N_* = 5$. The calculations are first evolved without sources for two Mach crossing times to allow initial cloud turbulence to develop.

above. The emitting regions in these cubes provide a map of the location of the wind-driven shells.

To account for observational resolution, we place each cube at a distance of 250 and 500 pc and convolve it with a 46" beam, which is the resolution of the COMPLETE ^{12}CO (1-0) survey of Perseus Ridge et al. [e.g., 199]. We also add random noise assuming $\sigma_{\rm rms} = 0.15$ K, which is comparable to the noise in the COMPLETE data.

2.4.4 Performance Metrics

The prediction of gas density and CO emission can be phrased in at least two ways, regression and segmentation. In the regression phrasing, the network is expected to output a floating point value corresponding with some measure of bubble material present in each pixel (e.g., molecular line emission or a continuum map, depending on the training data set). In the segmentation phrasing, the network is expected to classify each pixel as containing a 'low' or 'high' amount of bubble material.

The regression phrasing may provide more detail about perceived structures, al-

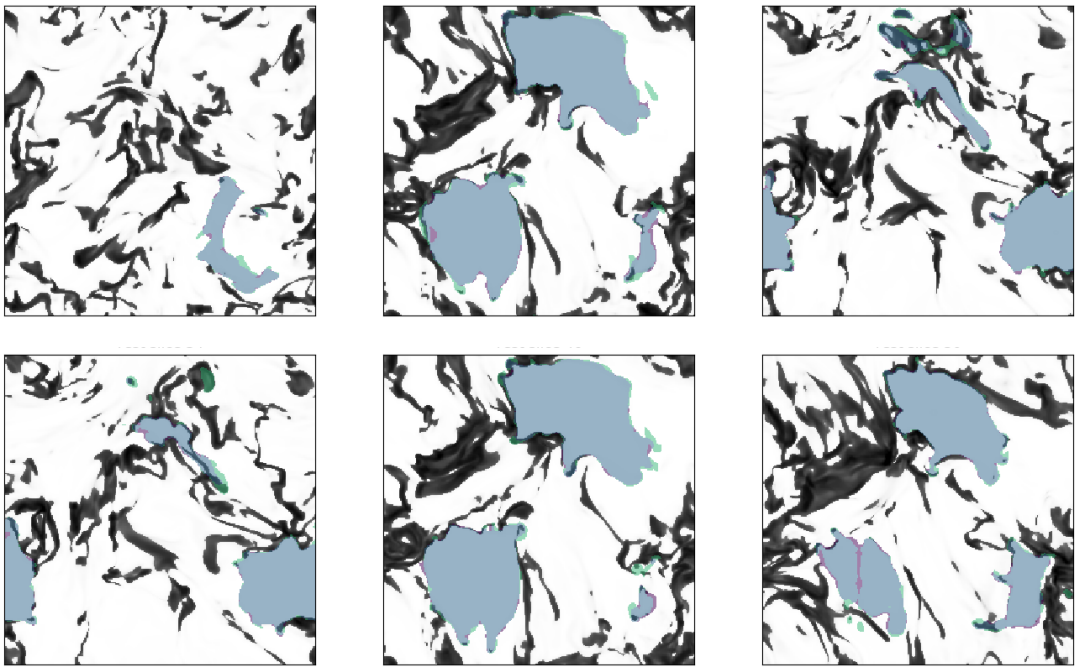

Figure 2.3: Density segmentation predictions from a Residual U-Net on samples randomly selected from the test set. Each frame contains a single slice from a simulated density cube, which is shown in gray scale. Since each slice is taken from a position-position-position cube, the x- and y-axis of each frame represent spatial coordinates within the cube. The tiles presented here have a side length of 5 pc that is inherited from the simulation. In each frame, true positives are shown in blue, false positives are shown in purple, false negatives are shown in green, and true negatives are not displayed. As a pre-processing step the density data was normalized so that it is now unit-less and falls approximately in the range [-0.4, 190], where lower density regions correspond with lighter colors and higher density regions correspond with darker colors. The color scale for the density data is identical across all tiles, and a logarithmic transformation is utilized in order to improve contrast.

lowing for certain kinds of analysis, such as the measurement of total bubble mass independently from the non-bubble gas along the line of sight. However, regression methods will need to handle heavy-tailed distributions of input and output values, which could lead to poor performance. The segmentation phrasing removes the potential difficulty of learning a heavy-tailed output distribution, but in doing so, loses some of the detail provided by regression methods.

Segmentation

Segmentation masks provide less detail when compared with regressed values, but they may be more useful for identifying interesting or important regions of the input data. For example, the outputs of segmentation models can be used to augment human efforts in processing large surveys by highlighting regions of interest or filtering regions without structures of interest.

The segmentation phrasing is achieved by selecting a threshold value, which may then be used to discretize the density and CO emission data. The threshold value may be selected arbitrarily by the user or by using some sort of heuristic, such as selecting a certain portion of the range of the data to constitute the negative and positive classes (e.g., the lower 1% of the range is the negative class and the upper 99% of the range is the positive class). We utilize a 1% threshold since it closely aligns with features that may be visually identified.

The loss function used in the segmentation phrasing is based on the Intersection over Union (IoU) score, also known as the Jaccard Index, and is defined as

$$\mathrm{IoU}\left(y,\ y'\right) = \frac{\mathrm{TP}\left(y,\ y'\right)}{\mathrm{TP}\left(y,\ y'\right) + \mathrm{FP}\left(y,\ y'\right)},$$

where $\mathrm{TP}(y,\ y')$ counts the number of true positives in prediction y' using the training label y and $\mathrm{FP}(y,\ y')$ counts the number of false positives.

The IoU score traditionally operates on binary inputs and is non-differentiable. In order to facilitate the training of neural networks via gradient descent, the following differentiable approximation is used,

$$\mathrm{IoU}\,(y,\ y') = \frac{\sum_{i=1}^{N} y[i] \cdot y'[i]}{\sum_{i=1}^{N} y[i] + y'[i] - \sum_{i=1}^{N} y[i] \cdot y'[i]},$$

where N is the number of pixels in y and $y[i]$ is the ith element of y. The IoU loss is simply $1 - \mathrm{IoU}(y, y')$.

Trained models are evaluated using tools from binary classification, namely confusion matrices and derived statistics, such as accuracy, F1 Score, and Matthew's Correlation Coefficient [192].

Given a confusion matrix with a number of true positives, TP, a number of true negatives, TN, a number of false positives, FP, and a number of false negatives, FN, accuracy is calculated as

$$\mathsf{Accuracy} = \frac{\mathsf{TP} + \mathsf{TN}}{\mathsf{TP} + \mathsf{TN} + \mathsf{FP} + \mathsf{FN}},$$

the F1 Score is calculated as

$$\mathsf{F1} = \frac{2 \times \mathsf{TP}}{2 \times \mathsf{TP} + \mathsf{FP} + \mathsf{FN}},$$

	True Pos	True Neg	False Pos	False Neg	Accuracy	F1-Score	Matthews Corr.
Mean	10.82	87.76	0.47	0.94	98.59	91.71	91.07
Std	7.04	7.75	0.34	0.82	1.08	10.83	10.42
Min	0.00	72.49	< 0.01	0.00	94.58	0.00	0.00
25%	5.43	80.00	0.02	0.36	97.98	90.99	90.18
50%	9.77	89.17	0.37	0.72	98.85	94.82	93.77
75%	17.09	93.13	0.69	1.22	99.37	96.11	95.45
Max	25.44	99.97	1.52	3.90	99.98	98.29	98.11

Table 2.3: Confusion matrix statistics for a Residual U-Net trained on the density segmentation task, computed over a test set containing 154 samples. True positives, true negatives, false positives, and false negatives are presented as a fraction of image pixels, thus assuming values between 0 and 100. The other three statistics, accuracy, F1-score, and Matthew's correlation coefficient, also assume values between 0 and 100, with higher values indicating better model performance. The minimum values observed in the F1-score and Matthew's correlation coefficient are caused by a few samples with no positively labeled pixels.

and Matthew's Correlation Coefficient is calculated as

$$\mathsf{MCC} = \frac{\mathsf{TP} \times \mathsf{TN} - \mathsf{FP} \times \mathsf{FN}}{\sqrt{(\mathsf{TP} + \mathsf{FP}) \times (\mathsf{TP} + \mathsf{FN}) \times (\mathsf{TN} + \mathsf{FP}) \times (\mathsf{TN} + \mathsf{FN})}}.$$

Receiver Operating Characteristic (ROC) curves are generated by plotting the true positive rate against the false positive rate of a model at different prediction threshold values and can provide more information about the predictive behavior of a classifier than single number statistics.

Regression

In the regression phrasing, models are trained using target values that have not been thresholded and the mean squared error (MSE) is used as the loss function.

Evaluation of regression models is traditionally dominated by the analysis of residuals, with the assumption that models featuring residuals that are tightly and sym-

metrically distributed about zero are better. We utilize the following scoring function in order quantitatively evaluate and compare models according to these assumptions,

$$f(R) = -|\langle R \rangle| - \texttt{std}(R) - |\texttt{skew}(R)|,$$

where f denotes the fitness function and R denotes the computed residuals. Note that the first term directly penalizes residual distributions whose mean value strays from 0, the second penalizes residual distributions that feature a non-zero standard deviation, and the final term penalizes residual distributions with non-zero skew.

Additional qualitative evaluation of the residuals can be obtained using histograms, kernel density estimates (KDE), and scatter plots, each providing a slightly different perspective on the distribution of residuals.

2.4.5 Case Study 1: Gas Density

In both problem phrasings the network is provided 2D slices of a 3D molecular gas density cube as input, though the expected output differs. As noted in Sections 2.4.4 and 2.4.4, the expected output for the regression task is the fraction of gas density associated with wind-swept bubbles and the expected output for the segmentation task is a binary mask that identifies regions with "high" levels of gas density associated with wind-swept bubbles.

We cut each 3D simulated density cube along its primary axes in order to form a stack of 2D slices, which are then divided into training, validation, and testing sets. We then normalize each set of 2D slices by subtracting the mean value and dividing by the standard deviation, after which it is ready to be used in training.

Density Segmentation

In Figure 2.3, we display examples of Residual U-Net predictions on several samples randomly selected from an unseen test set. In the figure, the gray scale components depict re-scaled density values and and the colored components depict network predictions and errors. Qualitatively, the model appears to correctly segment all major contiguous structures, though there may be some smaller structures that are missed. Additionally, note that the majority of errors are located on or near the edge of identified structures and would have little effect on whether or not a particular structure is identified. Finally, note that the upper left tile contains bubble structure that was correctly identified by CASI but may be difficult for a human to identify due to a lack of corresponding features in the density data.

In Figure 2.4 we present a ROC curve for the same model, which shows that the model attains a true positive rate of 95.52% while maintaining a false positive rate of only 1%. Supporting this, we summarize the distributions of several binary classification statistics in Table 2.3, where the classification statistics are computed across a test set of 154 samples. In particular, Table 2.3 clearly highlights the low error rate obtained by our model, where the maximum fraction of false positives is 1.52% and the maximum fraction of false negatives if 3.9%.

In order to better grasp the effect of random initialization on final model performance we trained 60 instances of the model using the same data and parameter settings, recorded the Receiver Operating Characteristic Area Under Curve (ROC AUC) statistic for each model, and then constructed confidence intervals for the mean of the ROC AUC distribution. The results of this experiment are presented in the first column of Table 2.4, which shows that CASI is able to consistently obtain

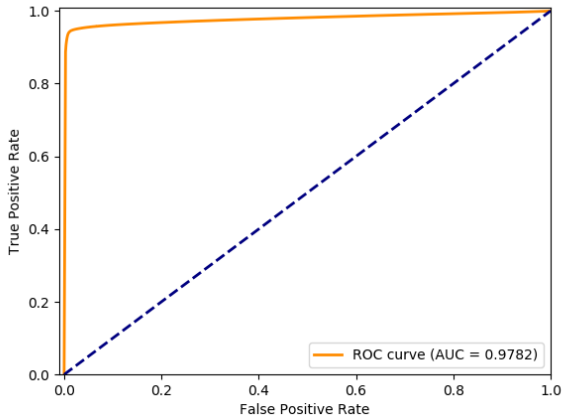

Figure 2.4: Example ROC curve for a Residual U-Net trained on the Density segmentation task. The dashed blue line represents $y = x$, which corresponds with the expected performance of a random binary classifier. A true positive rate of 95.52% is obtainable with a false positive rate of 1%, suggesting that this method may perform well as a content filter.

Performance Stat	Distribution Stat	Density Segmentation	^{12}CO Segmentation
ROC AUC	Mean	0.9768	0.909
	Std Error	0.0016	0.0018
	85% Conf Int	(0.9745, 0.9792)	(0.9063, 0.9117)

Table 2.4: Segmentation task performance statistics collected by training and evaluating 60 randomly initialized networks on the same training, validation, and testing splits. The first column indicates a statistic that was computed using the predictions of each trained network, while the second column indicates a statistic that was applied to the results of the column one statistic. The Receiver Operating Characteristic Area Under Curve (ROC AUC) statistic is calculated by computing the integral of the ROC curve, such as Figures 2.8 and 2.4.

ROC AUC scores close to the maximum value of 1.0.

Density Regression

Applying a Residual U-Net to the density regression task leads to a tight distribution of residuals that is not strongly correlated with the size of the input value, indicating that the model has captured much of the relationship between the input and output. Figure 2.5 displays a 2D histogram that shows the relationship between residuals and

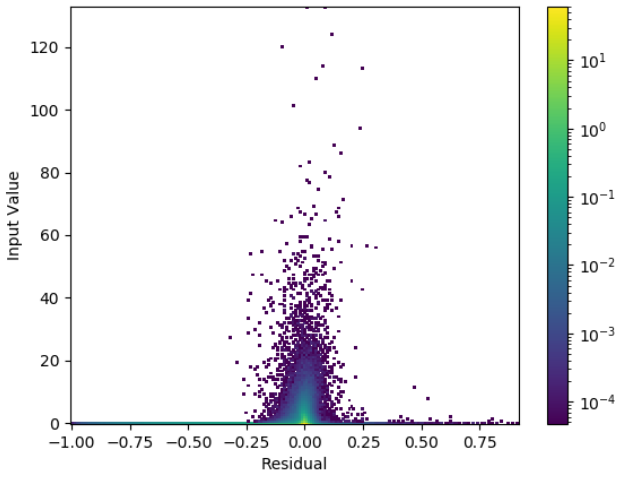

Figure 2.5: A 2D histogram investigating the scaling of residuals with respect to the input value for a Residual U-Net trained on the density regression task. The color scale is logarithmic in order to increase contrast and represents the density of points associated with each residual value-input value pair. Recall that the input values here are density values that have been scaled to have zero mean and unit standard deviation, thus the y-axis of this plot is unit-less. Due to the heavy-tailed nature of the input values, this re-scaling results in the data that falls approximately within the range [-0.4, 190].

input values.

Figure 2.6 displays the example prediction residuals for several samples from the test set. Note that the larger residuals tend to be clustered together near the edges of structures, similar to what was observed in the density segmentation setting.

2.4.6 Case Study 2: Synthetic Molecular Emission

The ^{12}CO data features position-position-velocity coordinates, rather than the position-position-position coordinates used for the density data. In both the segmentation and regression tasks the input data is inspected along the velocity axis such that the network is provided with position-position slices, those slices are divided into training,

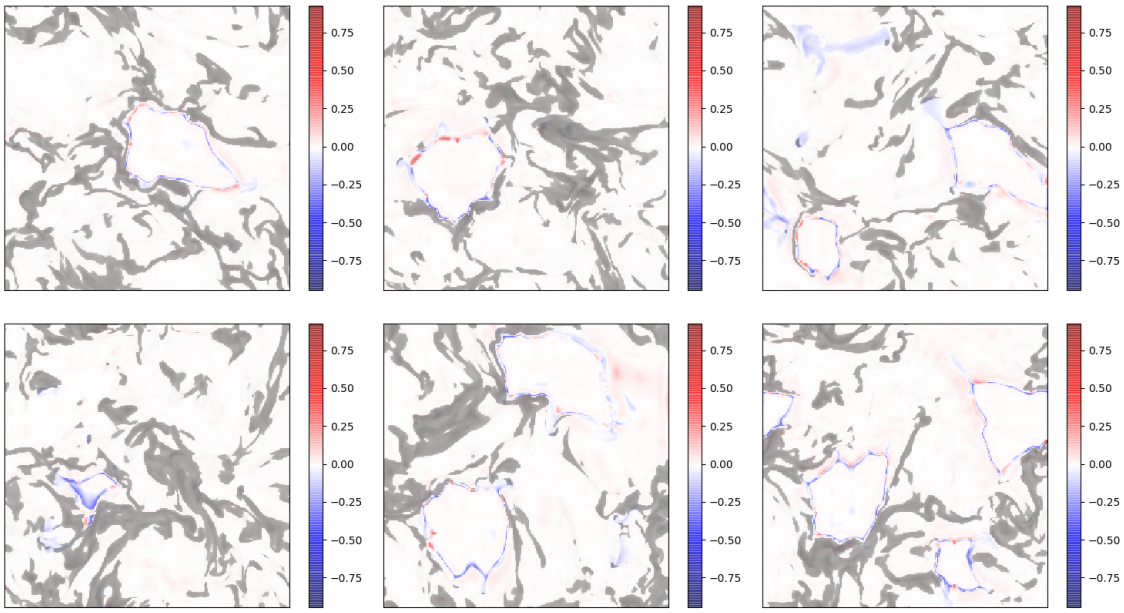

Figure 2.6: Example residuals from a Residual U-Net trained on the density regression task using the mean squared error loss function. Positive residuals, shown in shades of red, correspond to over-estimation, while negative residuals, shown in shades of blue, correspond to under-estimation. As with Figure 2.3, the side length of each tile is 5 pc and the gray scale components represent re-scaled density values.

Performance Stat	Distribution Stat	Density Regression	^{12}CO Regression
Mean	Mean	-0.0527	-0.019
	Std Error	0.0009	0.0008
	85% Conf Int	(-0.054, -0.0513)	(-0.0201, -0.0179)
Std Dev	Mean	0.2012	0.3483
	Std Error	0.0031	0.0011
	85% Conf Int	(0.1968, 0.2058)	(0.3466, 0.3499)
Skew	Mean	-3.8254	-13.17
	Std Error	0.0211	0.0854
	85% Conf Int	(-3.8562, -3.7946)	(-13.2945, -13.0454)
Score	Mean	-4.0793	-13.5372
	Std Error	0.01778	0.0854
	85% Conf Int	(-4.1053, -4.0534)	(-13.6618, -13.4126)

Table 2.5: Regression task performance statistics collected by training and evaluating 60 randomly initialized networks on the same training, validation, and testing splits. The first column indicates a statistic that was computed over the residuals of each trained network, while the second column indicates a statistic that was applied to the results of the column one statistic. The fourth element of the first column, Score, refers to the regression score defined in Section 2.4.4. CASI is able to reliably obtain a residual distribution with a mean near zero and a small standard deviation, indicating a tight residual distribution that is clustered about the origin. For both tasks the mean and skew components feature negative values, indicating that CASI tends to under-estimate values more often than it over-estimates values. Additionally, the negative skew value indicates that the tail of the residual distribution is longer in the negative direction, thus the largest errors tend to be under-predictions. However, the fact that the residual distribution is tightly grouped about the origin indicates that the relatively large skew value is not concerning and due in part to the characteristics of the input data.

validation, and testing splits, then each data split is normalized by subtracting the mean and dividing by the standard deviation.

CO Segmentation

The U-Net attains slightly lower performance in the ^{12}CO tasks, when compared with corresponding density tasks, even though the training set is more than a factor of two larger. This indicates that the relationship between the ^{12}CO observations and the constructed tracer data may be more complex than the relationship between the density data and corresponding tracer.

Figure 2.7 shows example predictions, which feature similar characteristics to the density segmentation predictions. The major structures are all correctly identified, with some smaller structures being missed, and errors clustered along the edges of larger structures.

The ROC curve, provided in Figure 2.8, features a sharp curve that is pushed up towards the upper-left corner of the plot, where the model reaches a true positive rate of 91.45% while maintaining a false positive rate of 1%. This accuracy is slightly lower than that achieved by the density segmentation task, however, the results still constitute excellent performance.

An investigation of final model performance variation due to random initialization is provided in column two of Table 2.4, which shows that CASI is robust to random initialization on the CO segmentation task.

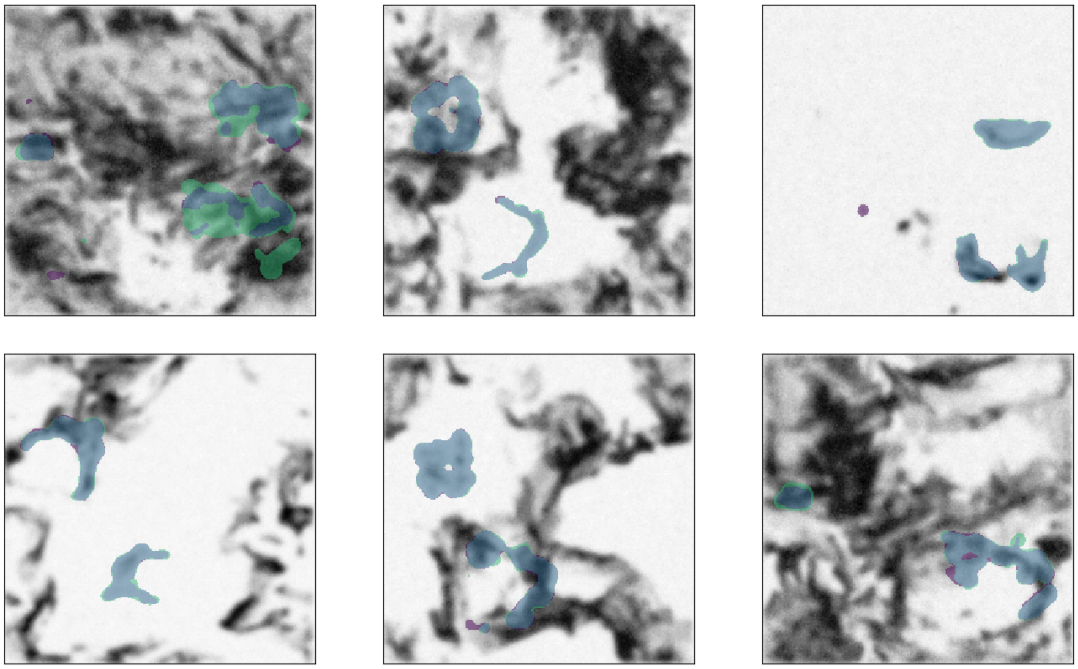

Figure 2.7: ^{12}CO *segmentation predictions from a Residual U-Net on samples randomly selected from the test set. As with Figure 2.3, the side length of each tile is 5 pc, the gray scale components represent re-scaled density values, true positives are shown in blue, false positives are shown in purple, false negatives are shown in green, and true negatives are not displayed.*

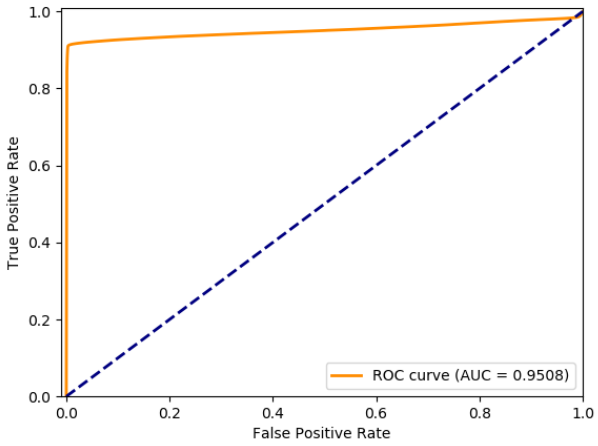

Figure 2.8: Example ROC curve for a Residual U-Net trained on the ^{12}CO segmentation task. The dashed blue line represents $y = x$, which corresponds with the expected performance of a random binary classifier. A true positive rate of 91.45% is obtainable with a false positive rate of 1%, supporting the proposal that this method may perform well as a content filter.

CO Regression

For the ^{12}CO regression task an element-wise logarithm operation is applied prior to the normalization operations in order to further reduce the dynamic range of the data. Specifically we apply the following transformation, $x' = \ln(1 + x - \min(x))$, where x is one of the training, validation, or testing sets, and $\min(x)$ is the element-wise minimum. Subtracting by the minimum value and adding 1 ensures that there are no invalid output values and that all values fall within the compressive regime of the logarithm.

We investigated the effect of random initialization on this task using the same experiment structure seen for the other conditions. These results are reported in column two of Table 2.5.

2.5 Conclusions

Our results indicate that methods from deep learning, namely the U-Net and its variants, are a flexible and effective tool for learning relationships in simulated density and ^{12}CO data. Moreover, our algorithm is completely general and could be trained to identify other astronomical signatures, such as protostellar outflows, filaments and dense cores, given appropriate training sets.

Our models learn well under several different conditions and generalize to unseen data from the same distribution with a minimal performance impact. Additionally, CASI features a low false positive rate and a clustering of errors that makes it well-suited to assisting astronomers by filtering large-scale survey data that is being inspected by humans.

We also note that CASI is relatively quick to train, especially on smaller datasets, taking approximately 2 seconds per epoch for the Density tasks (approximately 8 minutes for 200 epochs) and 7 seconds per epoch for the CO tasks (approximately 25 minutes for 200 epochs). After training, CASI can process more than 100 samples per second, allowing for rapid application to new data. With fast training times and even faster post-training predictions, CASI may be rapidly applied to new datasets with minimal overhead.[6]

Despite the generally positive results presented, there are several important research directions surrounding the application of deep learning techniques to facilitate the analysis of astronomical image data that have not been addressed.

First, all results presented in this work focus upon learning from simulated data,

[6]The computer that was used to collect timing information was outfitted with an Intel i7-6700K CPU and a Nvidia GTX 1080Ti GPU.

but in order to assist in the processing of survey data these models must operate on true observations that may greatly differ from the simulated data that they were trained upon. For example, we adopt a simple CO abundance model and do not take into account chemistry. Boyden et al. [22] show that self-consistently computing abundances and temperatures can produce statistically different emission maps. However, Xu and Offner [261] demonstrate that synthetic dust emission maps of the simulations also utilized here can be used to successfully train a random forest algorithm to correctly identify observed bubbles. This lends confidence that our CO emission maps have, at minimum, similar underlying morphologies to observational data. We extend our study to observational data in Xu et al. (in prep) and demonstrate that training sets based on synthetic CO emission can indeed be applied to observed CO data. Beyond assessing and improving the simulations that are used to generate training data, a comprehensive investigation of regularization and data augmentation techniques may lead to models that are better able to bridge the gap between simulation and observations.

Second, our methods leverage the high fidelity information and annotations provided by the simulations to learn relationships in a supervised setting. However, there exist considerable amounts of unlabeled survey and observational data that may be utilized in semi-supervised or unsupervised approaches. Semi-supervised and unsupervised approaches could reduce or remove the overhead involved with hand labeling and curating large data sets, while still drawing insights from said data.

Finally, only 2D models were investigated in this work, which ignore the 3D structure present in density and ^{12}CO cubes. We have found that 2D models seem to be sufficient for solving certain problems in this domain, certainly the benchmarks in-

vestigated here are well solved by 2D models, but some problems may require models with greater knowledge of 3D structure.

3D convolutional models have begun to find application in human action recognition [119], object detection in 3D point clouds [161], medical imaging [123], and other domains [236]. These 3D models may also be well-suited to identifying structures in stellar feedback, and we begin to explore such models, as well as their application to observational data, in upcoming work (Xu et al. in prep).

2.6 Acknowledgements

CVO, DX, SSRO, and RAG were supported by NSF grant AST-1812747. SSRO also acknowledges support from NSF Career grant AST-1650486.

Appendix

2.A Neural Network Operations

2.A.1 Batch Normalization

Batch normalization allows a network to re-normalize data at arbitrary points during the forward pass using moving mean and standard deviation statistics calculated over training batches. Following the description of batch normalization provided by Ioffe and Szegedy [116], if $\mathcal{B} = \{x_1, x_2, \ldots, x_n\}$ represents a batch of training samples then the mean and variance of the batch are calculated as

$$\mu_{\mathcal{B}} = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \sigma^2_{\mathcal{B}} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2.$$

The data are then normalized using the batch mean and variance

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}},$$

where ϵ is an arbitrary constant used for numerical stability. Finally, the output of the batch normalization is calculated using

$$y_i = \gamma \hat{x}_i + \beta,$$

where γ and β are learned parameters that allow the network to reverse or modify the batch normalization procedure when beneficial.

Batch normalization is applied slightly differently during training and inference, though this is handled internally by most deep learning frameworks. Interested readers should refer to Ioffe and Szegedy [116].

2.A.2 Convolution

A 2D convolution in this context involves an image with dimensions (image height, image width, image channels), or $(H_i,\ W_i,\ C_i)$, and a set of filters with dimensions (filter count, image channels, filter height, filter width), or $(F,\ C_i,\ H_f,\ W_f)$. The convolution is computed by sliding each filter over the spatial dimensions of the image. At each location an element-wise product between the filter and the corresponding image pixels is computed, the results of which are summed and become a single pixel in the output of the convolution. The sliding behavior of the convolution is controlled by horizontal and vertical stride parameters, s_h and s_v, which indicate how far the filter should move in each direction after each calculation. The output of the convolution described above would have the dimensions $\left(\frac{H_i - H_f + 1}{s_v},\ \frac{W_i - W_f + 1}{s_h},\ F\right)$.

It is common to pad the image with zeros in order to force the dimensions of the output into desired values. Notably, if the spatial dimensions of the filter are odd

and the image is padded by $\lfloor H_f/2 \rfloor$ on the top/bottom and $\lfloor W_f/2 \rfloor$ on the left/right then the the output of the convolution will have the dimensions $(H_i,\ W_i,\ F)$. This is referred to as the "same" padding scheme, since the output has identical spatial dimensions to the input. In practice, this operation is usually applied to a batch of several images in parallel.

2.A.3 Max Pooling

The max pooling operation is designed to reduce the spatial dimensions of an image while keeping the most important data intact. It does this by inspecting small sub-regions of the image, commonly 2×2 windows, and filtering out the maximum value in that sub-region. The max pooling operation, like the convolution described above, has stride parameters which adjust the spatial relationship between the sub-regions. It is common to have strides that are equal to the size of the sub-regions, resulting in disjoint sub-regions which fully cover the input image.

Note that max pooling is applied to each channel independently, and thus the result of applying a max pooling operation with a 2x2 window and a stride of 2 to a (64, 64, 3) image would be a (32, 32, 3) image.

2.A.4 Nearest-Neighbor Interpolation

Nearest-neighbor interpolation is an extremely simple up-sampling operation that increases the spatial dimensions of an image by an integer factor, n, by expanding each pixel into an $n \times n$ block with identical values. This may be used to reverse the effects of a max pooling operation, though some detail is lost.

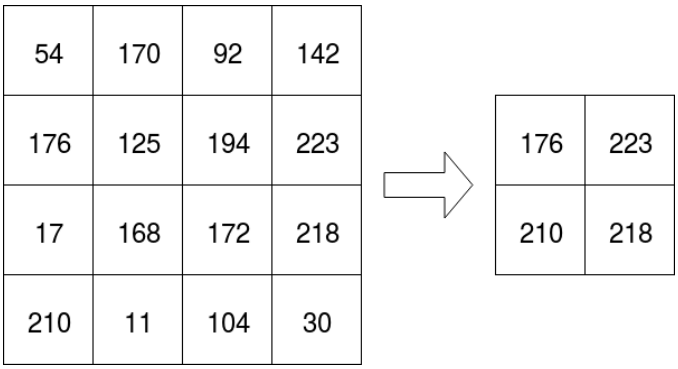

Figure 2.A.1: Max pooling with a 2×2 window, used to map a 4×4 input to a 2×2 output

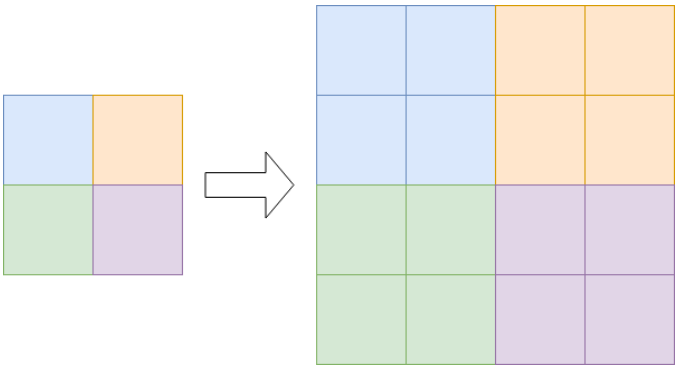

Figure 2.A.2: Nearest-neighbor interpolation with a 2×2 window, used to map a 2×2 input to a 4×4 output.

2.A.5 Activation: Exponential Linear Units

Introduced by Clevert, Unterthiner, and Hochreiter [37], the exponential linear activation function is defined as:

$$\text{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0, \end{cases}$$

where α controls the negative saturation value of the function. Use of exponential linear units (ELU) has been shown empirically to allow faster and more robust training of deep neural networks when compared to rectified linear units (ReLU) and other common activation functions.

Scaled Exponential Linear Units (SELU), defined as

$$\text{SELU}(x) = \lambda \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0, \end{cases} \qquad \lambda > 1,$$

exhibit similar properties to ELUs but with the added benefit of having a normalizing effect on network activations, similar to batch normalization. See Klambauer et al. [126] for more details.

2.A.6 Residual Connections

Sometimes referred to as skip connections, this architecture component can improve performance [102], reduce training instability in deeper networks [102], and encourage iterative inference [118].

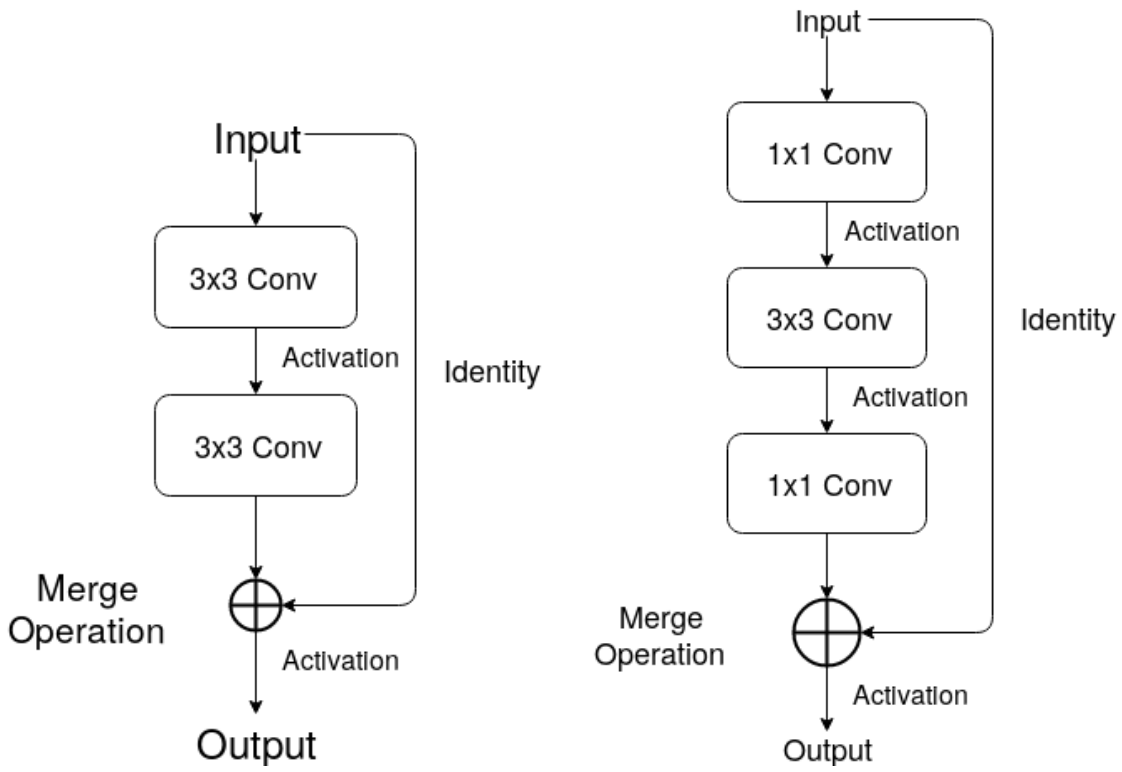

Figure 2.A.3: Left: A basic residual block using 3×3 *filters, the number of filters used in each convolution is a free parameter that must be selected. Common activation functions include ReLU, sigmoid, and tanh. Common merge operations include concatenation, element-wise addition, and element-wise maximum [maxout, 89]. If addition is used as the merging operation then a projection skip-connection, commonly implemented using a* 1×1 *convolution, may be required in place of the identity skip-connection in order to obtain the correct dimensions for the merge operation. Right: A bottleneck residual block, which uses* 1×1 *convolutions in order to reduce the number of parameters required, relative to the basic residual block. If the input volume has* n *channels, it is common to use* $n/2$ *or* $n/4$ *filters in the first two convolutions followed by* n *channels in the final convolution. This compresses the data before the larger convolution is applied resulting in a reduced number of parameters.*

Chapter 3

AMPGAN v2: Machine Learning Guided Design of Antimicrobial Peptides

This chapter is derived from Van Oort et al. [243].

3.1 Abstract

Antibiotic resistance is a critical public health problem. Each year ∼2.8 million resistant infections lead to more than 35,000 deaths in the U.S. alone. Antimicrobial peptides (AMPs) show promise in treating resistant infections. However, applications of known AMPs have encountered issues in development, production, and shelf-life. To drive the development of AMP-based treatments it is necessary to create design approaches with higher precision and selectivity towards resistant targets.

Previously we developed AMPGAN and obtained proof-of-concept evidence for

the generative approach to design AMPs with experimental validation. Building on the success of AMPGAN, we present AMPGAN v2 a bidirectional conditional generative adversarial network (BiCGAN) based approach for rational AMP design. AMPGAN v2 uses generator-discriminator dynamics to learn data driven priors and controls generation using conditioning variables. The bidirectional component, implemented using a learned encoder to map data samples into the latent space of the generator, aids iterative manipulation of candidate peptides. These elements allow AMPGAN v2 to generate of candidates that are novel, diverse, and tailored for specific applications—making it an efficient AMP design tool.

3.2 Introduction

AMPs contribute to the natural immune response in all classes of life and are active against a broad spectrum of microbes [198, 2]. Some AMPs are less likely to induce bacterial resistance, relative to traditional small molecule antibiotics [137, 226]. Additionally, AMPs can have synergistic effects when used in combination with traditional antibiotics [108, 203, 269] or other AMPs [262, 266].

Over 15,000 antimicrobial peptides (AMPs) have been identified [190], but few have been advanced to clinical trials despite their promise as treatments for antibiotic resistant pathogens. Many known AMPs have limitations that have prevented effective therapeutic application, such as relatively low half-lives [160, 86], undesirable or unknown toxicity to human cells [23, 157], and high production costs relative to traditional antibiotics [23, 104, 155].

Designing AMP candidates that mitigate these shortcomings is a difficult problem.

AMPs are made of amino acids arranged in a chain of arbitrary length, and feature a massive chemical search space. There are approximately 4.5×10^{41} unique peptides with 32 or fewer residues, if we consider only the 20 standard proteinogenic amino acids. Since the number of confirmed AMPs is low in comparison, it seems that the density of AMPs in the space of all peptides is also low [30]. Efficient methods are required to effectively develop AMP-based therapeutics.

Machine learning has aided in the discovery and development of AMPs, with many recent approaches relying on predictive models [130, 73, 233, 170, 169, 168, 259, 3, 13, 256]. Such approaches are usually labelled as quantitative structure-activity relationship (QSAR) models. The basic QSAR recipe is to select a property of interest (e.g., antimicrobial activity), train a machine learning model to predict that property using relatively easily obtained features (e.g., primary peptide structure), then apply the trained model to unlabelled samples to estimate the property of interest. After training, QSAR models can be used to identify properties of peptides present in a database that have yet to be experimentally validated.

The predictive approach can be extended to a generative one by adding an uninformed candidate generator (e.g., select a random peptide with length no more than 32). The randomly generated candidates can then be sorted and selected based on the property predicted by the QSAR model. This approach often suffers from excessive sampling requirements that inhibit discovery and design applications, due to the sparsity of AMPs in the peptide space. Additionally, reliance on engineered features constructed with domain expertise can further restrict the ability of these models to generate candidates that are qualitatively distinct from known AMPs. For example a commonly used feature is structure-based, however, at the time of writing only

approximately 2.5% of known AMPs have structures, severely hindering the use of structure as an AMP predicting metric. In fact analyzing the presence of amino acids for structures with either alpha or beta characteristics (e.g., table 3.A.1) demonstrates that half the amino acids show up with less prevalence than chance, and those that do appear more frequently do not share equal probabilities, implying utility beyond structure. Even if the statistical rules were stronger, it is quite possible that some AMPs simply have no well-defined structure [136].

Explicitly generative models that are better informed by data can reduce the amount of sampling required to identify promising candidates. Proving this point, several studies have successfully applied recurrent neural networks (RNNs) [171, 174] and variational autoencoders (VAEs) [125] to AMP design and discovery [51, 50, 217, 101, 34]. If we expand our scope to the more general case of molecular design, we find several more applications of VAEs [87, 120], some of adversarial autoencoders (AAEs) [121, 18], and even the use of a generative adversarial network (GAN) [6].

Despite fairly broad adoption of machine learning techniques in this domain and growing interest in generative models, there is relatively little work investigating the use of generative adversarial networks (GANs) for AMP design and discovery [237]. GANs are generative models that learn to produce samples from arbitrary data distributions by pitting a pair of artificial neural networks, dubbed the generator and discriminator, against each other in a zero-sum game [210, 211, 88]. This family of models has seen great success in learning to generate images following an explosion of research interest in 2014 [88, 167, 150, 35, 205, 25]. GANs can also generate text [111, 71, 33], a task that is qualitatively similar to AMP sequence generation and may indicate the potential for a new application.

Recently, we provided a proof-of-concept for such an application with AMPGAN and tested its ability to design antibacterial peptides [72]. For 12 generated peptides that are cationic and likely helical, we assessed the membrane binding propensity via extensive molecular simulations. The top six peptides were promoted for synthesis, chemical characterizations, and antibacterial assays. Three of the six candidate peptides were validated with broad-spectrum antibacterial activity.

GANs have served as core components in several creative image manipulation tools [26, 271, 189], allowing for the generation of realistic looking images that satisfy user imposed constraints. Inspired by the iterative and controllable development process afforded by these creative image manipulation tools, we seek to apply similar models to AMP design. In particular, bidirectional conditional GANs (BiCGANs) [65, 60] are ideal for the AMP design task, since they provide a data driven generative process, designer control over some features of generated samples, and iterative development.

The data driven priors are learned via the zero-sum game between the generator and discriminator. In this game, the generator maps samples from a latent distribution (e.g., a multi-variate normal distribution) to samples that appear to be drawn from the real data distribution, while the discriminator (or critic) is given samples and must identify if they were drawn from an authentic data distribution or produced by the generator. During training the discriminator minimizes a classification error, while the generator maximizes the error of the discriminator.

GANs can create realistic looking samples, but each sample will contain arbitrary features. In BiCGANs the control that we seek is created through the use of conditioning variables [167], where the generator and discriminator are provided an

additional input that contains meta-data for the current sample. By allowing the discriminator to learn associations between features and conditioning variables, the generator is encouraged to account for the same associations, which then allows a designer to control the output of the generator. The conditioning variables are often constructed as binary vectors that indicate the presence or absence of the features of interest. For example, in an image generation context, a conditioning vector could indicate whether the generated image should contain certain objects.

The iterative development process that we want to enable is made possible by the bidirectional component of the BiCGAN. The bidirectional component is driven by a third network, the encoder, which maps data samples (e.g., AMP sequences) into the latent space of the generator. This allows real data samples to be projected into the latent space, which can be used to create landmarks in the latent space, facilitate latent space interpolations, and incrementally manipulate a particular sample.

In the following sections we discuss our training data, data pre-processing, and details of AMPGAN v2—our BiCGAN-based model for AMP design. We show that AMPGAN v2 can generate novel AMP candidates with similar physio-chemical properties to the training data, while also incorporating designer constraints.

3.3 Methods and Models

3.3.1 Training Data

We constructed our training set by combining the Database of Antimicrobial Activity and Structure of Peptides (DBAASP [84, 190]), Antiviral Peptide database

(AVPdb [195]), and UniProt [43] databases. We extracted the FASTA formatted sequence information, microbe targets (e.g., Gram-positive bacteria, Gram-negative bacteria, viruses), mechanism targets (e.g., cell membrane, cytoplasmic protein, cell replication), and activity measures (primarily MIC50 measured in μg/ml) from each database as available. Sequences containing non-FASTA symbols (e.g., tail modifications, lower case characters, etc.) or more than 32 amino acid residues were filtered. We chose MIC50 as our primary activity measure since it was one of the most prevalent measurements present in DBAASP. We did not consider other activity measures, such as MBC, due to difficulty in correctly combining such measurements with MIC50.

After removing duplicated sequences between DBAASP and AVPdb, as well as "false negative" sequences from UniProt that also appear in DBAASP or AVPdb, we obtained 6238 sequences from DBAASP, 312 sequences from AVPdb, and 490341 sequences from UniProt. If a particular sequence has measured effectiveness against multiple microbe targets or mechanism targets, then we considered the superset of these. For sequences that have multiple activity measurements against one or more microbes, all measurements with compatible units are converted to μg/ml and the arithmetic mean was used to represent the general antimicrobial activity of the sequence.

Conditioning Data

We constructed conditioning vectors for our model using indicators for the target microbes, target mechanisms, MIC50 level, and sequence length (Figure 3.3.1). The target microbe classes are cancer, fungus, Gram-positive bacteria, Gram-negative bacteria, insect, mammalian, mollicute, nematode, parasite, protista, and virus. The

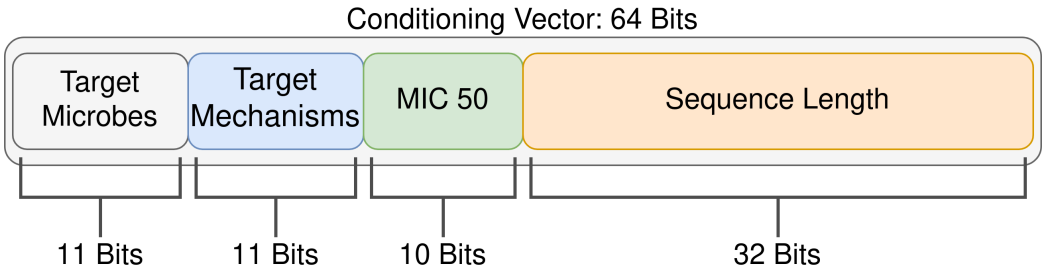

Figure 3.3.1: A visual summary of the contents and dimensions of a conditioning vector. All elements are binary encoded. For the target microbes and target mechanisms each element of the binary vector indicates activity against a particular microbe class or cellular mechanism. A one-hot encoding is used for the MIC 50 element, indicating membership in single MIC 50 decile. The sequence length is encoded as a bit mask, where 1 indicates the presence of a character and 0 indicates an empty slot.

target mechanisms are lipid bilayer, replication, virus entry, DNA/RNA, cytoplasmic protein, assembly, virucidal, membrane protein, surface glycoprotein, release, and unknown.

The conditioning vector is then constructed as a 64 bit binary vector. The target microbes are encoded with 11 bits indicating activity, or lack thereof, against each microbe group. Likewise, the target mechanisms are encoded with 11 bits indicating interaction with a particular cell process or element. The MIC50 values are discretized into deciles using the following bin edges: 3.7×10^{-6}, 5.7557×10^{0}, 1.1×10^{1}, 1.79869×10^{1}, 2.7×10^{1}, 3.88498×10^{1}, 5.75996×10^{1}, 8.53173×10^{1}, 1.28×10^{2}, 2.324687×10^{2}, and 1.1240×10^{4} μg/ml. Finally, the length of the sequence is represented using 32 digits, each indicating the presence or absence of a FASTA character.

We assumed that the sequences from UniProt did not have antimicrobial activity, since arbitrary peptides are unlikely to feature antimicrobial properties, and we already removed known AMPs. Thus, when we constructed conditioning vectors for these sequences the only non-zero elements were the length component, which was

set appropriately, and the MIC50 component, which was set to the highest bin (the lowest activity).

Figures 3.B.1 and 3.B.2 show the distributions of values across the conditioning vector elements (i.e. target microbes, target mechanisms, MIC50, and sequence length).

3.3.2 AMPGAN v2 Design and Training

AMPGAN v2 is a BiCGAN constructed with three neural networks: the generator, discriminator, and encoder (Figure 3.3.2A).

The generator is composed of a dense layer that mixes the latent representation and conditioning vector, followed by a stack of exponentially dilated convolutions, and terminated by a single convolution that combines the multi-scale features extracted by the prior convolution stack (Figure 3.3.2B). Global position information is added to the features as they enter the convolution stack to improve global sequence structure [149].

The discriminator architecture contains a stack of strided convolutions, followed by several dense layers (Figure 3.3.2C). We apply spatial dropout before each convolution and dropout before each dense layer, excluding the output layer. Strided convolutions are used to quickly downsample the feature maps, while dropout increases the variance of the signal provided by the discriminator and can stabilize training [25].

The AMPGAN v2 encoder shares the same architecture as the discriminator, with the only difference being a larger final layer with a linear activation function.

We trained AMPGAN v2 for 2000 epochs, where AMPGAN v2 was shown all 6550 AMP sequences along with a random sample of the 490341 Non-AMP sequences

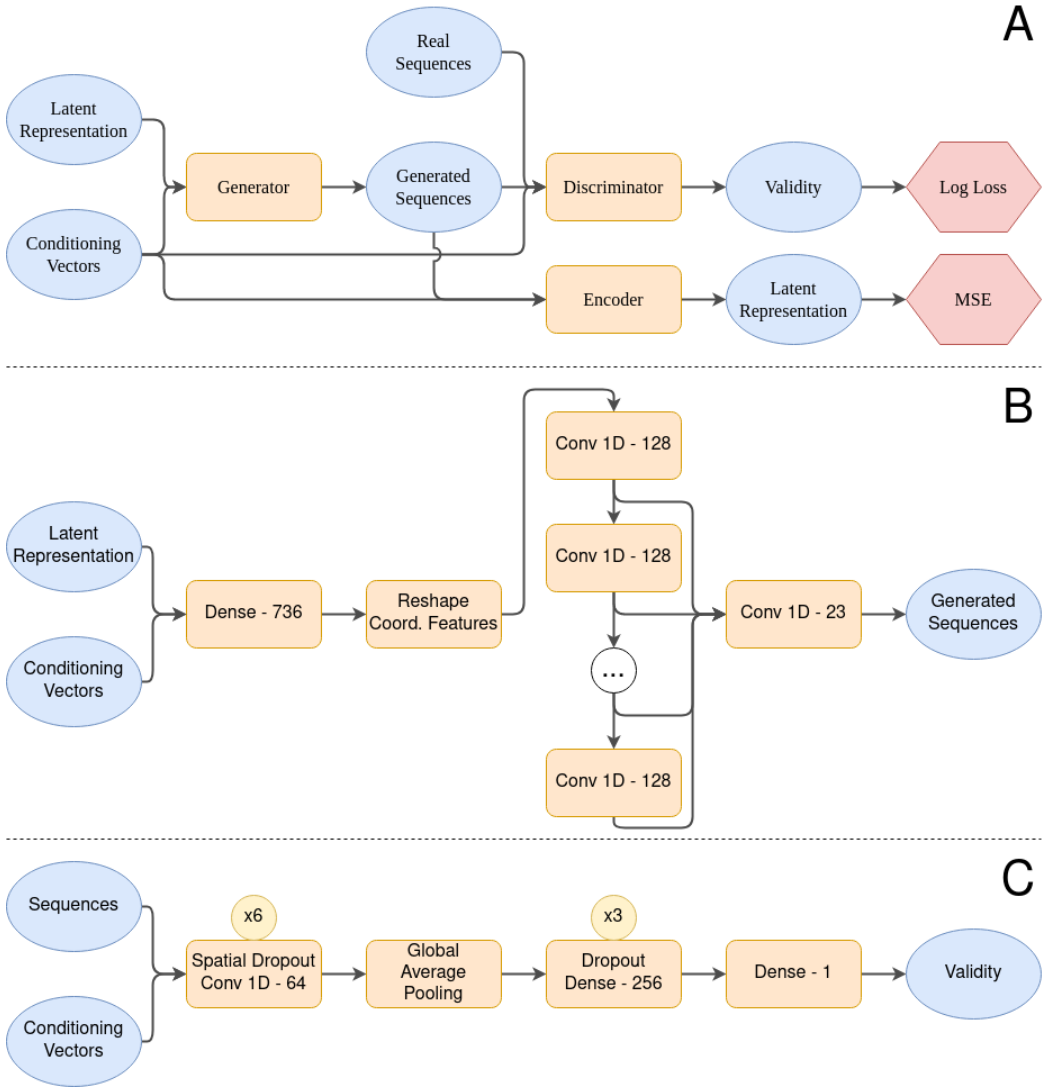

Figure 3.3.2: ***A) AMPGAN v2 Macro-architecture.*** *AMPGAN v2 is a BiCGAN that consists of three networks: the generator, discriminator, and encoder. The discriminator predicts whether a sample is generated or not, and is updated using the log loss. The generator synthesizes samples, and is updated to maximize the loss of the discriminator. The encoder maps sequences into the latent space of the generator, and is trained using the mean squared error (MSE).* ***B) Generator architecture details.*** *We use 6 convolution layers in the central stack, each with a kernel size of 3 and an exponential dilation rate. All dense and convolution layers are followed by a leaky ReLU activation, except the final convolution layer, which has a hyperbolic tangent activation. The final convolution has a kernel size of 1.* ***C) Discriminator architecture details.*** *The convolutions use a filter size of 4 and a stride of 2. All applications of Dropout and Spatial Dropout use a drop rate of 25%. All dense and convolution layers are followed by a leaky ReLU activation, except the final dense layer, which has a sigmoid activation. The condition vectors are tiled and concatenated with the sequences along the features/channels dimension. The encoder uses the same architecture with a different output dimension on the final layer corresponding to the selected latent space dimension and a linear activation function.*

in each epoch. Training proceeded with a batch size of 128 samples, where half were drawn from the AMP set and half from the Non-AMP set. The training signal for the generator and discriminator is provided by the binary crossentropy loss, while the mean squared error is used for the encoder. The discriminator is regularized using a gradient penalty, which has been shown to improve training stability and generalization [202, 165]. In this configuration it takes roughly 30 seconds per epoch, adding up to 16 GPU hours for 2000 epochs using a Nvidia Tesla V100.

AMPGAN v2 builds on our previous experience with AMPGAN v1 [72], though there are several differences in the implementation and evaluation procedure that make direct comparison of the two difficult. Full implementation details for AMPGAN v2 can be found in our GitLab repository [242].

3.4 Results and Discussion

3.4.1 Training Stability

GANs can be difficult to train depending on properties of their architecture and training data. Poor training stability can involve generator mode collapse [208, 32, 227], cyclic generator-discriminator dynamics [208, 202, 165], and vanishing gradients caused by discriminator failures [10, 175].

To investigate the training stability of AMPGAN v2 we trained 30 replicates from scratch using different random initializations. We used a heuristic criteria with two conditions to determine if a trial is successful. First, the model must generate sequences with a character-level entropy that falls between 2 and 4. This removes

models that tend to generate sequences with unrealistically low or high FASTA character diversity. For reference, the average character-level entropy across our training AMPs, non-AMPs, and their combination was ∼2.6, ∼3.43, and ∼3.42 respectively. Second, the model must generate sequences whose length closely matches the value dictated by the conditioning vector. We quantified this by computing the R^2 score over batches of generated sequences, and consider values greater than 0.5 to be successful.

These conditions were selected after observing two common failure modes in the training of AMPGAN v1. The first type were models that correctly handled the dictated sequence length, but only generated sequences composed of one or two amino acids. This resulted in a low character-level entropy, usually close to zero, and these models were clearly ineffective for generating true AMP candidates. The second failure mode resulted in models that produced sequences with more realistic character-level entropy, but completely failed to respond to the dictated sequence length. By not correctly responding to the elements of the conditioning vector, this type of model no longer provides human domain experts with a reliable method for directing the generative process, thus losing one of the primary benefits of the BiCGAN architecture that we have chosen.

We observed three successful trials that led to models with realistic sequence entropy and high correlation between the dictated sequence length and the length of the generated sequence. The other 27 trials failed to produce acceptable models, resulting in a ∼10% training success rate. Figure 3.C.1 summarizes the variance observed during this experiment across several training metrics.

Our training success criterion requires that a successful generator account for the

sequence length provided in the conditioning vector, but there is room for variation between the requirement of $R^2 = 0.5$ and the ideal value of $R^2 = 1.0$. Despite the allowed variance, all three successful trials resulted in models with high R^2 scores– specifically 0.9852, 0.9986, and 0.9975. Qualitatively, this means that almost all the generated sequences have a sequence length that is within ± 3 of the dictated sequence length, which is visualized in Figure 3.D.1.

The observed $\sim$10% training success rate increases the amount of resources required to train new iterations of AMPGAN, relative to a more stable model. Based on the estimate provided in the Design and Training section it will take an average of 160 GPU hours, a little less than a week, to obtain a quality model. However, this can be naively parallelized to reduce the wall clock time to only the 16 hours that it takes to train a single model.

Though it is inconvenient, the low training stability is not a dire issue, since an arbitrary number of AMP candidates can be generated once a quality model has been obtained. Also, It is likely that the training duration can be shortened from 2000 epochs to 1000 epochs, since Figure 3.C.1 indicates that all successful models had passed the criteria by that point.

We briefly investigated the training stability of our model on MNIST, an alternative dataset composed of handwritten digits. The digits were presented as a sequence of pixels, and the conditioning vectors were constructed using the classification labels. Under these conditions we found that our model trains quickly and reliably. This indicates that qualities of the training dataset may be the primary cause, rather than elements of the GAN architecture. We hypothesize that the lower quantity of labelled data and larger conditioning space of our training set (relative to MNIST)

may contribute to the training instability.

3.4.2 Physio-chemical Similarity

To be applicable to AMP design and discovery, we need to evaluate the quality of the generator and the properties of the generated candidates. However, it is prohibitively expensive to experimentally validate the ability of the generator to create sequences that follow the target microbe, target mechanism, and MIC50 values provided in the conditioning vector—so instead we focus on comparisons between easily measurable physio-chemical properties of generated and authentic peptide sequences.

We observe a high similarity between the amino acid distribution of the training and generated AMP sequences, which differ by less than 1% for most of the 20 natural amino acids (Figure 3.F.1). The most significant discrepancies come from Arginine (R) and Lysine (K), which are more prevalent in the generated sequences by 6.3% and 2.2% respectively. In contrast, three non-polar amino acids, Alanine (A) and Leucine (L) are 1.1% and 1.3% more common in the real AMP sequences respectively. Generally, these small differences suggest a consistency between the generated peptides and known AMPs. Figures 3.E.1 and 3.E.2 show additional amino acid distribution comparisons between various groups of peptides.

Figures 3.4.1 only investigates the appearance frequency of single amino acids, but there is a large body of research [24, 152, 79, 253, 197] that suggests peptides feature complex grammatical structure. We investigated this higher-order organization using generalized word shifts [77], which extend the simple analysis done at the character level to sub-sequences of arbitrary length. Word shifts measure the contributions of distinct sub-sequences to a divergence measure between two groups of sequences and

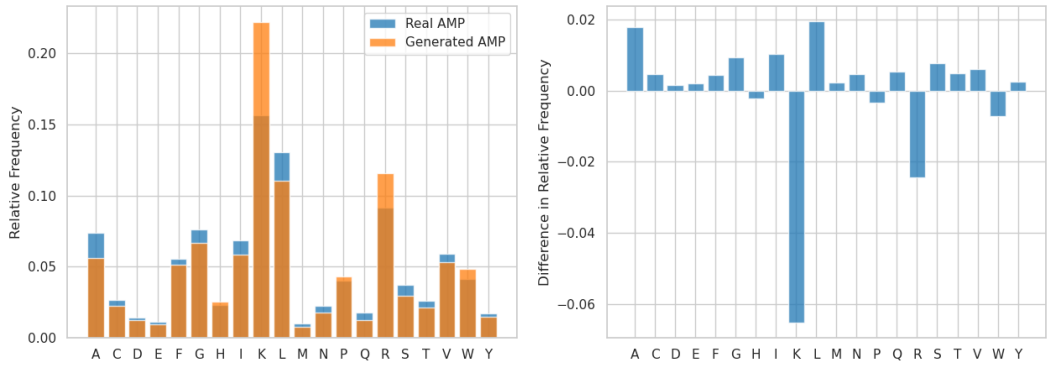

Figure 3.4.1: Distributions of amino acids present in generated vs non-generated AMP sequences. The distributions are layered in the left panel and the difference is shown in the right panel, facilitating different comparison perspectives. The generated distribution was created using 4855 sequences with conditioning vectors drawn at random from the training set. 50% of the conditioning vectors were taken from AMP sequences and 50% from non-AMP sequences. The model used to generate these sequences was arbitrarily selected from the set of successfully trained models. The non-generated distribution was created using a sample of 5120 sequences that were randomly drawn from the training set with a 50%/50% split between AMP and non-AMP sequences. In all comparisons K is the largest outlier, appearing 4–6% more often in generated sequences than real sequences.

highlight the largest contributors.

In Figure 3.4.2, we provide word shifts between generated AMPs and real AMPs for sub-sequences of length 2 and 3. The sub-sequences that were more common in generated peptides mostly involve one or more instances of K or R. Likewise, the sub-sequences that were more common in real peptides tended to involve A or L. These two observations reinforce the results of the character level analysis. Many of the sub-sequences present in both plots feature positive charge or are hydrophobic, which corresponds well with known properties of alpha-helical AMPs. In the length 2 sub-sequence shift, the GP and PG motifs are of particular interest since they are often part of hinge-like structures near bends or kinks in proteins. Figures 3.F.1 and 3.F.2 provide baseline analysis that compares two uniformly randomly constructed samples of sequences using the same tools, which gives additional context for interpreting

Figures 3.4.1 and 3.4.2 respectively.

3.4.3 Sequence Diversity

When proposing candidate AMPs it is important that the generated candidates are diverse as a population and novel relative to known AMPs. If the generator produces sequences with low diversity, it can run into the same sampling problems as the extended predictive models discussed earlier. A generative model will be less useful for discovering new AMPs if it does not produce sequences that are novel relative to known AMPs. We applied the Gotoh global alignment algorithm [91, 41] to quantify the relative similarity of two bags of sequences. The distribution of alignment scores obtained between a pair of bags indicates the relative similarity of the bags, with more similar bags receiving higher scores.

Figure 3.4.3 contains letter-value plots [107] that summarize the scores obtained by comparing the training AMPs, generated sequences, generated AMPs, and generated non-AMPs to themselves (i.e. a measure of diversity). Additionally, the final letter-value plot shows the distribution of global scores obtained by comparing the generated and training AMP sequences.

The training AMP score distribution features much higher median and upper percentile scores than any other distribution under consideration, indicating that there is relatively low sequence diversity in the training AMP set. The median score of 16.55 and mean score of 16.49 indicate a low diversity, especially relative to the generated AMP sequences that feature a median score of 7.83 and a mean score of 7.95. The generated non-AMP sequences feature a similar level of diversity to the AMP sequences, reaching a median score of 7.8 and a mean score of 7.92. The

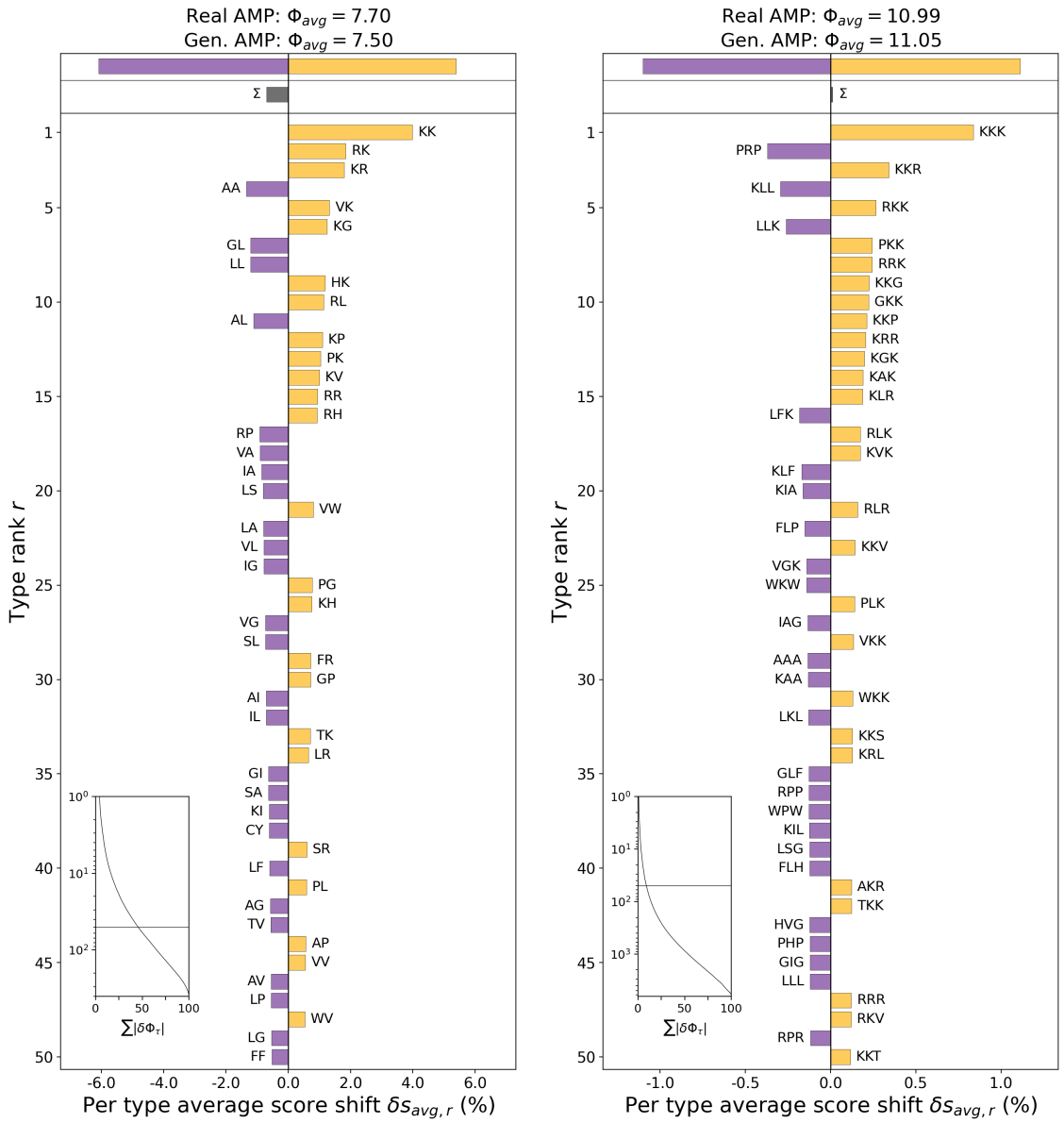

Figure 3.4.2: Shannon's entropy divergence between the distributions of length 2 (left) and length 3 (right) sub-sequences of FASTA characters in AMPs from the training set (real) or AMPs created by the generator (generated). Purple bars indicate a greater prevalence of a particular sub-sequence in real AMPs, while gold bars indicate a greater prevalence in generated AMPs. The two values in the title of each panel indicate the average entropy of each group. For reference, the distribution of sub-sequences drawn from uniformly random sequences results in a maximum entropy of ∼8.64 for length 2 sub-sequences and ∼12.97 for length 3 sub-sequences. Both groups in both plots feature a lower entropy than the maximum, thus we should expect to see meaningful structures in each group. The CDF plot in the lower left corner of each panel indicates that the top 50 contributors to the divergence only account for ∼50% (left) and ∼10% (right) of the total divergence, thus both distributions are extremely flat.

combined set of generated sequences obtains slightly higher scores than either the AMPs or non-AMPs separately, with a median of 8.0 and a mean of 8.17, which may indicate a slight chemical overlap between the two groups or may be due to chance. Comparing the generated AMPs with the training AMPs results in the lowest scores observed, with a median of 5.24 and a mean of 5.54, indicating that the generated AMPs are novel relative to the training AMPs. Figure 3.G.1 provides additional context for interpreting the global alignment scores shown in Figure 3.4.3.

3.4.4 Estimated Antimicrobial Activity

We applied the predictive models developed by Waghu et al. [248] to estimate the probability that sequences generated by AMPGAN will feature antimicrobial activity. This allows us to evaluate the quality of AMPGAN v2 in an absolute sense, ideally all AMP candidates generated by AMPGAN v2 would feature antimicrobial properties, and in a relative sense, by comparing it with AMPGAN v1.

We generated 5000 AMP candidates from AMPGAN v1 and 5000 from AMPGAN v2, then evaluated them using each of the four predictive machine learning models available on the CAMP_{R3} web page. From these predictions we calculated the percentage of sequences that were predicted to have antimicrobial properties, relative to the total number of sequences. Additionally, we estimated a 95% confidence interval for each percentage using bootstrapping. The results of this evaluation are summarized in Table 3.4.1, which shows that AMPGAN v2 strongly outperforms AMPGAN v1 which successfully predicted experimentally validated AMPs.

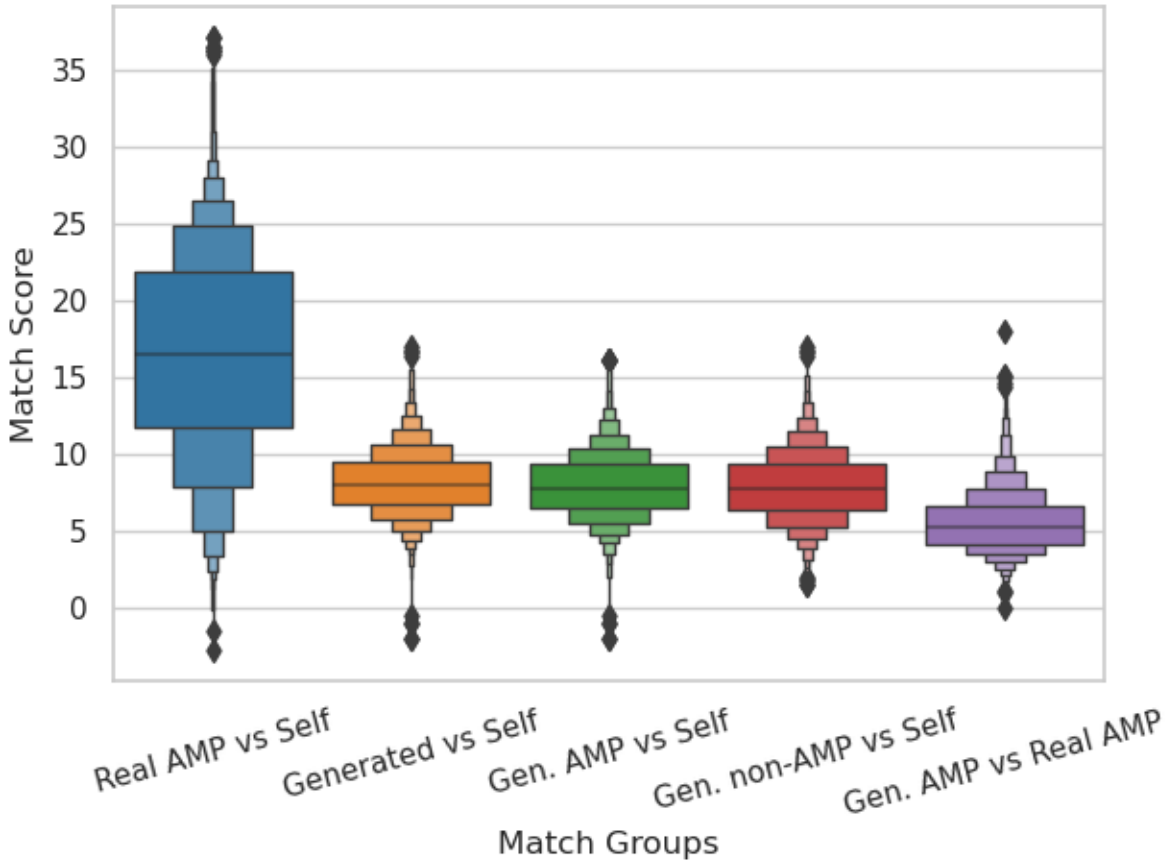

Figure 3.4.3: Letter-value plots showing distributions of match scores obtained from comparisons between different groups of sequences. The central horizontal line in each column denotes the median value. Each box extending from the median line indicates a percentile that is a half step between the starting percentile and the terminal percentile in that direction. For example, starting from the median line, the first box above is terminated at the 75th percentile, halfway between the 50th percentile and the 100th percentile. The diamonds in the tails indicate outliers, which in this case are approximately 5 to 8 of the most extreme values in each tail. The first distribution shows the match scores obtained when comparing the set of training AMPs with itself. The distribution of match scores for training AMPs has a median value that is approximately double that of the distribution for generated AMPs. This indicates that the set of generated AMPs is more diverse than the set of training AMPs. If we compare the generated AMPs directly with the training AMPs, which is shown in the final distribution, we find the lowest median match score observed so far. A low median match score here shows that the generated AMPs are novel relative to the training AMPs.

	AMPGAN v1	AMPGAN v2
Support Vector Machine	5.24% (4.44%, 6.08%)	79.85% (78.27%, 81.39%)
Random Forest	7.66% (6.68%, 8.72%)	88.36% (87.06%, 89.62%)
Artificial Neural Network	4.22% (3.52%, 5.00%)	88.24% (86.94%, 89.46%)
Discriminant Analysis	7.76% (6.76%, 8.72%)	83.71% (82.23%, 85.18%)

Table 3.4.1: Investigation of the expected antimicrobial properties of samples generated by AMPGAN v1 and v2 using the machine learning models developed by Waghu et al. [248]. 5000 AMP candidates were drawn from each generative model and each candidate was evaluated by four predictive models: a support vector machine, a random forest, an artificial neural network, and discriminant analysis. The percentage of generated samples that were predicted to have antimicrobial activity is presented, along with a bootstrapped 95% confidence interval in parenthesis.

3.5 Conclusion

In this work, we introduced AMPGAN v2, a BiCGAN that allows for the controlled generation of peptides with varying degrees of antimicrobial properties. We demonstrate that AMPGAN v2 can be trained successfully using a combination of AMP and non-AMP data. Notably, our data, from extensive comparison between known AMPs and generated peptides, indicates the capacity of AMPGAN v2 to generate sequences that are diverse and novel relative to the training data, but still maintain key AMP features. Additionally, AMPGAN v2 is responsive to changes in the conditioning vector, allowing for effective control of the generative process.

Based on the experimental validation of AMPGAN v1 [72] and the conditional VAE presented by Das et al. [50], we expect the true success rate of AMPGAN v2 to be between 10% and 50%. If that proves to be the case, then AMPGAN v2 represents a fair improvement over the less than 1% success rate of more traditional design methods [54]. Supporting this estimate, sequences generated by AMPGAN v2 were much more likely to be labeled as having antimicrobial properties than sequences

generated by AMPGAN v1, when evaluated by a suite of predictive machine learning models.

AMPGAN v2 has many valuable features, though there are limitations that should be addressed in future work. Specifically, the low training stability of the current system should be improved to reduce training costs. Furthermore, additional validation is needed to ensure that AMPGAN v2 is responsive to manipulations of the target microbe and target mechanism conditioning elements. Greater responsiveness to manipulation of conditioning variables in combination with better training stability will improve designer confidence when developing new AMPs. Finally, additional quantitative methods for evaluating the quality of generative AMP models are needed to aid in development and performance comparisons. We believe that an extension of Fréchet Inception Distance [103] to this domain and the use of Adversarial Accuracy [263] are promising directions to investigate. Along with these faster evaluation methods, we plan to experimentally validate the antimicrobial properties of several AMPGAN v2 designed peptides.

AMPGAN v2 contributes a GAN-based model to an area where non-generative models or VAEs are more prevalent. Additionally, we open source AMPGAN v2 [242], allowing the community to interact with and deploy our tool to design and discover AMPs.

Supporting Information Available: Distributions of conditioning variables, summary of training stability experiment, sequence length correlation figure, additional comparisons of amino acid frequency distributions, sequence analysis baselines, and global alignment score baseline.

Data & Software Availability: All source code for the methods, experiments,

and visualizations presented in this work are available under the MIT license via the project GitLab repository (https://gitlab.com/vail-uvm/amp-gan). All data required to train AMPGAN v2 is present in the GitLab repository, and can be obtained using the Git Large File Storage extension (https://git-lfs.github.com/).

3.6 Acknowledgement

We thank Thayer Alshaabi, Lapo Frati, Gabriel Meyer-Lee, and Ollin Demian Langle Chimal for their helpful discussion and suggestions. Computations were performed on the Vermont Advanced Computing Core, supported in part by NSF award No. OAC-1827314. JMR and JL were partially supported by an NIH R01 award (R01GM129431 to JL) and JBF was supported by an NSF award (CHE-1945394 to JL).

Appendix

3.A Sequence Structure Profile

Amino Acid	Helix	Sheet
A	9.29	3.60
R	5.44	9.86
N	2.87	4.10
D	3.00	2.69
C	2.34	13.5
E	3.16	2.39
Q	2.65	2.59
G	10.0	10.8
H	2.38	1.54
I	7.13	4.59
L	10.9	4.69
K	12.8	5.87
M	1.42	0.88

F	4.98	3.70
P	3.14	4.16
S	5.41	6.42
T	3.39	5.41
W	1.87	2.00
Y	1.71	5.18
V	6.18	6.03

Table 3.A.1: Percentage of each amino acid's presence in the respective structure type. A completely random ordering should result in a table of 5% for all positions.

3.B Conditioning Information Distributions

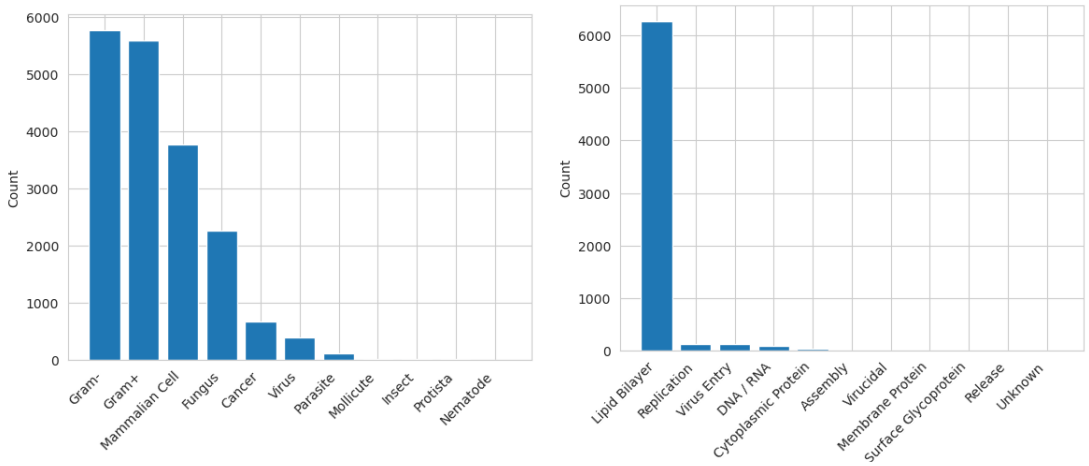

Figure 3.B.1: Label frequency for the target microbe (Left) and target mechanism (Right) conditioning variables.

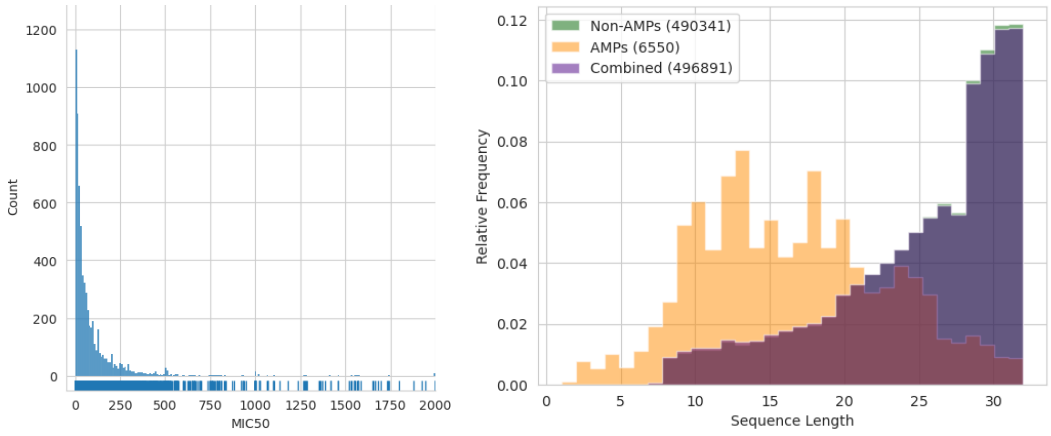

Figure 3.B.2: The distribution of MIC50 values before discretization (Left) and peptide sequence lengths (Right). 27 samples with MIC50 values greater than 2000 were truncated to ease inspection of the rest of the distribution.

3.C Training Stability

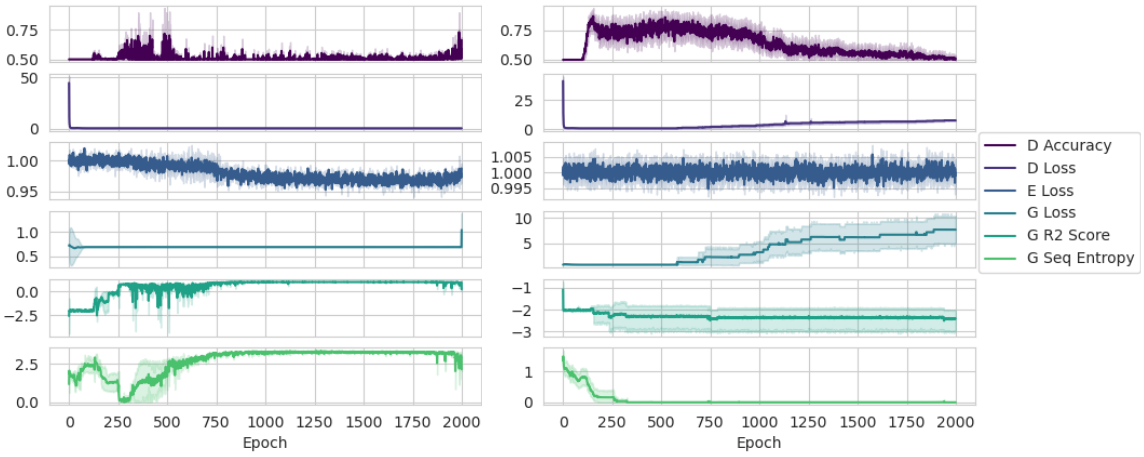

Figure 3.C.1: Investigation of training stability, summarizing the results of 30 independent trials. The left panel was constructed using the successful trials (3/30) and the right panel was constructed with the failed trials (27/30). From top to bottom the panels display the classification accuracy of the discriminator, the discriminator loss (log loss), the encoder loss (MSE), the generator loss (log loss), the R^2 score between the length dictated by the conditioning vector and generated sequences, and the average character-level entropy calculated over batches of generated sequences. This experiment highlights the relative instability of AMPGAN v2, with a success rate of ∼10%.

3.D Sequence Length Correlation

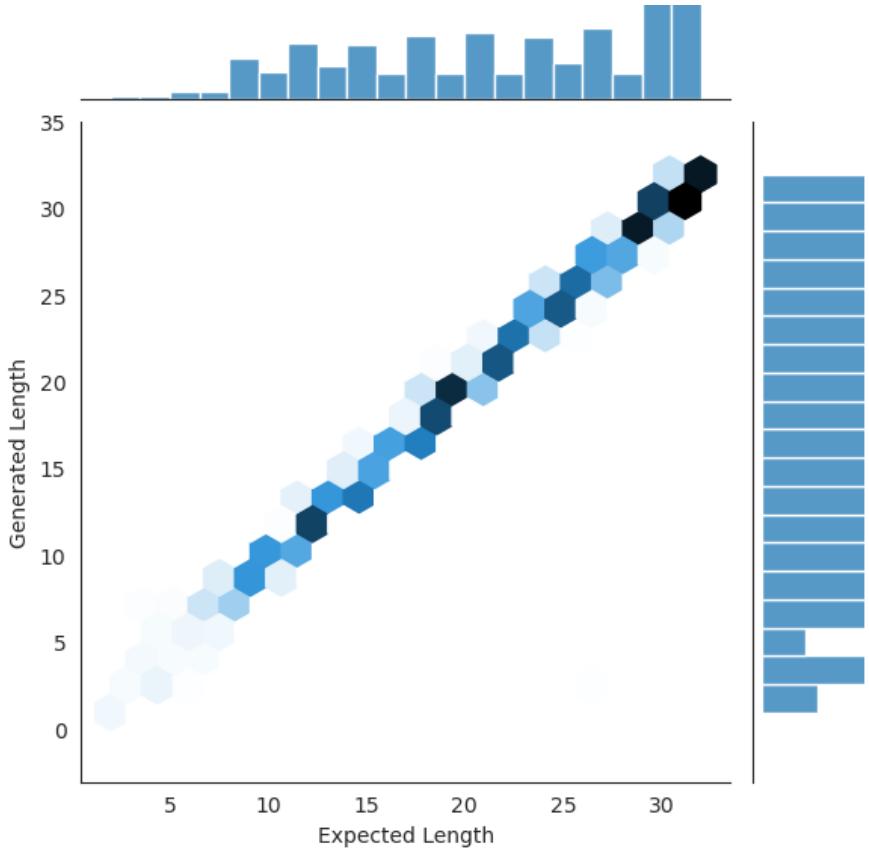

Figure 3.D.1: Agreement between the sequence length dictated by the conditioning vector and the length of sequences produced by the generator. This figure was created using 4855 sequences that were generated using conditioning vectors drawn at random from the training set. 50% of the conditioning vectors were taken from AMP sequences and 50% from non-AMP sequences. The model used to generate these sequences was arbitrarily selected from the set of successfully trained models. The generator pays close attention to the sequence length conditioning variable, resulting in an R^2 score of 0.9798.

Figure 3.E.1: Distribution of amino acids used in generated vs non-generated sequences. Similar to Figure 3.4.1, but shows the distributions for all sequences (top) and non-AMP sequences (bottom). K remains the largest outlier, appearing 4–6% more often in generated sequences than real sequences.

3.E Amino Acid Distribution Comparisons

Figures 3.4.1 and 3.E.1 only compare real and generated distributions, however, the relationship between AMPs and non-AMPs within each group is also important. The generator may create AMP sequences that are similar to real AMP sequences and non-AMP sequences that are similar to real non-AMP sequences, but fail to adequately capture the relationship between AMP and non-AMP sequences. To investigate this we create two additional comparisons between AMP and non-AMP sequences in both real and generated groups (Figure 3.E.2). Though there are some slight deviations present in the individual distributions in the panels on the left, which were already identified in Figure 3.4.1, the differences shown in the panels on the right are nearly identical. This indicates that the generator has learned the relative relationship between AMP and non-AMP sequences, despite some slight biases in its understanding of those distributions individually. Additionally, the lower right panel of Figure 3.E.2 is directly comparable to Figure 3 from Das et al. [51], which agrees with the signs of the relative changes shown here for all amino acids except F and G. This qualitative agreement may indicate that both models have accurately captured the qualities of the training data distribution, or at least that both models acquired a similar bias profile.

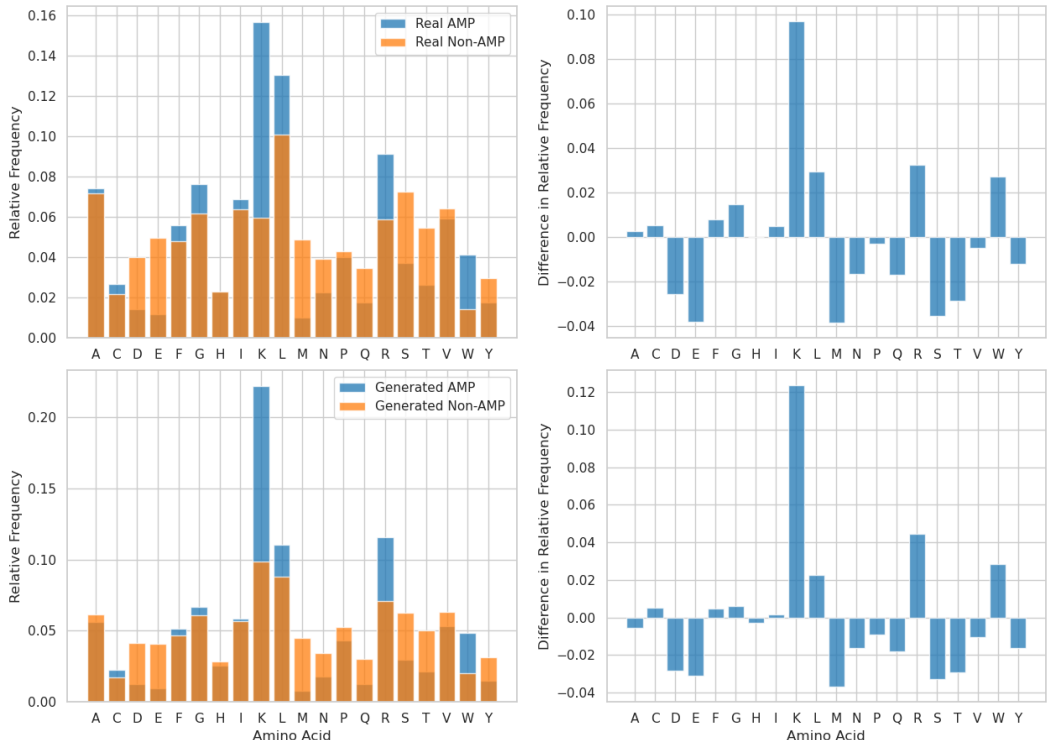

Figure 3.E.2: Amino acid usage frequency distributions for generated AMP and generated Non-AMP sequences (left) along with the difference between the two distributions (right). Comparisons are made between real (top) and generated (bottom) groups.

3.F Sequence Analysis Random Baselines

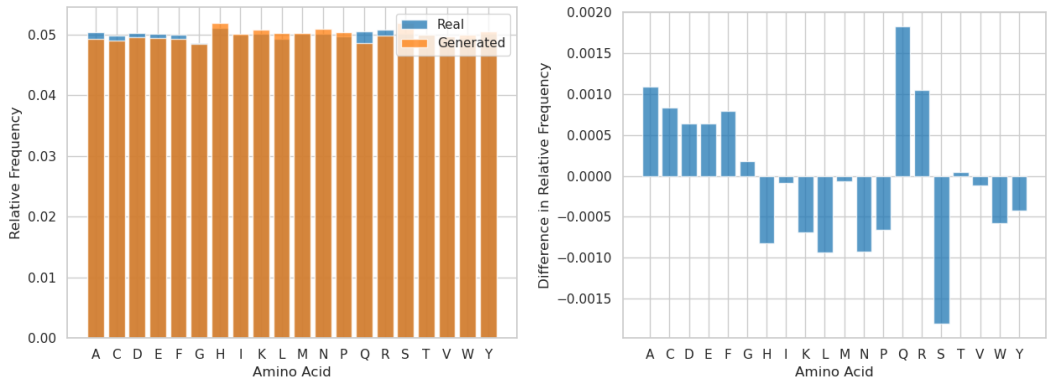

Figure 3.F.1: Amino acid frequency distribution comparison between two independent groups of 5000 uniformly randomly constructed sequences with a maximum length of 32. The distributions are flat, excluding a small amount of sampling noise. Additionally, the deviation between the two is extremely small, with the largest difference value being several orders of magnitude smaller than the largest value present in Figures 3.4.1 or 3.E.2.

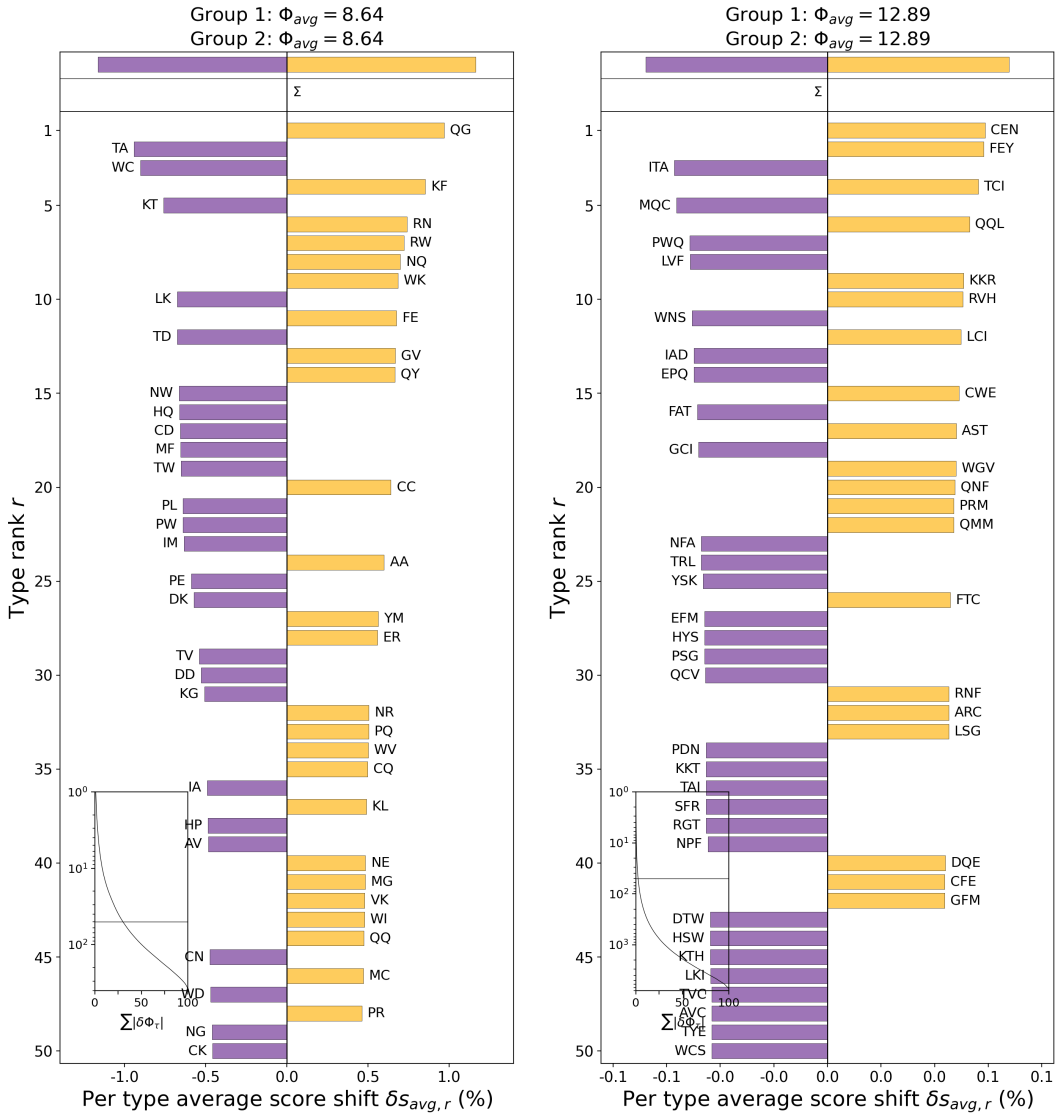

Figure 3.F.2: Word shift plots comparing two independent groups of 5000 uniformly randomly constructed sequences with a maximum length of 32. Similar to the character level analysis shown in Figure 3.F.1, these word shifts are extremely flat. However, since the number of distinct elements grows exponentially with the sub-sequence length, sampling error may have a larger impact here. The maximum entropy for length 2 sub-sequences constructed from the 20 common amino acids is ∼8.64, which is reliably obtained by a sample of this size. The maximum entropy for length 3 sub-sequences is ∼12.97, but is not reached due to sampling error. Approximately 1 to 5 length 3 sub-sequences are unobserved in a sample of this size. There are 400 unique length 2 and 8000 unique length 3 sub-sequences, thus a uniform distribution over those sets has an element-wise probability of 0.0025 and 0.000125 respectively.

3.G Global Sequence Alignment Scores

To investigate the similarity of two bags of sequences we applied the Gotoh global alignment algorithm [91]. We use the implementation provided by Biopython's `PairwiseAligner` object [41], configured with the `BLOSUM62` substitution matrix, an open gap score of -10, and an extend gap score of -1.

Interpreting global alignment scores can be difficult, so we performed a Monte Carlo experiment to uncover information about the distribution of scores in particular circumstances. Specifically, we construct two bags of random sequences, called S_1 and S_2, containing N and $N/2$ sequences respectively. These sequences have a uniformly random length selected from 1 to 32, and the elements of each sequence are uniformly randomly selected from the 20 common amino acids. Next, we construct a new bag, S_3, by combining S_2 with a duplicate. Thus, S_1 and S_3 both contain N sequences, where all sequences in S_1 are likely to be unique and S_3 contains two copies of every unique sequence in S_2. Next, we construct S_4 by randomly mixing the sequences of S_1 and S_3 using a control parameter $m \in [0, 1]$. This mixing is implemented by iterating pairwise over the sequences of S_1 and S_3, then iterating pairwise over the FASTA characters of those sequences. The characters from S_1 are selected with probability $1 - m$, and the characters from S_3 are selected with probability m. From this construction, the mixing parameter directly controls the diversity of S_4, providing a stochastic interpolation between relatively high and low diversity bags of sequences. Finally, pairwise scoring is computed between S_4 and itself using the system described above.

The full experiment then involved sampling the mixture parameter at 50 evenly

spaced points that span the interval $[0, 1]$ and executing 30 replicates of the scoring procedure at each. All replicates use $N = 1000$. Figure 3.G.1 summarizes this experiment, where the orange line indicates the mean match score across the replicates, and the shaded area covers plus or minus one standard deviation. Percentile information is shown by the blue lines, and extreme values are shown with grey markers. This indicates a roughly exponential scaling in the expected match score as the mixing parameter varies from 0 to 1, ranging from $\sim$2.5 to $\sim$20. The standard deviation indicates the existence of heteroscedasticity, where the tails of the score distribution spread out as the mixture parameter increases in value.

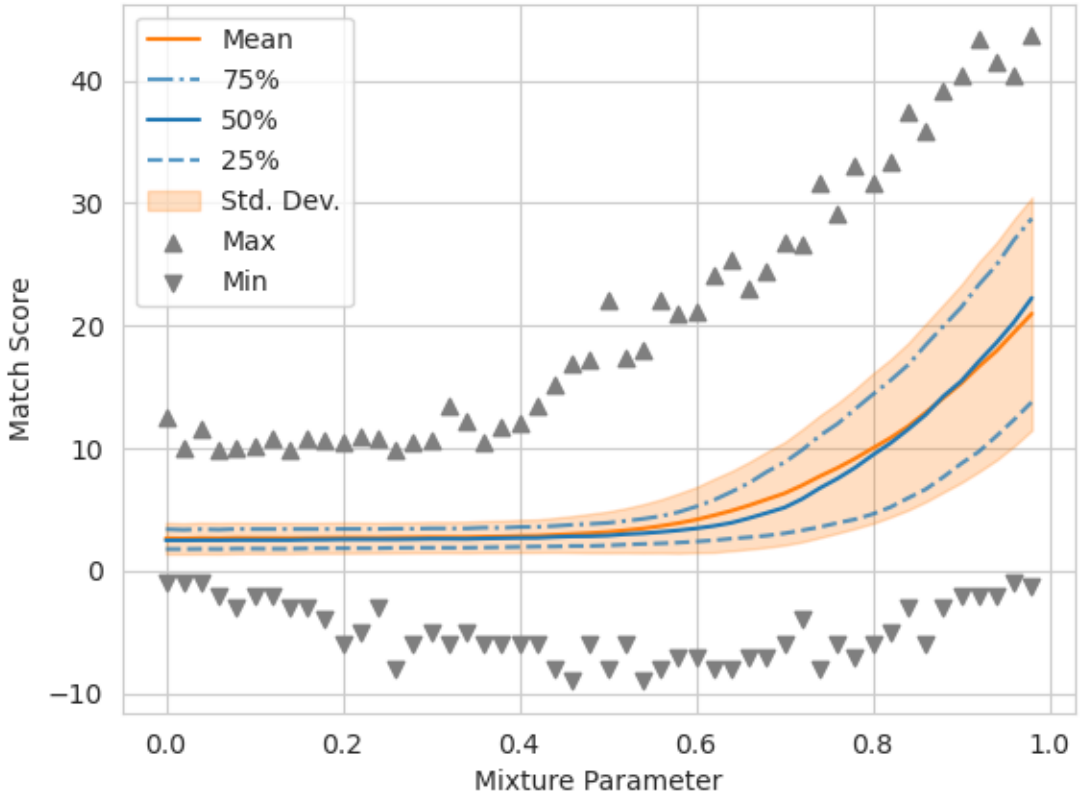

Figure 3.G.1: Distributions of global match scores between a bag of FASTA sequences and itself. The dark line indicates the mean match score, and the shaded area indicates plus or minus one standard deviation. The horizontal axis corresponds with a mixture parameter that controls the level of diversity in the bag. For low values the bag of sequences is composed entirely of unique sequences, resulting in low match scores on average. The value of the mixture parameter increases as the level of diversity in the bag decreases. When the mixture parameter reaches a value of 1.0 the bag contains an exact duplicate for every sequence, resulting in match scores in the 30s.

Chapter 4

Adaptive Agents and Data Quality in Agent-Based Financial Markets

This Chapter is derived from Van Oort, Tivnan, and Wshah [244].

4.1 Abstract

We present our Agent-Based Market Microstructure Simulation (ABMMS), an Agent-Based Financial Market (ABFM) that captures much of the complexity present in the US National Market System for equities (NMS). Agent-Based models are a natural choice for understanding financial markets. Financial markets feature a constrained space of agent interactions that should simplify model creation, produce a wealth of data that should aid model development, and a successful ABFM could improve the evaluation of design and policy decisions. Despite these advantages, ABFMs have

largely remained an academic novelty. We hypothesize that two factors limit the usefulness of ABFMs. First, many ABFMs fail to capture relevant microstructure mechanisms, leading to differences in the mechanics of trading. Second, the simple agents that commonly populate ABFMs do not display the breadth of behaviors observed in human traders or the trading systems that they create. We investigate these issues through the development of ABMMS, which features a fragmented market structure, communication infrastructure with propagation delays, realistic auction mechanisms, and more. As a baseline, we populate ABMMS with simple trading agents and investigate properties of the generated data. We then compare the baseline with experimental conditions that explore the impacts of a simplified market topology or a meta-reinforcement learning agent. The combination of detailed market mechanisms and adaptive agents leads to models whose generated data more accurately reproduce stylized facts observed in actual markets. These improvements increase the utility of ABFMs as tools to inform design and policy decisions.

4.2 Introduction

Decades of market microstructure research have shown that the mechanics of trading meaningfully impact price formation processes [179, 154, 100]. Price formation processes generated by agent-based financial markets (ABFMs) are similarly impacted by their market architecture. Thus, ABFMs that fail to capture major market microstructure mechanisms present in their target systems may observe divergences in behaviors and outcomes.

The ecology of agents that populate an ABFM is just as important as the market

infrastructure that mediates their interactions [138]. Zero intelligence (ZI) agents [83], and other simple agents, have been heavily used in market microstructure research to understand baseline characteristics of markets [221, 69, 139, 45, 92]. However, simple agents do not exhibit the heterogeneity of strategies observed in real markets or the adaptability of real market participants.

In real markets, it is common for short-term trading strategies to lose effectiveness over time, a phenomenon that is referred to as alpha decay [57]. By some estimates, short-term strategies take 3 to 7 months to develop and remain effective for 3 to 4 months [209]. Since the average development duration is longer than the average strategy lifetime, we might expect the population of short-term strategies to have a high turnover rate. This high turnover rate may be a driving mechanism behind the non-stationarity of trading dynamics. It also indicates that strategy adaptation is a critical attribute of successful market participants and that static strategies may be poorly suited for realistic ABFMs.

The use of adaptive strategies in ABFMs can promote agent specialization, leading to emergent heterogeneity. Since agent heterogeneity contributes to financial market resilience [19], this emergent heterogeneity developed by adaptive agents could improve the resilience of ABFMs. Additionally, agent adaptability is critical to realize economic rationality in non-trivial ABFMs [245, 99, 158]. Economically rational agents avoid using trading strategies that lead to financial ruin. Thus, when agents with generally fixed strategies encounter unfavorable market conditions they may be forced to exit the market if they are unable to adapt their strategy sufficiently. This in turn can lead to complete failure of a simulated marketplace. Observed deviations from the Efficient Markets Hypothesis [156, 27] and the rise of the Adaptive

Markets Hypothesis [151] indicate a growing realization of the importance of agent adaptability in financial markets.

In this paper, we present our Agent-Based Market Microstructure Simulation (ABMMS), an ABFM with detailed market mechanisms and agent adaptability as core design principles. We evaluate ABMMS under different configurations to determine the impacts of market fragmentation and adaptive agents on the quality of generated data. Our evaluation procedure is built using stylized facts and analytical methods developed by the econometrics, market microstructure, and ABFM communities. ABMMS can reproduce several stylized facts of asset prices, along with other features of realistic market data, and thus may be more suitable to inform system design or policy than simpler ABFMs.

4.3 Related Work

4.3.1 Market Infrastructure in the National Market System

ABMMS targets the US National Market System for equities (NMS), and many design decisions were based on this choice of target system. To provide the appropriate context for understanding our model, we summarize the market infrastructure present in the NMS and indicate references with additional details.

Trading in the NMS occurs in a fragmented market that consists of 16 securities exchanges, which manage a set of continuous double auctions (CDAs) to support trading for a corresponding set of stocks. A CDA allows traders to submit orders to

buy (bid) or sell (offer) at any time and processes them upon receipt. Orders that cannot be fulfilled immediately are collected in a Limit Order Book (LOB). Almost all CDAs prioritize order execution based on price and time, though some may use additional attributes. For additional details regarding CDAs or LOBs, see one or more of Smith et al. [221], Gould et al. [92], and Abergel et al. [1], and Friedman [75].

The 16 exchanges that form the NMS are housed within at least four data centers in northern New Jersey [234]. These data centers are connected by an Electronic Communication Network (ECN) that is implemented with a combination of fiber optic technology and wireless alternatives [178].

The use of continuous trading mechanisms causes a race to react any time new public information is released. The speed of light guarantees the existence of propagation delays on each leg of an ECN, an average of 100 microseconds based on the current configuration of the NMS [234]. Optimized trading algorithms on specialized hardware may only take between tens of nanoseconds to a few microseconds to react to incoming messages. The combination of these three properties means that the propagation delays imposed by the topology and geometry of the ECN can have an immense impact on trading outcomes. See Section 3 of Tivnan et al. [234] for additional details regarding the organization of the NMS.

Beyond the mechanical details mentioned above, the regulatory environment that surrounds the NMS plays an important role in shaping the market infrastructure and agent behaviors. Readers interested in understanding key NMS regulations should refer to Appendix 3 from Tivnan et al. [234] for an overview, or the regulation itself for details [216].

4.3.2 Market Infrastructure in Prior ABFMs

Many ABFMs implement a simple infrastructure that allows clear emphasis to be placed on specific elements, while also reducing computational costs and allowing for rapid experimentation. For example, Wah, Wright, and Wellman [250] study a population of heterogeneous agents trading a single asset in a single continuous double auction (CDA). The simplicity of the infrastructure emphasizes the heterogeneous agents, the quality of their interactions, and differences in their outcomes.

It is possible to model financial markets without agents or an explicit market microstructure. Equation-Based Models (EBMs) boil down all activity to a set of mathematical equations, commonly differential equations, that describe macro-level quantities, such as asset prices [206]. However, abstracting away these details can restrict or eliminate the possibility of emergent phenomena, greatly reducing the expressiveness of a model.

Early modeling efforts focused on simpler market architectures, such as Walrasian auctions and dealer markets [179, 100]. But most modern equity markets feature a fragmented CDA with trading activity distributed across multiple locations.

Some have used ABFMs to investigate the impacts of market fragmentation, usually focusing on the simplest case involving two auctions [249, 64, 11]. To simulate fragmented markets these ABFMs must account for communication latency, otherwise, the fragmentation would not have a material impact on trading activity. When modeling fragmented markets, it is common to implement one or more securities information processors (SIPs) [235]. SIPs serve as data aggregators that disseminate important signals to keep prices synchronized in a fragmented market, such as the

National Best Bid and Offer (NBBO), an indicator of market-wide best price.

In addition to market fragmentation, which occurs at the level of financial exchanges, some have explored ABFMs that capture the interactions that occur in multi-level markets containing many interconnected financial systems, such as equity markets, options markets, brokerages, and banks [20].

Speed can be a deciding factor in the competition of trading strategies, especially in market systems with continuous auction mechanisms. An often underappreciated element of this competition is response delays, the time it takes a trading strategy to ingest an incoming market message and issue an appropriate response. These response delays are often so small that they are assumed to have minimal impact on trading outcomes, and thus are not implemented in many ABFMs. However, since many aspects of the race for speed have been commoditized (e.g., colocation, wireless communication channels, specialized computing hardware, etc.) the microseconds that can be shaved via software optimization can have serious impacts [200].

There are an endless number of market infrastructure details that can impact trading processes, and should be captured in detailed models. However, in this work, we focus exclusively on a stock market with detailed implementations of market fragmentation, communication infrastructure, auction mechanisms, and adaptive agents.

4.3.3 Adaptive Agents

Mechanisms for adaptive agents can be classified as active or passive [140]. Active learning is driven by the intentional change of an agent's strategy, while passive learning occurs via the accumulation of wealth by more effective strategies over time. We focus on active learning due to recent advances in the field of machine learning, as well

as the potential relationship between active learning and economic rationality [245].

Active adaptive agents can feature two types of strategies, fixed or free form. Fixed strategies cover a single qualitative class of behaviors and tune a set of parameters to optimize profits or adapt to changing market conditions. Despite their ability to modify certain aspects of their behavior, such as interaction frequency or pricing beliefs, fixed strategies cannot spontaneously adopt qualitatively distinct strategies. On the other hand, free form strategies can implement two or more classes of behavior, and perhaps even develop new strategies on the fly. Since the behavior of fixed strategies is more constrained than free form strategies, they tend to be simpler to develop and understand.

Fixed Strategy Agents

The ZI agents introduced by Gode and Sunder [83] are simple and broadly applicable, which lead to a proliferation of applications and sparked a vein of research that has been developed for decades. ZI Plus (ZIP) agents, like the ZI agents that inspired them, take stochastic actions using minimal information but develop pricing beliefs over time, based on bid and offer prices lead to trades [40, 38, 193, 39]. The agents created by Gjerstad and Dickhaut [81] (GD) have a similar structure to ZIP agents, they develop price belief functions based on quotes and trades. However, GD agents take actions that greedily maximize surplus, whereas ZIP agents do not directly optimize profits. By covering some pathological edge cases in the GD algorithm, Modified GD (MGD) agents [232] avoid excessive volatility and outperform their predecessor. The GDX strategy [231] also builds on the pricing belief functions seen in GD agents but accounts for future rewards via dynamic programming. This forward-looking op-

timization promotes longer-term strategies with more interesting behavior. Adaptive Aggressiveness (AA) agents [247] combine price belief functions with an aggression function that allows them to strategically account for their "desire to trade". Taking a slightly different approach, Assignment Adaptive (ASAD) agents [228] use a relatively simple strategy that is less adaptive in some ways than ZIP agents but explicitly accounts for the information contained in an agent's submitted orders. ASAD agents can generate interesting dynamics, especially when reacting to exogenous price shocks in a homogeneous strategy space, but are generally outclassed by ZIP agents when in direct competition.

This line of research has created several relatively simple agents that combine domain knowledge with basic machine learning and optimization techniques, resulting in adaptive, but fairly restricted strategies. Through the use of more advanced machine learning techniques, removing imposed strategy structure, and allowing for greater strategy complexity, we can create agents that develop qualitatively distinct strategies.

Free Form Strategy Agents

Free form strategies are constructed around a behavior adaptation mechanism, commonly implemented using machine learning, that allows the agent to respond appropriately to changing market conditions.

Supervised learning techniques, in the form of imitation learning, can replicate observed patterns in order flow data [29, 220, 257]. However, agents built with imitation learning tend to regurgitate observed behaviors, and thus have little ability to respond to market conditions that were not observed during training or to generate new strate-

gies. Generative Adversarial Networks (GANs) can create realistic streams of order flow in a similar manner to imitation learning based methods. GANs may be better than imitation learning at generating novel content, due to the adversarial learning mechanism, but still lack a mechanism necessary for effective generalization [145].

One of the most obvious learning signals present in financial markets is profit. Profit motive is relied on as one of the fundamental forces in financial markets, and it makes intuitive sense to train trading strategies with it. Two classes of algorithms are particularly effective at deriving appropriate behavior from arbitrary reward signals: meta-heuristic search and reinforcement learning. Both meta-heuristic search [229, 110] and reinforcement learning [212, 53] have been applied repeatedly, and with varying degrees of success, to the learning of trading strategies.

Many traditional applications of meta-heuristic search and reinforcement learning focused on narrowly defined problems and did not emphasize the ability to adapt to dynamic environments. The rise of meta-learning, commonly described as learning to learn, has greatly improved the ability of machine learning models to learn from, and adapt to, more broadly defined problems [109]. Trading agents developed using meta-learning techniques, such a hierarchical reinforcement learning [230] or meta-learned evolutionary strategies [225], can learn more quickly, display higher peak performance, and handle new market conditions more gracefully than agents developed without meta-learning.

4.3.4 Model Examination

For an ABFM to be a useful tool for informing policy or system design, it must satisfy three properties. First, the model must align with the system that it is intended

to influence. Second, the model must provide useful insights into that target system. Third, the model must garner a certain amount of trust from policymakers and designers that control the target system.

Figure 4.3.1 summarizes the model development pipeline, including ABFM development, which is driven by three processes that ensure quality and consistency: verification, validation, and replication [258, 196, 9]. During verification, an implemented model is compared and contrasted with a conceptual model. Good software testing and debugging are core verification tasks, though visual inspection of model outputs and other similar actions also play a role. Validation compares an implemented model with the target system via iterative calibration, which involves tuning free model parameters so that data generated by the model resembles data from the target system. Comprehensive validation and verification, along with clear communication, establish a baseline level of trust in a model. Replication, which is a collection of tasks ranging from running code provided by the creators of a model to complete re-implementation, can further bolster the reputation of a model. The primary goal of replication is to ensure that the outputs of the model display the advertised properties, and are not the result of spurious factors.

Model validation can be driven by data collected from the target system, stylized facts that have been developed based on quantitative observation of the target system, or other forms of distilled knowledge. In most cases, this decision is based on data availability. For example, it is prohibitively costly to obtain high frequency data from all of the exchanges in the NMS. Like many who have come before us [78, 19, 163], we validate our model using stylized facts of asset price time series [44], order book metrics [186], and dislocations [234], instead of depth-of-book data.

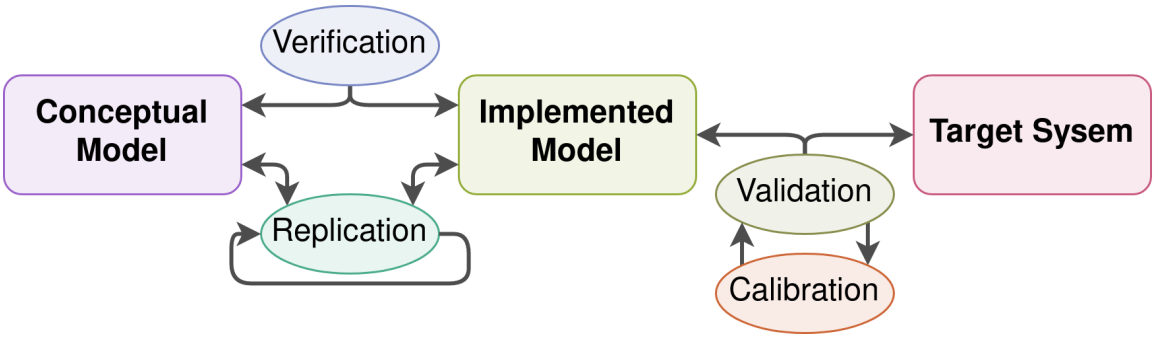

Figure 4.3.1: The model development process, which includes ABFM development, involves three entities (the conceptual model, implemented model, and target system) connected by four processes (verification, validation, calibration, and replication).

4.4 Methods

4.4.1 Market Infrastructure in ABMMS

We developed ABMMS, a highly configurable ABFM that targets the US National Market System for equities (NMS), to investigate the impacts of market microstructure and adaptive agents. ABMMS emphasizes the explicit representation of many market microstructure elements, starting with a realistic electronic communication network (ECN). The ECN consists of a queue of in-flight messages and a topology that those messages travel over. The topology is an undirected graph with weighted edges, where nodes are data centers, edges are communication channels, and edge weights are deterministic propagation delays. Messages are routed based on the shortest weighted path, identified via Dijkstra's algorithm. Exponential noise with a mean of 5 microseconds is added to the propagation delay to simulate latency jitter and other stochastic delays.

Figure 4.4.1 shows the default configuration for ABMMS, which is derived from

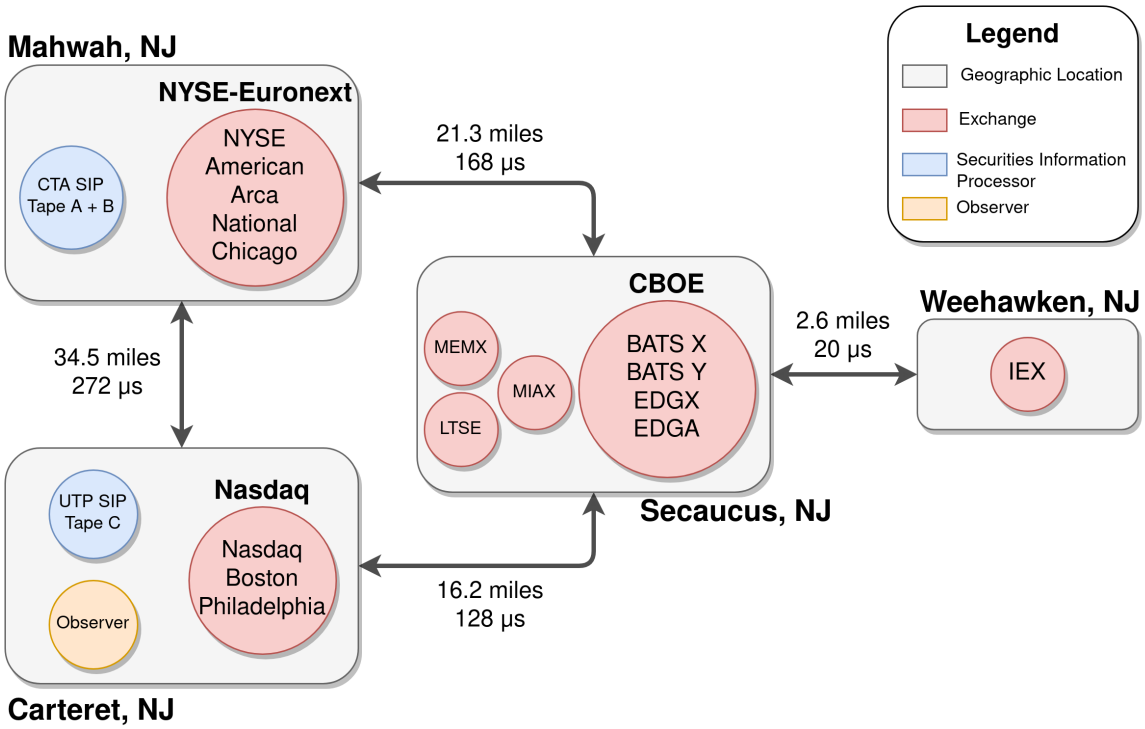

Figure 4.4.1: A visual summary of the default configuration of ABMMS. The topology and propagation delays are adopted from Tivnan et al. [234], with four data centers distributed across northern New Jersey. The choice of 16 exchanges and 2 SIPS is based on our understanding of the NMS in early 2021. Traders are randomly distributed across the four data centers unless otherwise noted. Every configuration of ABMMS has an observer, located at the Carteret node, that exports data from the simulation for analysis.

the state of the NMS in early 2021 and adopts the propagation delays presented by Tivnan et al. [234]. ABMMS uses a discrete-event scheduler to process messages passed between agents via the ECN, capturing the temporal heterogeneity of market events and agent response times. Messages are processed sequentially based on the time they should arrive at their recipient, resulting in a dynamic step size for the simulation clock. Exchanges are distributed across the nodes of the ECN, where each exchange manages a CDA for each actively traded stock. All CDAs in ABMMS prioritize the execution of orders based on price, visibility, and time, with ties broken randomly. Each Exchange implements a fee schedule that includes market access fees, also known as maker-taker fees, which incentivize liquidity demand or supply depending on the configuration. The default configuration of ABMMS includes a pair of SIPs that construct and disseminate NBBOs, LULD bands, and TAQ feeds. See Appendix 4.A for an in depth description of ABMMS following the Overview, Design concepts, Details (ODD) protocol [93, 94, 95].

When compared with previous ABFMs, ABMMS implements several market elements that are usually abstracted away, and have never been investigated simultaneously in a single model. Specifically, CDAs to facilitate trading, multiple assets traded simultaneously, market fragmentation beyond two exchanges, SIPs that issue NBBOs as well as LULD bands, trade-through protection, common order modifiers (hidden, immediate-or-cancel, all-or-nothing, inter-market sweep), and market access fees. There is an expectation of emergent phenomena in ABFMs, thus the inclusion of these additional details may have non-trivial impacts on market dynamics, especially if leveraged strategically by a learning agent.

4.4.2 Traders

We developed our adaptive trading agent using meta-reinforcement learning [62, 251]. Meta-reinforcement learning is better able to adapt to dynamic environments than traditional reinforcement learning, and financial markets are extraordinarily dynamic. One mechanism that causes meta-reinforcement learning to foster adaptability is the use of effective experimentation processes. Meta-reinforcement learning agents can actively investigate the state of their environment and incorporate that information into their decision process [52].

Given the importance of agent adaptability and heterogeneity discussed earlier, one approach to developing reinforcement learning traders might populate a simulation entirely with such reinforcement learning traders to develop a population of co-adapted strategies. However, multi-agent reinforcement learning is unstable [28]. With each agent adapting in real-time, the optimal strategy for all agents becomes a moving goal that is difficult to approach. Instead, we focus on the impact of a single reinforcement learning agent in simulations otherwise populated with simple agents. For this purpose, we select ZIP agents, since they have a long history of effective applications in ABFMs and have not been bested by another simple strategy [200].

We develop our ZIP traders based on the reference implementation provided by Cliff [39], with one minor deviation. The original implementation of ZIP traders uses an exogenous stream of limit prices as a basis for the pricing beliefs of each agent. We replace this exogenous input with random limit prices drawn from a truncated normal distribution that is centered at the Limit Up-Limit Down (LULD) reference price, covers the LULD interval, and is updated each time a new LULD band is issued.

For more details, see Appendix 4.A.11.

4.4.3 Stylized Facts

The econometrics community has been developing stylized facts that capture various features of data generated by financial markets since the mid-'90s, if not earlier [187, 44, 21, 191, 218]. Stylized facts are statistical properties that are observed across a broad range of assets, markets, and periods. Stylized facts are qualitative and trade precision in favor of generality, thus there can be exceptions. However, through the combination of many stylized facts, it becomes possible to identify data that has been generated by authentic trading processes.

We focus on the eleven stylized facts outlined by Cont [44] since they are relatively simple to test for with moderate amounts of data. However, three facts (#1: Absence of Linear Auto-correlation, #4: Aggregational Gaussianity, and #11: Asymmetry in Time Scales) require longer periods of coarser-grained data, which are costly to generate with a model that operates at high frequencies. We eschew the three problematic facts and rely on the remaining eight to validate ABMMS.

The stylized facts described by Cont [44] are exclusively concerned with properties of asset price time series. However, ABMMS produces much more information than asset price time series. In particular, we have access to a complete depth-of-book feed, thus metrics that investigate limit order book properties [21, 191, 218, 186] can help to quantify the impacts of our meta-reinforcement learning trader.

4.5 Results

To ensure that our tests for stylized facts are effective, we calibrate them on easily accessible historical price data. Specifically, we use minute resolution data obtained from Alpha Vantage [114] for 30 US stocks: AAPL, AXP, BA, CAT, CSCO, CVX, DD, DIS, GE, GS, HD, IBM, INTC, JNJ, JPM, KO, MCD, MMM, MRK, MSFT, NKE, PFE, PG, RTX, TRV, UNH, V, VZ, WMT, and XOM. The data for most symbols covers two years of trading, roughly from April 2019 through February 2021. RTX was formed as the result of a merger in 2020, so we only have data from April 2020 through May 2021 (roughly 14 months worth of trading data). The Alpha Vantage data is built using SIP feeds and aggregated at the minute level, with open, high, low, close, and volume features. The data is adjusted to account for splits and dividends. See the Alpha Vantage API documentation for more details [115].

Using this data, we calibrate our stylized fact tests by optimizing free parameters to improve the detection rate. This calibration process assumes that the selected stylized facts should be expected in these stocks and during this time period, but stylized facts are not without exceptions and market dynamics may have qualitatively changed since the early 2000's. Table 4.5.1 summarizes the stylize fact calibration, with the main result being that facts #3 and #10 were difficult to reliably detect. Due to this lack of consistency, we rely on the remaining six stylized facts (#2 and #5 through #9) when validating ABMMS.

To provide the appropriate context for interpreting the impact of a learning agent, we select three control configurations. In `zip_simple`, we use a single exchange, a single SIP, and 30 ZIP traders, all of which are located at the Carteret node of the

Table 4.5.1: Summarized results from the calibration of stylized fact tests. Stylized facts that were not confirmed in more than half of the stocks after calibration were not considered for evaluating the ABFM.

Stylized Fact	Free Parameters	Best Parameter Values	Pass Rate
#2: Heavy Tailed Returns	Window Size	max(len(returns) // 1000, 30)	29 / 30
#3: Asymmetry of Returns	Window Size	max(len(returns) // 1000, 390)	10 / 30
#5: Intermittency of Returns	Window Size	max(len(returns) // 60, 100)	29 / 30
#6: Volatility Clustering	Lag Count	5000	30 / 30
#7: Heavy Tailed Conditional Returns	Window Size	max(len(returns) // 1000, 30)	29 / 30
#8: Slow Decay of Return Autocorrelation	Lag Count	100 or 10000	21 / 30
#9: Leverage Effect	R Value Threshold, P Value Threshold	None, None	25 / 30
#10: Volume / Volatility Correlation	R Value Threshold, P Value Threshold	None, None	12 / 30

ECN. The `zip_nms` configuration features a relatively complete representation of the NMS, with 16 exchanges distributed across the four nodes of the ECN, a SIP located in Mahwah, and a SIP located in Carteret. This condition is populated with 29 ZIP traders that are randomly distributed and one Arbitrage trader located at Secaucus. The final condition, `zip_no_arb_nms`, is identical to `zip_nms` except that it replaces the Arbitrage trader with a ZIP trader. Between these three configurations we can isolate the impacts of market infrastructure differences and understand some of the effects of market fragmentation. The experimental condition is identical to `zip_nms`, but replaces the Arbitrage trader with a Reinforcement Learning trader.

We collect data from 30 independent trials for each condition, where a single trial covers five trading days. Figure 4.5.1 shows the ability of data generated by each condition to display the six stylized facts that were selected based on the calibration discussed above. Each of the selected configurations display roughly four of the

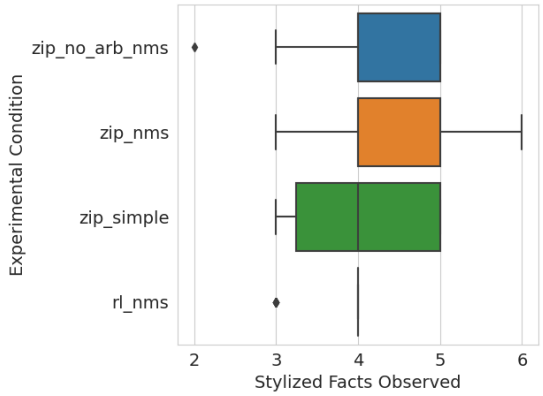

Figure 4.5.1: Box and whisker plots summarizing the number of stylized facts detected for each experimental condition. The four experimental conditions display similar capabilities for reproducing stylized facts, with an average (standard deviation) of 4.26 (0.8), 4.1 (0.78), 4.03 (0.76), and 3.75 (0.43) for the `zip_no_arb_nms`*,* `zip_nms`*,* `zip_simple`*, and* `rl_nms` *conditions respectively. The only significant difference, determined via two sided t-tests, was the lower mean for the* `rl_nms` *relative to the other conditions.* `zip_nms` *was the only condition able to display all six stylized facts simultaneously.*

six stylized facts. Two of the conditions with NMS-inspired market infrastructure, `zip_nms` and `zip_no_arb_nms`, had a slight advantage over `zip_simple`, but that difference was not statistically significant.

Figure 4.5.2 displays the detection rate of each stylized fact, across all trials and by experimental condition. Facts #5 and #6 had a perfect detection rate, facts #2, #7, and #9 were detected in more than 50% of trials, and fact #8 was detected in less than 10% of trials. The two conditions with NMS-inspired infrastructure were more likely to display fact #2 and less likely to display fact #9 than the condition with simple infrastructure. Additionally, the condition with simple infrastructure was unable to produce a single trial that displayed fact #8, whereas the NMS-inspired conditions both produced a single trial that did.

The stylized facts developed by Cont [44] are exclusively concerned with proper-

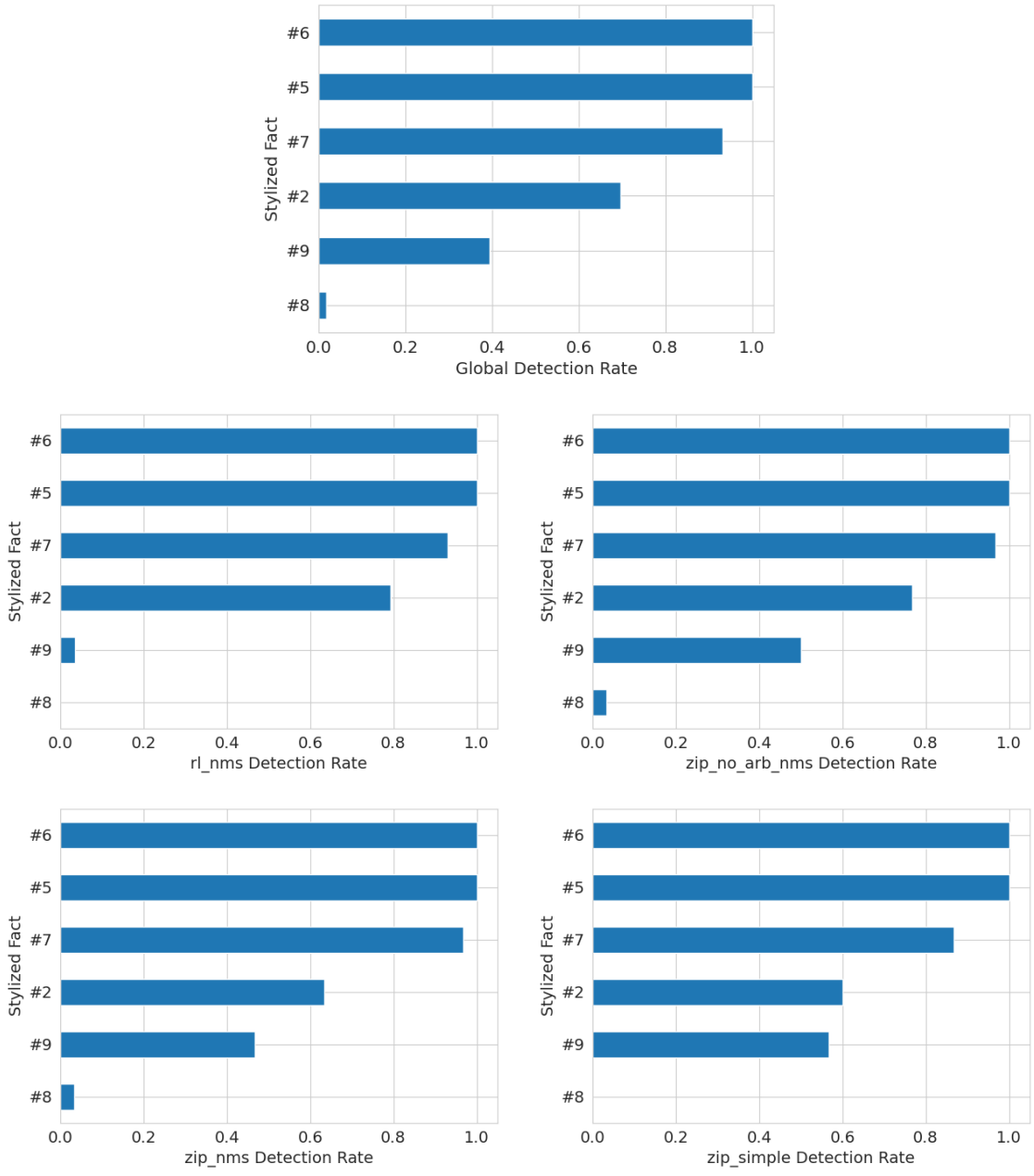

Figure 4.5.2: The detection rate for each stylized fact across all trials (top-center), `rl_nms` *trials (center-left),* `zip_no_arb_nms` *trials (center-right),* `zip_nms` *trials (bottom-left), and* `zip_simple` *trials (bottom-right).* `rl_nms` *and* `zip_simple` *conditions were unable to display fact #8, while* `zip_nms` *and* `zip_no_arb_nms` *were able to display fact #8 exactly once.* `rl_nms` *displayed fact #9 less than the other conditions, but displayed fact #2 more frequently.*

ties of asset price time series. However, there is a wealth of additional information that is produced by real markets, and by ABMMS. Figure 4.5.3 investigates differences between the experimental conditions using daily occurrences of trades, quotes, and NBBOs. Market fragmentation and the arbitrage trader both have non-trivial impacts on all of these statistics, but market fragmentation has a much larger effect. Figure 4.5.4 summarizes the occurrence of dislocations, as discussed in Tivnan et al. [234], in ABMMS. The `zip_simple` condition generates roughly an order of magnitude less dislocations than the conditions with NMS-inspired infrastructure, but those dislocations tend to be longer. The arbitrage trader appears to cause an increase in the mean dislocation magnitude, but also a large decrease in dislocation duration.

4.6 Discussion and Conclusion

There are three major directions that this work could be extended. First, we investigated the impacts of a single learning agent to avoid development difficulties that can be encountered when multiple learning agents interact, however, future work should tackle these issues and develop populations of heterogeneous learning agents. Second, we captured many important mechanisms in our implementation of ABMMS, but the NMS is an extremely complicated system and there are bound to be details that we have abstracted away. Enumerating and implementing these additional mechanisms will improve the accuracy of future models, and open additional strategies for learning agents to explore. Third, we chose to exclusively implement an equities market. However, real equity markets are linked with several financial systems, including lending systems and options markets. Extending ABMMS to account for any of these

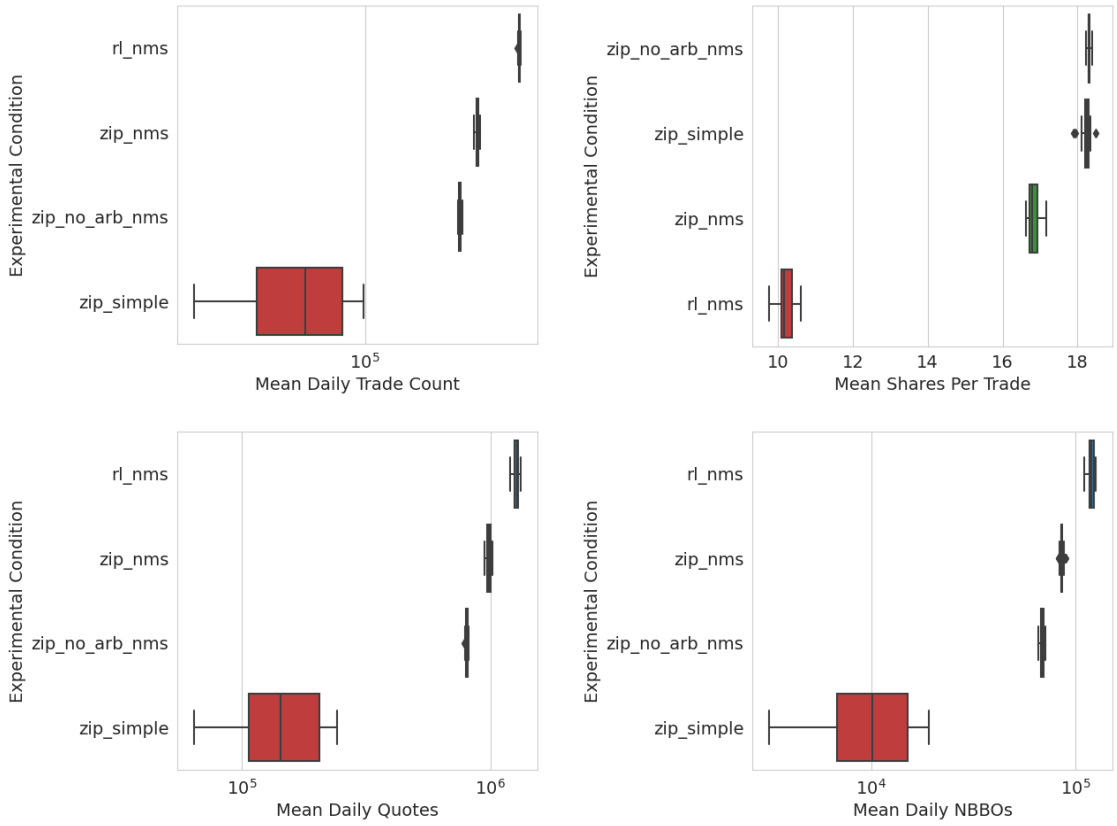

Figure 4.5.3: Basic trading day statistics for each experimental condition. The arbitrage trader causes a noticeable drop in the mean number of shares per trade (top-left). Market fragmentation leads to an order of magnitude increase in trades (top-right), quotes (bottom-left), and NBBOs (bottom-right). The arbitrage trader leads to a sizeable increase in trades, quotes, and NBBOs, but has a smaller impact than market fragmentation. The `rl_nms` *had a higher level of activity than the other conditions, but featured smaller trades on average.*

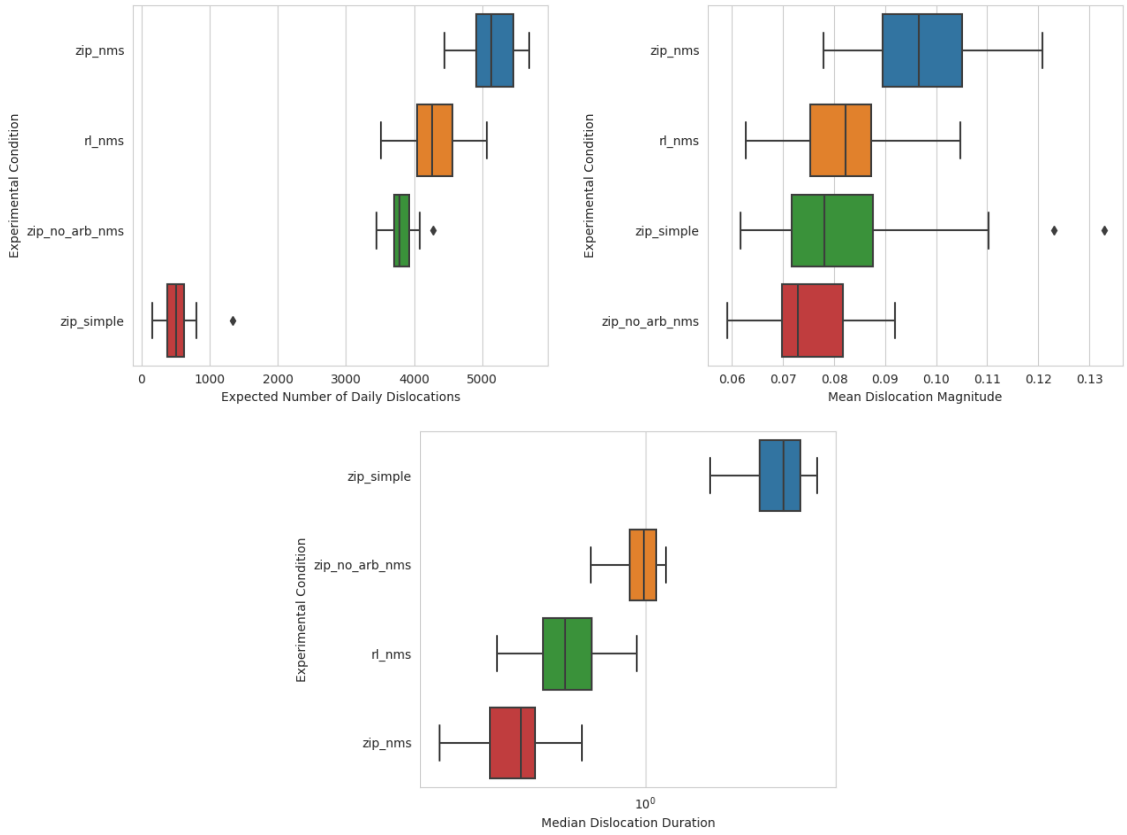

Figure 4.5.4: Summary statistics for dislocations by experimental condition. Fragmented configurations of ABMMS display roughly five times as many dislocations when compared with `zip_simple` *(top-left). The arbitrage trader leads to an increase in the number of dislocations (top-left) and an increase in the mean dislocation magnitude (top-right), but a decrease in the dislocation duration (bottom). RL trader lead to less dislocations, smaller dislocations, and longer dislocations than the arbitrage trader.*

additional financial systems could enrich the produced results. Beyond these direct extensions, our implementation and calibration of tests for stylized facts indicates a need to revisit some common stylized facts, which may be more difficult to identify, or may not be displayed in the same ways as previously observed.

Financial market policy has been shaped largely by public comments [214], recent events [213], and live pilots [215]. However, each of these influences can be problematic. Public comments can be subjective or self-serving, recent events only help retrospectively, and live pilots impose implementation costs on exchanges [166]. There have been a few successful applications of ABFMs to policy evaluation [224, 49, 98, 131, 19, 42, 20], but additional efforts could increase the amount of policy informed by ABFMs and avoid the noted issues associated with other policy influencing mechanisms.

We provide the full source code for our agents, models, and analysis (pending acceptance) [241].

Acknowledgements

We thank Thayer Alshaabi, Matthew Koehler, John Ring for their insightful discussion and suggestions. Computations were performed on the Vermont Advanced Computing Core, supported in part by NSF award No. OAC-1827314.

Appendix

4.A ODD Protocol for ABMMS

Below we describe ABMMS following the Overview, Design concepts, Details (ODD) protocol [93, 94, 95].

4.A.1 Purpose

The purpose of ABMMS is to evaluate the impact of market mechanism implementation details in combination with adaptive agents on the quality of data generated by an agent-based financial market (ABFM). Phrased more explicitly, "Does the combination of detailed market mechanisms and adaptive learning agents create synergistic effects that improve the level of realism of data generated by an ABFM?" A higher-level goal of ABMMS is to develop an ABFM that can better evaluate the impacts of policy and design decisions in the US National Market System for equities (NMS). ABMMS primarily targets the NMS, but is designed to simulate arbitrary market configurations, allowing for the investigation of design and policy perturbations.

4.A.2 Patterns

We evaluate ABMMS by its ability to reproduce the following patterns.

Stylized Facts of Asset Prices

ABMMS should produce asset price time series that satisfy stylized facts proposed by Cont and others. Replicating all of the 11 stylized facts proposed by Cont is extremely difficult, especially considering the volume of data that is required to evaluate facts #1, #4, and #11, so we aim to replicate at least 4. Additionally, we found that facts #3 and #10 were difficult to detect when calibrating our stylized fact tests on real data. Therefore, replicating four stylized facts should be considered acceptable, and six desirable.

Profits of Learning Agents

Simple agents may have positive or negative profits depending on the market conditions that they experience and random chance. However, learning agents should have positive expected profits, otherwise, economic rationality would demand that they cease participation. An agent need not have positive profit in any particular period, or even over the entirety of a simulation run, only in the long run and on average.

Daily Trading Activity

Financial markets tend to feature a smile-shaped activity density curve at the trading day time scale. The start of the trading day features a burst of trading activity, which decays as the day goes on, as well as a ramp-up of activity as the day reaches its close.

There is no mechanism to generate such an activity curve when an ABFM is populated entirely with ZI or ZIP agents, beyond engineering such activity patterns into their trading behavior. However, when learning agents are introduced, one possible mechanism for generating this activity distribution comes with them, and that is the opportunity cost associated with the market closure between trading days. To test for this pattern we can construct activity histograms for each trading day, with bins covering 10-second intervals, then test if the bins in the first and last 5 minutes of the trading day feature significantly more activity than other bins.

Dislocations

Tivnan et al. [234] describe the occurrence of quote dislocations in the NMS. Since ABMMS is intended to model the NMS, we expect to observe similar quote dislocations in the data generated by it. The quote dislocations observed in ABMMS should have similar distributions of attributes to what was observed in the NMS. On average, stocks in the NMS can exhibit daily dislocation counts that fall anywhere between roughly 3000 and 16000, for thinly traded members of the Russell 3000 and members of the Dow 30 respectively [56]. When accounting for time of day, the occurrence distribution should have a smile-like shape, where more dislocations occur near the open and close of a trading day. Additionally, the duration distribution should be heavy-tailed, with a tail reaching towards a longer duration, and a mean between 10^{-4} and 10^{-2} μs. The distribution of dislocation magnitudes should be heavy-tailed, possibly a power law, with a greater frequency of small magnitude dislocations and an exceptionally long tail [239].

4.A.3 Entities

ABMMS features the following:

- Simulation Driver: Orchestrates the execution of the simulation and manages global variables.
- Electronic Communication Network (ECN): Mediates interactions between agents.
- Agent: An actor in the simulation. Agent classes can have heterogeneous roles and incentives. All agents share a set of common state variables that cover general information.
  - Exchange: Manage auctions that facilitate stock trading.
  - Securities Information Processor (SIP): Provides a signal to synchronize prices across a fragmented marketplace.
    - Limit Up-Limit Down Queue: Tracks historical trades in a time window to aid in calculating LULD bands.
  - Trader: Buys and sells financial instruments.
    - Zero Intelligence (ZI): Based on the agents developed by Gode and Sunder, all trading decisions are selected randomly. One deviation is that we do not implement separate buyer and seller agents. Instead, each time a ZI agent is able to trade it randomly selects whether to act as a buyer or seller.
    - Minimum Intelligence (MI): Similar to ZI agents, but the width of the random price distribution is based on the spread of the NBBO, and

orders are always sent to an exchange that holds one or both sides of the NBBO.

    * ZI Plus (ZIP): Based on the agents developed by Cliff and Bruten. Takes trading actions that are nearly as random as ZI agents, except that prices are determined by a belief function that is updated based on orders that result in trades.
    * Arbitrage: Attempts to profit by uncrossing distributed markets that are crossed.
    * Reinforcement Learning (RL): Learns a trading strategy via meta-reinforcement learning, leading to a more adaptive and free form strategy.
  - Observer: An aggregator that constructs consolidated data products.

- Message: Information sent from one agent to another. All messages feature the same header information, while the body content varies based on the message type.
  - Add: A bid (buy interest) or offer (sell interest) has been added to an order book.
  - Modify (Mod): Shares have been removed from an order book without execution.
  - Trade: Shares have been removed from an order book due to execution.
  - Quote: The best bid or best offer at an exchange have updated.
  - National Best Bid and Offer (NBBO): The best bid or best offer across all exchanges in a market system have updated.

- Limit Up-Limit Down (LULD) Bands: Range of valid trading prices for an asset.
- Request: Traders may submit an add or mod request to an exchange. Requests may be rejected if they are malformed.
- Receipt: Exchanges indicate the status of a request via a receipt that is sent exclusively to the sender of the request.
- Trigger: Schedules the occurrence of a discrete event, such as a trade or an auction. Usually sent from an agent to itself, though this is not explicitly enforced.
- SIP Message: Trade and Quote messages that pass through a SIP.

An ECN represents the communication infrastructure through which all other agents interact. The core of an ECN is the topology of the communication infrastructure, which is represented as an undirected graph. The nodes of the topology represent physical locations, and edges represent communication pathways between locations. Edges are weighted to represent a deterministic propagation delay associated with sending a message across that edge. An exponential random variable is added to the deterministic propagation delay, simulating other aspects of electronic communication systems, such as queuing delays or packet loss. All messages sent via the ECN are subjected to a minimum delay, which primarily impacts messages sent between agents located at the same node. The state variables for ECNs are summarized in Table 4.A.2.

Exchanges facilitate the trade of assets by matching buyers and sellers via an auction mechanism. The auction mechanism is implemented by the combination of an

order book, which accumulates market state, and a matching engine, which matches incoming orders against those resting in the order book. Trading in multiple assets can be supported through the use of multiple independent order books. Exchanges may use transaction fees, also called market access fees or maker-taker fees, to monetize their activity. The state variables for exchanges are summarized in Table 4.A.4.

SIPs act as a synchronization mechanism by aggregating information across a fragmented market system and disseminating indicators. A SIP constructs several signals, including the national best bid and offer (NBBO), limit up-limit down (LULD) band indicators, as well as a trade and quote (TAQ) for each asset it is responsible for. The state variables for SIPs are summarized in Table 4.A.6. Each SIP tracks historical trades over a small time window to implement the Limit Up-Limit Down (LULD) mechanism. These trades are stored in a LULD Queue, which aids in the calculation of LULD bands. The state variables for LULD Queues are summarized in Table 4.A.7.

Traders buy, sell, and hold financial instruments by interacting with other traders via an exchange. Each trader tracks the state of its holdings, the amount of each traded asset, plus cash, that it possesses. Additionally, each trader implements a strategy for placing bids and offers. The state variables for Traders are summarized in Tables 4.A.8–4.A.11.

ABMMS implements a variety of message types, whose relationships are summarized in Figure 4.A.1. The state variables for each message type are summarized in Tables 4.A.12–4.A.20.

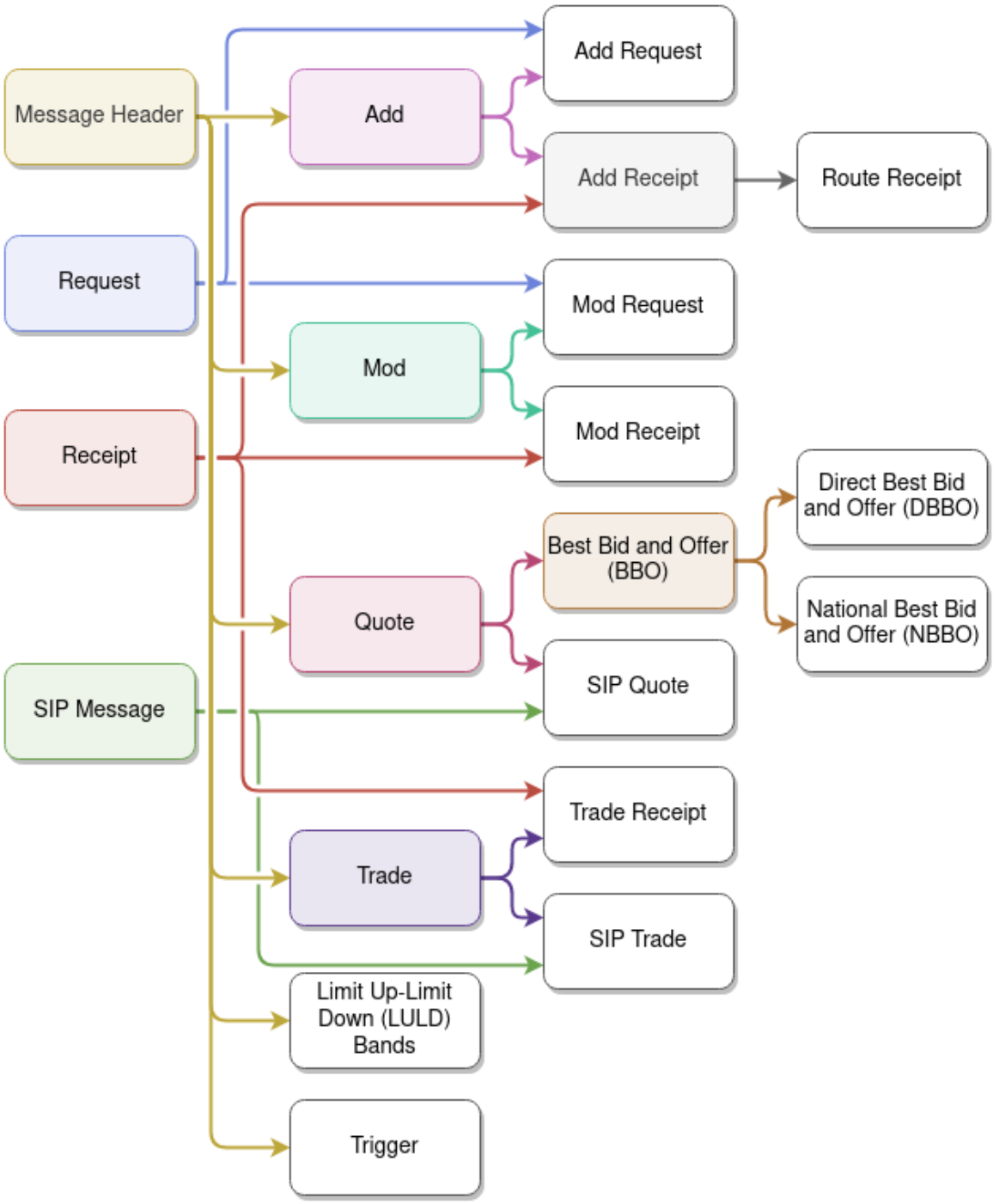

Figure 4.A.1: A graphical summary of the relationships between the message types implemented in ABMMS. Message types that are higher up in the tree share their state variables with message types that are lower in the tree, if they are connected.

4.A.4 State Variables

Table 4.A.1: State variables for the Simulation Driver entity.

Variable Name	Variable Type and Units	Meaning
Global Clock	Timestamp, dynamic; μs	Time keeper for the simulation.
Simulation Start	Timestamp, static; μs	The Global Clock is set to this at the start of the simulation.
Simulation End Time	Timestamp, static; μs	The simulation is terminated if the Global Clock reaches or passes this.
Electronic Communication Network	ECN, static	Communication infrastructure that mediates agent interactions. See Table 4.A.2 for more details.
Agents	List[Agent], static	Agents that populate this simulation. See Table 4.A.3 and related Tables for more details.
Trading Symbols	List[String], static	Identifiers for the stocks that will be traded in this simulation.

Table 4.A.2: State variables for the ECN entity.

Variable Name	Variable Type and Units	Meaning
Topology	Undirected Graph, static	Nodes represent physical locations that other agents might inhabit. Edges represent communication channels between locations. Weights on edges indicate the magnitude of deterministic propagation delays associated with communication along each edge.
Minimum Delay	Integer, static; μs	The minimum delay imposed on all communications. Primarily impacts messages passed between agents located at the same node in the ECN.
Mean Delay Noise	Float, static; μs	Scale parameter for an exponential random variable that is used to create stochastic communication delays.
Message Queue	Sorted Queue, dynamic	Contains the messages that have been sent into the ECN, but not yet arrived. Always sorted such that the first element of the queue is the next message that will arrive at its destination.

Table 4.A.3: State variables for the Agent entity.

Variable Name	Variable Type and Units	Meaning
Identifier	String, static	A unique identifier, or name, that is used to refer to this agent.
Location	Categorical, static	A node in the ECN where this agent is located.
Clock	Timestamp, dynamic; μs	A local clock. A copy of the global simulation clock by default.
Trading Symbols	List[String], static	Identifiers of stocks that this agent may interact with.
Subscribers	List[Agent], dynamic	Agents subscribed to the broadcast feed of this agent.

Table 4.A.4: State variables for the Exchange entity.

Variable Name	Variable Type and Units	Meaning
Agent State Variables	N/A	See Table 4.A.3 for more details.
Order Books	Dict[String, Order Book], static	Mapping from Trading Symbols to their associated Order books (Table 4.A.5).
Matching Engine	Matching Engine, static	Strategy for matching incoming orders with resting orders.

Table 4.A.5: State variables for the Order Book entity.

Variable Name	Variable Type and Units	Meaning
Trading Symbol	String, static	Orders are managed for this trading symbol.
Bid Priority	Ordering Function, static	Defines an ordering for the execution priority of bids.
Bids	List[Add Request], dynamic	List of Bid Requests that have been accepted, but not executed. Sorted according to Bid Priority.
Offer Priority	Ordering Function, static	Defines an ordering for the execution priority of offers.
Offers	List[Add Request], dynamic	List of Offer Requests that have been accepted, but not executed. Sorted according to Offer Priority.

Table 4.A.6: State variables for the SIP entity.

Variable Name	Variable Type and Units	Meaning
Agent State Variables	N/A	See Table 4.A.3 for more details.
LULD Queues	Dict[String, LULD Queue], static	Map from trading symbols to LULD Queues. One LULD Queue for each trading symbol that this SIP is responsible for. See Table 4.A.7 for more details.
Round Lot Size	Integer, static; shares of stock	How many shares must be associated with a quote for it to be considered a round lot, and thus eligible for inclusion in the NBBO.

Table 4.A.7: State variables for the Limit Up-Limit Down (LULD) Queue entity.

Variable Name	Variable Type and Units	Meaning
Trading Symbol	String, static	LULD bands are managed for this trading symbol.
LULD Reference Price	Integer, static	Initial reference price for the LULD bands.
LULD Window	Timedelta, static; μs	Length of the time window used to select recent trades.
LULD Percentage	Float, static; percent	Half the width of the LULD bands as a fraction of the reference price.

Table 4.A.8: State variables for the Trader entity.

Variable Name	Variable Type and Units	Meaning
Agent State Variables	N/A	See Table 4.A.3 for more details.
Holdings	Dict[String, Integer or Float], dynamic; Shares of stock or \$0.0001	Mapping from asset identifiers to possessed asset quantities.
Pending Orders	Dict[Integer, Message], dynamic	Mapping from order identifiers to orders that have been submitted to an exchange and have an unknown status.
Active Orders	Dict[Integer, Message], dynamic	Mapping from order identifiers to orders that have been placed into an order book on an exchange.
NBBOs	Dict[String, NBBO], dynamic	Mapping from trading symbols to the current NBBO for that trading symbol.

Table 4.A.9: State variables for the Zero Intelligence (ZI) Trader and Minimum Intelligence (MI) Trader entities.

Variable Name	Variable Type and Units	Meaning
Trader State Variables	N/A	See Table 4.A.8 for more details.
Maximum Limit Prices	Dict[String, Integer], dynamic; $0.01	Maximum prices for submitted limit orders, one for each traded stock.
Minimum Limit Prices	Dict[String, Integer], dynamic; $0.01	Minimum prices for submitted limit orders, one for each traded stock.

Table 4.A.10: State variables for the Zero Intelligence Plus (ZIP) Trader entity.

Variable Name	Variable Type and Units	Meaning
Trader State Variables	N/A	See Table 4.A.8 for more details.
Profit Margins	Dict[String, List[Float]], dynamic	Mapping from trading symbols to pairs of profit margins, one for bids and one for offers.
Limit Prices	Dict[String, Integer], dynamic	Mapping from trading symbols to the worst price that the agent is willing to transact at.
Target Prices	Dict[String, List[Integer]], dynamic	Mapping from trading symbols to target prices used to update the profit margins.
Momentum Values	Dict[String, Float], dynamic	Mapping from trading symbols to current momentum values.

Table 4.A.11: State variables for the Arbitrage Trader and Reinforcement Learning Trader entities.

Variable Name	Variable Type and Units	Meaning
Trader State Variables	N/A	See Table 4.A.8 for more details.
DBBOs	Dict[String, DBBO], dynamic	Mapping from Trading Symbols to their current Direct Best Bid and Offer (DBBO).

Table 4.A.12: State variables for the Message Header entity.

Variable Name	Variable Type and Units	Meaning
Message ID	Integer, static	Identifier associated with this message.
Related ID	Integer, static	Optional identifier of a related message.
Sender ID	String, static	Identifier of the agent that sent the message.
Recipient ID	String, static	Identifier of the intended recipient.
Send Time	Timestamp, static; μs	When the message was sent.
Receive Time	Timestamp, static; μs	When the message will be received.
Trading Symbol	String, static	Indicates what Trading Symbol this message is associated with.
Random	Float, static	Value drawn from a $\mathcal{U}[0, 1)$ distribution.

Table 4.A.13: State variables for the Add message entity.

Variable Name	Variable Type and Units	Meaning
Message Header	N/A	See Table 4.A.12 for details.
Sequence Number	Integer, static	Identifier applied by an exchange to indicate the processing order of requests.
Order Type	Categorical, static	Limit, market, or midpoint order.
Side	Categorical, static	Bid or offer.
Shares	Integer, static	Quantity of shares to be bought or sold.
Limit Price	Integer, static; \$0.01	Highest acceptable bid price, or lowest acceptable offer price.
All or Nothing	Boolean, static	Indicates that this order should execute in its entirety, or not at all.
Hidden	Boolean, static	Indicates that this order should not be displayed if placed in an order book.
ISO	Boolean, static	Indicates that this order is part of an inter-market sweep, and that standard execution price protections are waived.
Time in Force	Duration, static; μs	Amount of time that this order should rest in a limit order book before it is cancelled by the exchange.

Table 4.A.14: State variables for the Modify (Mod) message entity.

Variable Name	Variable Type and Units	Meaning
Message Header	N/A	See Table 4.A.12 for details.
Sequence Number	Integer, static	The Sequence Number of a resting order.
Side	Categorical, static	The Side (bid or offer) of the resting order.
Shares to Remove	Integer, static	Quantity of shares to be removed from the resting order.

Table 4.A.15: State variables for the Trade message entity.

Variable Name	Variable Type and Units	Meaning
Message Header	N/A	See Table 4.A.12 for details.
Price	Integer, static; $0.01	Execution price of the trade.
Shares	Integer, static	Quantity of shares traded.
Triggering Side	Categorical, static	Side of the order that triggered the trade.
ISO	Boolean, static	ISO status of the order that triggered the trade.

Table 4.A.16: State variables for the Quote message entity.

Variable Name	Variable Type and Units	Meaning
Message Header	N/A	See Table 4.A.12 for details.
Bid Price	Integer, static; $0.01	Highest price among bids in an order book.
Bid Shares	Integer, static	Quantity of shares associated with the highest priced bid.
Offer Price	Integer, static; $0.01	Lowest price among offers in an order book.
Offer Shares	Integer, static	Quantity of shares associated with the lowest priced offer.

Table 4.A.17: State variables for the National Best Bid and Offer (NBBO) message entity.

Variable Name	Variable Type and Units	Meaning
Quote	N/A	See Table 4.A.16 for details.
Bid Exchange	String, static	Identifier of the exchange that holds the National Best Bid.
Offer Exchange	String, static	Identifier of the exchange that holds the National Best Offer.

Table 4.A.18: State variables for the Limit Up-Limit Down (LULD) band message entity.

Variable Name	Variable Type and Units	Meaning
Message Header	N/A	See Table 4.A.12 for details.
Upper Band	Integer, static	Highest eligible trade price.
Lower Band	Integer, static	Lowest eligible trade price.

Table 4.A.19: State variables for the Receipt message entity.

Variable Name	Variable Type and Units	Meaning
Message Header	N/A	See Table 4.A.12 for details.
Success	Boolean, static	Indicates whether a request was successful or not.
Reason	String, static	Optional error message indicating why a request failed.

Table 4.A.20: State variables for the Trigger message entity.

Variable Name	Variable Type and Units	Meaning
Message Header	N/A	See Table 4.A.12 for details.
Trigger Event	Categorical, static	What event should be triggered, Auction or Trade.

4.A.5 Scales

The minimum observable time increment of ABMMS is 1 microsecond, though the "step size" is variable and based on scheduled events. The model is usually run in segments of 1 trading day (6.5 trading hours), 5 trading days (1 trading week), or 20 trading days (1 trading month). Spatial relationships are not explicitly represented, though they appear implicitly in the ECN, where propagation delays are estimated based on properties of fiber optic communication technology and geographic locations

of real world data centers [234]. The round lot size is 100 shares, and is used to filter quotes when constructing an NBBO, but odd lots are not restricted. The minimum tick size for prices of quotes is $0.01, trades can occur in $0.001 increments, and maker-taker fees are in increments of $0.0001.

All scales present in ABMMS are selected with the intent to model the NMS as closely as possible. The round lot size and minimum price increments are directly taken from regulation and documentation of NMS participants. The most subjective choice is the minimum time increment of 1 microsecond, which allows for accurate modeling of most trading processes. However, if agent response times were to be accurately modeled, a smaller minimum increment (i.e. 1 nanosecond) may be needed, since exchanges and some high frequency trading strategies may have faster response times than 1 microsecond.

4.A.6 Process overview

ABMMS is event-driven, with discrete events occurring in fine-grained, but discrete time. To initiate a simulation, a start method is called for each agent, allowing it to perform setup actions at run time and schedule initial trading actions. The results of these start methods form the seed of the event-driven simulation, where messages are processed in increasing order of time of receipt. Each agent has its own strategies for how it reacts to particular message types, but generally all state variables are updated asynchronously. The only variable that is updated synchronously is the global clock, which is shared by all agents in a simulation. The existence of a single global clock removes the possibility of clock synchronization issues, which is a prevalent and difficult problem in distributed high frequency systems. Events are

processed sequentially until there are no remaining events to process, or the simulation termination time is reached.

All agents in ABMMS respond to messages, which either convey information about state changes elsewhere in the system or that a scheduled discrete event should occur.

Agent behaviors fall into one of two classes: planned and reactive. Planned behaviors are triggered by a message sent from the agent to itself, with the intent of performing a certain action at a specific time. For example, an agent may decide that it would like to submit an order to an exchange in five minute, and schedule this planned action using a trigger message.

Reactive behaviors are triggered by messages sent to the agent from elsewhere in the system. For example, suppose an arbitrage trader receives a new quote from an exchange that indicates a crossed market state. The arbitrage trader then reacts by sending orders to the two exchanges involved in the cross, intending to be the counter party to the bid at one exchange and the offer at another.

ECN Processes

Message routing is determined based on the shortest weighted path identified by Dijkstra's algorithm. The total weight of that path is a deterministic propagation delay, which is then combined with a small exponential random variable to simulate queuing, latency jitter, and other stochastic delays. All messages are subjected to a minimum propagation delay, which mainly impacts messages passed between agents at the same node in the ECN.

Exchange Processes

Exchanges implement a trading day schedule such that trading occurs between 9am and 4pm on week days (Monday through Friday), excluding U.S. federal holidays. This is enforced by opening and closing processes, outside of which any submitted requests will be rejected.

The opening and closing procedures for exchanges in ABMMS are relatively simple compared to what is seen in the NMS. Orders are not accepted prior to market open, and continuous trading begins immediately at market open without the occurrence of an opening batch auction. Similarly, the closing process in ABMMS is simply the termination of continuous trading, unlike the NMS where it is common to execute a closing batch auction. As part of the closing process, all orders that use the default time in force value or have a time in force value less than 17.5 hours (the duration between the close and next open) are cancelled by the exchange.

During open trading hours, Exchanges will validate any incoming requests to ensure that they are well formed. Valid messages are passed to the Matching Engine, while ill formed messages result in receipts returned to their senders indicating issues.

Beyond these processes, Exchanges track the current NBBO and LULD bands for each trading symbol, and construct a Direct Best Bid and Offer (DBBO) using quotes observed directly from other exchanges.

Matching Engine Processes

Add Requests that are validated by an exchange are handed off to a Matching Engine, which determines if that order will immediately lead to trades or if it will fall to rest in a Order Book. An incoming bid immediately matches against the highest priority

offer in the Order Book if the limit price of the bid is greater than or equal to the limit price of the offer. A similar, but inverted, relationship holds for an incoming offer matching against resting bids. A single incoming request may result in multiple trades, depending on the quantity of shares desired by the incoming order and the quantity provided by resting orders. If there are not enough resting shares to complete an incoming request, then that request will fall to rest in the order book and await further counter parties.

If simulation is configured with multiple exchanges and a trade would occur at a price that is worse than what is displayed by the current NBBO, then the Matching Engine will defer execution of that order and route it to the Exchange that holds the appropriate side of the NBBO (assuming that the Matching Engine is not managed by the indicated Exchange). In addition to forwarding the remainder of the order to the NBBO holder, the routing exchange will also send a receipt to the sender of the request indicating that routing has occurred. This process is referred to as trade through protection, and the mechanism that implements trade through protection is mandatory routing.

The results of each trade are emitted in a Trade message via the feed of the Exchange that manages a Matching Engine. In addition, the pair of agents involved in a trade each receive a receipt that indicates their involvement.

Till this point we have mainly considered the case of Limit Orders, but, Exchanges in ABMMS also support Market Orders and Midpoint Orders, which function slightly differently. Limit Orders guarantee price, but not execution, using the limit price specified by the trader. On the other hand, Market Orders guarantee execution, but not price. This is implemented in ABMMS as a Limit Order whose limit price is set

to the appropriate side of the current LULD band (maximum price for Market bids, minimum price for Market offers). Midpoint Orders are Limit Orders whose limit price is set to the midpoint of the current NBBO, e.g. Best Bid Price + Best Offer Price / 2. The limit price of Midpoint Orders is updated by the Exchange each time a new NBBO is received, and this limit price adjustment does not impact other aspects of the order (i.e. submission time).

Beyond these order types, all Add Requests have a set of modifiers that can be applied to shape how they are executed. The “Time in Force” attribute of an order indicates how long it should be considered valid for once the exchange has received it. By default, orders are considered valid for a single trading day, and will be cancelled as part of the closing process. Orders with a “Time in Force” of 0 microseconds are either executed or cancelled immediately. The “All or Nothing” flag indicates that a request should be executed completely or not at all. In the current implementation, use of the "All or Nothing" flag implicitly causes the order to have a “Time in Force” of 0 microseconds. The “Intermarket Sweep Order” (ISO) flag disables trade through protections for the order, ensuring that it is not routed to another Exchange. Orders can be marked as “Hidden”, which inhibits the issuance of any Add or Quote messages based on the entry of the order.

Order Book Processes

Order Books store Add Requests that have been accepted by an Exchange, but have not yet been fulfilled. These Add Requests are stored in a pair of sorted queues, one for bids and one for offers, where the sort ordering is based on price, visibility, submission time, and a random tie-breaker. For bids, higher prices result in higher

priority, while for offers the opposite is true. Lower visibility, e.g. the use of the hidden modifier, always results in a lower priority. Older orders have priority over more recently submitted orders.

Every Add Request that falls to rest in the Order Book causes an Add Receipt to be issued to the request sender and and Add message to be issued on the feed of the controlling Exchange.

Mod Requests that have been validated by the Exchange are executed by the Order Book, resulting in a Mod Receipt response to the request sender and a Mod message issued on the feed of the controlling exchange.

Any Request that replaces or modifies the Best Bid or Best Offer results in the construction and issuance of a new Quote message. Note that hidden orders are not considered when constructing a new Quote.

Additionally, the Order Book implements the price updates for any resting Mid-point Orders on behalf of the Exchange any time the NBBO is updated.

SIP Processes

Each SIP provides a suite of services for a set of trading symbols. For each trading symbol in this set, the SIP will construct and disseminate a Trade and Quote (TAQ) feed, a National Best Bid and Offer (NBBO), and Limit Up-Limit Down (LULD) bands.

The TAQ feed is nearly a forwarding service, but the SIP appends an additional timestamp to each message indicating the time that it was received by the SIP.

A NBBO is constructed by aggregating quotes from multiple exchanges that are simultaneously trading the same symbol. The bid with the highest limit price across

Exchanges is selected for the National Best Bid, and the offer with the lowest limit price is selected for the National Best Bid. In order for a bid or offer to be considered during the construction of an NBBO it must be at least as large as a round lot, where the default round lot size is 100 shares.

LULD bands are constructed using a rolling window of trades, which covers the past 5 minutes by default. A reference price is constructed using the simple mean of the trade prices. The band values are then calculated as a percentage deviation from the reference price. We use a default value of 5% for the LULD band spread, though the real world implementation contains multiple classes of securities with varying band sizes. Additionally, the LULD band width doubles during the last 25 minutes of trading on regular trading days.

One element of the LULD system that ABMMS does not implement is trading halts. In a complete implementation, a trading halt would be triggered if trading occurred outside of the LULD bands, and did not return within 15 seconds. Full details of the LULD system can be found at https://www.luldplan.com/.

Trader Processes

Here we discuss general processes that are shared by all traders, namely a budget constraint and request tracking. Details related to trading decisions are covered in Section 4.A.8.

All traders are subject to a budget constraint that restricts their actions when they do not have enough resources. Specifically, each trader estimates a portfolio value that includes cash and shares of stock. The value for the shares of stock are estimated based on the prices displayed by the NBBO and the quantity of shares held by the

agent. If the agent owns a positive quantity of shares, then they must sell those shares to convert them into cash, thus the National Best Bid price is used. Alternatively, if the agent owns a negative quantity of shares (shorting), then they must buy shares to close out the position, thus the National Best Offer price is used. In either case, the estimated value is the number of shares multiplied by the estimated execution price. This estimate is extremely coarse and simplified, but can be computed quickly, which is necessary since traders operate at relatively fine grained time scales.

to implement the budget constraint, all traders must track their assets. The only messages that lead to a change in assets are Trade Receipts, which convert between shares and cash.

All traders also perform basic request tracking. Requests created by a trader can have two states, pending or active. Pending requests have been created by the trader and sent to an exchange, but have yet to be accepted by the exchange. Active requests have been accepted by the exchange. Mod Requests have an immediate impact if accepted, and thus can be disregarded by the trader once their impact has been noted. On the other hand, Add Requests can land in an order book, waiting there until a counter party is found. Resting Add Requests are the primary reason for Active Request tracking. Since Exchanges cancel all orders with insufficient time in force values at the end of a trading day, all traders similarly clear any tracked orders with insufficient time in force values at the end of a trading day.

4.A.7 Scheduling

Each time an agent receives a message, it may create and send any number of messages to arbitrary recipients. The receipt time of a new message is based on the time it

was sent, along with the shortest path (in terms of total propagation delays) between the location of the sender and the location of the intended recipient. Messages are processed in order of increasing receipt time.

4.A.8 Design Concepts

Basic Principles

ABMMS is designed to emulate the NMS, with the assumption that more detailed modeling of mechanics of a target system will improve the ability of an ABFM to evaluate the impacts of system perturbations. Additionally, ABMMS aims to investigate the impact of more realistic agent behavior in a similar lens. Human traders are flexible, adaptive, and heterogeneous in ways that are not captured by the relatively simple agents that commonly populate ABFMs. Both of these principles likely have non-trivial impacts on their own, but we hypothesize that their combination will create synergistic effects.

Detailed market mechanics are captured at the system level, with the ECN accounting for the details of communication and the fragmented market that is distributed across that ECN, as well as at the agent level, with the implementation of realistic exchanges, matching engines, and order books.

Simple agents, particularly ZI [83] and ZIP [38], provide a foundation upon which we can develop learning agents. We use meta-reinforcement learning [251, 62] to incentivize our learning agents to develop adaptive strategies. Additionally, we hope that the learning agents will make full use of system elements that are largely unused by the simple traders, such as market orders, midpoint orders, time in force attributes,

and mod requests.

ABMMS aims to output data that is nearly identical to data products produced by participants of the NMS. In particular, the primary output of ABMMS is a comprehensive depth of book data feed, which should allow for any kind of market data analysis commonly applied to authentic data products of similar scope. The only analyses not supported by this output are those that require attribution data, which is also true of all commercial data products. The only entities in the real system that have access to data with attribution are exchanges, regulators, and entities similar to them.

Emergence

We designed ABMMS to minimize the amount of imposed or prescribed behaviors, with the intent that almost all results collected from ABMMS would be emergent. The primary output of ABMMS is a full depth-of-book data feed, which is created entirely through mediated agent interactions. Since a minimal amount agent behaviors are prescribed or imposed, this generated data feed is a completely emergent result.

Adaptation

The main adaptive behaviors displayed in ABMMS are the strategies implemented by traders.

ZI traders make trading decisions almost completely randomly. Add Requests are scheduled to avoid request submissions outside of normal trading hours, and are otherwise based on a uniform delay distribution (details in Section 4.A.11). When submitting an Add Request the ZI trader first selects a trading symbol to target,

then a side of the market (bid or offer). The limit price is randomly selected from a uniform distribution that ranges from \$99.75 to \$100.25 initially, then is updated based on the NBBO. To create some separation between the price distributions of bids and offers, the limit price distribution for bids is shifted downwards by 25% of the NBBO spread and the limit price distribution for offers is shifted upwards by the same amount. Add Request volume is determined by a log normal distribution (details in Section 4.A.11). Finally, the exchange that an Add Request routed to is selected at random. ZI traders only use limit orders, do not use any execution modifiers, and do not submit mod requests.

MI traders are identical to ZI traders, except that they send their Add Requests to the Exchange that holds the appropriate side of the NBBO (details in Section 4.A.11).

ZIP traders very similar to ZI traders, but they develop pricing beliefs based on what prices lead to trades. The limit price used for a particular order, referred to as the shout price by Cliff and Bruten, is constructed based on an internal limit price and a multiplicative profit margin then clipped to remain inside of the current LULD bands. The internal limit price is drawn from a truncated normal distribution (details in Section 4.A.11). The profit margins are randomly initialized, and evolves following the Widrow-Hoff "delta rule" as discussed in Cliff and Bruten.

We verify our ZIP trader implementation by comparison with the implementation provided by Cliff. The only notable deviation is the endogenously defined and updated limit prices.

Arbitrage traders construct a synthetic NBBO using direct feeds from each exchange, resulting in a Direct Best Bid and Offer (DBBO). Each time the Arbitrage trader observes a crossed DBBO, i.e. where the best bid price is higher than the best

offer price, it emits a pair Add Requests to arbitrage away the cross. An offer is sent to the Exchange that holds the best bid, and a bid is sent to the Exchange that holds the best offer. The limit price of the emitted Add Requests is set using the limit price of the order that it is targeting. Both Add Requests are flagged as ISOs and use a time in force attribute of 0 microseconds (i.e. immediate or cancel).

The Reinforcement Learning Trader is trained using the IMPortance weighted Actor Learner Architecture (IMPALA) algorithm [68], as implemented by RLLib [146]. The IMPALA configuration is summarized in Table 4.A.21. The Reinforcement Learning Trader is configured following the Meta-Reinforcement Learning paradigm [62, 251], where the policy is memory augmented (using a LSTM [105]), the inputs are augmented with the previous action and reward, and the agent is presented with a distribution of tasks during training (many independent instances of ABMMS with different initializations).

During training the agent learns how to adapt to different market conditions by accounting for the temporal relationships between observations, actions, and rewards. This adaptation mechanism can be thought of as an inner reinforcement learning algorithm that is implemented by the LSTM. During evaluation, the agent is no longer updated following the IMPALA training algorithm, but the adaptation strategy learned by the LSTM remains active.

Objectives

The Reinforcement Learning Trader optimizes the log returns of its estimated total portfolio value over episodes consisting of 2000 interactions.

ZIP traders minimize the distance between their shout prices and target prices

Table 4.A.21: Training configuration for the Reinforcement Learning Trader under the IMPALA algorithm.

Label	Value
env	ABMMS
num_gpus	1
rollout_fragment_length	1000
train_batch_size	5000
horizon	2000
soft_horizon	True
lr	1e-6
num_workers	7
framework	tfe
replay_proportion	0.5
replay_buffer_num_slots	200
entropy_coeff_schedule	[[0, 1e-10], [12000000, 0.0]]
model: use_lstm	True
model: max_seq_len	64
model: lstm_use_prev_reward	True
model: lstm_use_prev_action	True
model: lstm_cell_size	512
model: fcnet_hiddens	[256, 256]
model: fcnet_activation	swish
model: vf_share_layers	False
tune.run: run_or_experiment	IMPALA
tune.run: time_budget_s	24 hours
tune.run: checkpoint_freq	10
tune.run: checkpoint_at_end	True
tune.run: keep_checkpoints_num	10
tune.run: checkpoint_score_attr	episode_reward_mean
tune.run: reuse_actors	True

using a Widrow-Hoff "delta rule" update, but do not directly optimize for profits.

Learning

As mentioned in Sections 4.A.8 and 4.A.8, the Reinforcement Learning Trader uses meta-reinforcement learning, implemented via the IMPALA algorithm, to optimize its portfolio log returns. This reinforcement learning policy reacts to incoming messages, and thus learns a direct behavior mapping from observations to actions. This policy is represented by an artificial neural network constructed with dense layers and an LSTM layer.

The primary goal for the use of learning in the Reinforcement Learning Trader is to develop a free form strategy that is able to adapt to changes in market conditions on the fly, even if those conditions were not observed during training. This kind of generalization is difficult to achieve with traditional reinforcement learning techniques, which is why we opted to use meta-reinforcement learning.

Prediction

The IMPALA algorithm used to train our RL Trader features implicit prediction through the use of the policy gradient, and explicit prediction through the use of a policy critic that predicts the rewards associated with a sequence of actions.

No other elements of ABMMS explicitly perform prediction as a part of their function.

Sensing

Agents in ABMMS only have direct access to, and full knowledge of, their own state variables. Messages sent via the ECN can communicate the value of internal state variables to other agents, however, due to propagation delays the actual value of a state variable may have already changed by the time other agents receive such a message.

Each time a ZI trader is activated to trade, the following information is available:

- Current values for the traders limit price distribution (based of the current NBBO).
- Current holdings (used for budget constraint only)

MI traders use the same information as ZI traders, but additionally use the current NBBO to determine where to route their Add Request.

Each time a ZIP trader is activated to trade, the following information is available:

- Current limit prices
- Current profit margins
- Current LULD bands (used to clip shout prices)
- Current holdings (used for budget constraint only)

Each time the RL trader receives a message it produces a response based on the following:

- Incoming message

- Current holdings (may influence strategy, used for budget constraint)
- NBBO for the trading symbol associated with the incoming message
- The last response emitted
- The last reward received
- A state vector that can contain information from any of the above elements at previous time steps
- The set of active Add Requests issued by the RL trader

Interaction

All interactions in ABMMS are mediated by the ECN, which controls the propagation delay for messages sent by any agent.

Traders send messages to, and receive messages from, Exchanges. SIPs receive messages from Exchanges, then send messages to Exchanges and Traders. Exchanges may send messages to other exchanges. Traders do not send messages to other traders, though this is a convention, and not explicitly enforced.

Stochasticity

Stochasticity is used heavily in the initialization of ABMMS, with many of the details discussed below in Section 4.A.9. The primary goal of stochastic initialization in ABMMS is to aid in exploring a neighborhood of similar markets, since any particular initialization is unlikely to match a historical state of the NMS. In the context of evaluating the impacts of policy and design perturbations, it is important to know if the impacts of a particular policy are sensitive to initial conditions.

In the ECN, exponentially distributed noise is added to deterministic propagation delays, simulating stochastic elements of electronic communication technology, like queuing delays and packet loss.

Trading decisions made by ZI, MI, and ZIP agents are largely stochastic. This is important to provide a spark that starts trading in ABMMS. Many classes of hand coded trading strategies are purely reactive, the see a particular market state occur and emit orders to take advantage of that market state. For example, the Arbitrage Trader waits for a crossed market state and then attempts to profit from uncrossing it. However, if all traders were purely reactive, then a deadlock would occur at the start of the simulation, with each agent waiting for something to happen, and no trading would occur. The stochastic actions of ZI, MI, and ZIP agents ensure that this deadlock does not occur, and allows reactive strategies to function in ABMMS.

Collectives

ABMMS does not explicitly represent any collectives, though herding behavior may emerge.

Observation

The data collected from ABMMS is designed to mimic the features that might be found in a consolidated data product. That means comprehensive coverage of data feeds from all exchanges and SIPs, full depth of book information, and no attribution data. A single observer, located at the Carteret node of the ECN, is used consistently across all simulation runs to promote comparison. Selection of the Carteret node is arbitrary, and any of the other ECN nodes would serve equally well as the location

for the observer, as long as the same location was used across all simulations. An argument could be made for Secaucus as the location for the observer, since it has the shortest average propagation delay. However, we chose Carteret to create a more direct comparison with the dislocations observed in Tivnan et al.

4.A.9 Initialization

The first step of initialization creates the ECN that will mediate all agent communication. An ECN is composed of a network topology, a minimum propagation delay, and a delay jitter distribution. The network topology is a weighted and undirected graph that defines the places where agents can be located as well as deterministic propagation delays between those locations. The minimum propagation delay ensures that all inter-agent communication experiences some amount of delay, avoiding unrealistic scenarios that could occur if some kinds of communication experienced no delays. The delay jitter distribution adds a stochastic element to the otherwise deterministic communication delays, and is intended to model queuing delays and other stochastic delays present in electronic communication systems.

The default topology closely mimics our understanding of the current communication infrastructure in the NMS, featuring four locations, with Mahwah, Carteret, and Secaucus connected in a triangle, and Weehawken connected to Secaucus. The default minimum propagation delay is 5 microseconds, and is based on our understanding of intra-data center communication latency. The default delay jitter distribution is an exponential distribution with a mean of 5 microseconds, and was chosen arbitrarily.

Next, the active trading symbols for the simulation must be defined. This is deterministic by default, and the specific names used are arbitrary since there are no

fundamental value signal or other aspects of real world companies associated with the trading symbols in ABMMS. We use two trading symbols in the default configuration of ABMMS, so that each of the two SIPs present in the default agent configuration manage a single trading symbol.

After the trading symbols are determined, then the various agent groups, Exchanges, SIPs, Traders, and observers, are initialized. ABMMS currently implements two exchange configurations, one based on the NMS in early 2021 and a simplified system that features a single exchange. The primary configuration parameters for exchanges, beyond an identifier and location, are a maker fee and a taker fee. The maker fee and taker fee define a simplified access fee schedule, and allows for the implementation of a traditional maker-taker system as well as an inverted taker-maker system.

In the NMS exchange configuration there are two SIPs, one located in Mahwah and one located in Carteret. By convention the trading symbols are separated into three groups, Tape A, Tape B, and Tape C. Trading symbols on Tape A and B are managed by the SIP in Mahwah, while the trading symbols on Tape C are managed by the SIP in Carteret. In the simplified exchange configuration, there is a single SIP that is colocated with the single exchange and handles all of the trading symbols.

For the configuration of trading agents, the first concern is the ecology of agent types. ABMMS implements 5 trader types: Zero Intelligence, Minimum Intelligence, Zero Intelligence Plus, Arbitrage, and Reinforcement Learning. After the ecology of agents has been determined, they must be placed at a node in the ECN. ZI, MI, and ZIP traders are usually placed randomly. In the currently implemented agent configurations, either no arbitrage traders are included, or a single arbitrage trader

Table 4.A.22: Distributions used to initialize trading agent holdings. Initial holdings for each trading symbol are drawn independently from the indicated distributions. These initial holding distributions are arbitrary. Exponential distributions are used based on the understanding that wealth distributions tend to be heavy tailed. We chose not to use distributions with unbounded mean and/or variance to improve the consistency of ABMMS results.

Agent Type	Cash Initialization	Stock Initialization
RL Trader	Exponential with a mean of $100 million	0 shares
Arbitrage Trader	Exponential with a mean of $100 million	0 shares
ZIP Trader	Exponential with a mean of $100 thousand	Exponential with a mean of 10000 shares
MI Trader	Exponential with a mean of $10 thousand	Exponential with a mean of 10000 shares
ZI Trader	Exponential with a mean of $10 thousand	Exponential with a mean of 1000 shares
Exchange	$0	0 Shares
Default	Exponential with a mean of $1 thousand	Exponential with a mean of 100 shares

is placed at Secaucus. Similarly, most configurations do not feature any RL agents, but our primary experimental configuration features a single RL agent located at Secaucus. Each trader is randomly assigned an initial allocation of cash and shares of stock, see Table 4.A.22 for details.

The last agents to be initialized are the observers. The main initialization concern with observers is their number and location. In all configurations we use a single observer located at the Carteret node.

Once the ECN and agents have been initialized, they must be wired together before trading can begin. All agents subscribe to data feeds provided by agents that they plan to interact with. The default configuration is that SIPs subscribe to Exchanges,

while Exchanges, Traders, and Observers subscribe to Exchanges and SIPs. Next, all agents are registered with the ECN based on their location within the network topology.

To provide a common starting point for trader price beliefs, each SIP issues LULD bands for the trading symbols that it is responsible for. Since ABMMS does not currently implement opening auctions, the LULD bands issued at the open are based on the closing from the "previous day", which is drawn from a uniform distribution ranging from $90.00 to $110.00. Individual agents may initialize their price beliefs based on these LULD bands, or use their own initialization schemes.

ZI and MI traders do not have any parameters beyond their minimum and maximum limit prices that need to be initialized. Arbitrage traders also do not have any additional parameters that require initialization.

For the parameters of ZIP agents, we follow the same initialization process as Cliff. Specifically, the learning rate for each agent is drawn uniformly from 0.1 to 0.5, the momentum parameter is drawn uniformly from 0.0 to 0.1, and the profit margins are drawn uniformly from 0.05 to 0.35 (values are positive for offer profits and negative for bid profits). For the target price perturbation functions, absolute shifts are drawn uniformly from $0.00 to $0.05 and relative shifts are also uniformly random with a maximum change of plus or minus 5%.

4.A.10 Input Data

ABMMS does not use exogenous input data to represent time-varying processes.

4.A.11 Submodels

Trader Wait Intervals

ZI, MI, and ZIP traders schedule their trading actions based on a uniform distribution with a minimum wait time of 0.5 seconds and a maximum wait time of 1.5 seconds.

Trader Order Volume

ZI, MI, and ZIP traders determine the amount of shares associated with Add Requests based on a log normal distribution with $\mu = 2.6051702, \sigma = 1.40943376$. These parameters were selected so that the mode of the distribution was close to 100 (the round lot size), and the mean of the distribution was close to 270 (Average Order Size displayed by https://iextrading.com/stats/ on 2021/04/05).

Trader Order Routing

ZI and ZIP traders uniformly randomly select the target exchange for each request.

MI traders send their orders to the exchange that holds the appropriate side of the NBBO. If that side of the NBBO is currently undefined, then MI traders will fall back to random selection.

ZIP Trader Limit Price

Traditionally, ZIP traders develop shout prices, the price applied to Add Requests, using an exogenous limit price and a multiplicative profit margin. To avoid providing exogenous limit prices, our ZIP trader implementation instead selects random limit prices following a truncated normal distribution with parameters $\mu = \texttt{Mid}, \sigma =$

`Range`/12.0, where `Mid` is the midpoint of the current LULD band and `Range` is the difference between the upper and lower values of the current LULD band. The values drawn from this normal distribution are truncated to remain within the current LULD band. These limit prices are resampled each time a new LULD band is issued. The denominator used to construct σ is arbitrary, and is intended to keep the majority of limit prices centrally located.

Chapter 5

Conclusion

Chapters 2, 3, and 4 describe three recent machine learning applications, CASI, AMP-GAN, and ABMMS respectively. These applications serve as concrete examples of how deep learning systems can leverage domain knowledge to solve challenging problems.

In the first application, CASI, domain knowledge informed the problem formulation, data collection, data pre-processing, and model selection steps. Star formation theory [67] drove the desire to detect shells in telescope images. The massive amount of data collected by telescopes, and the lack of an automated solution, motivated a deep learning application. All the data and labels used to train CASI were generated by a simulation, which served as a mechanism to encode domain knowledge from human experts into a form that was appropriate for the deep learning model. During data pre-processing we cut the volumes of simulated data into 2D slices, increasing the number of training samples available and maintaining most of the important spatial relationships. Building on that decision we selected a 2D convolutional neural network model to explicitly account for those spatial relationships.

Result interpretation usually requires domain knowledge to provide an adequate frame of reference for understanding model performance. However, since we formalized shell detection as semantic segmentation during problem formulation, we were able to evaluate CASI using general segmentation metrics. The use of general metrics removed the need for domain-specific evaluation metrics.

In the second application, AMPGAN, domain knowledge informed all steps except model fitting. The rise of drug resistance, the slowing of development of traditional antibiotics, and rising interest in AMPs all contributed to the desire to accelerate AMP design. We used data that was collected almost exclusively through scientific experiments executed by chemists. The problem formulation required our use of peptide sequences as a feature, but domain knowledge directed the remainder of feature selection. Regarding model selection, the peptide sequences drove the selection of a 1D CNN-based model, the development/design objective led to the selection of a GAN, the desire for interactivity and human intervention influenced the decision to use a conditional model. Finally, for the result interpretation step, AMPGAN was evaluated using global peptide similarity scores, existing models, and cell membrane simulations, all of which directly encode domain knowledge.

In the third application, ABMMS, domain knowledge informed problem formulation, data collection, model selection, and result interpretation. The costs of live pilot programs and a desire to understand complex financial systems drove the need for models that can better evaluate counterfactuals. ABMMS, an ABFM, provided an environment to train meta-reinforcement learning agents, evaluate those agents, and generate simulated data from a variety of market configurations. Data pre-processing, in the form of feature selection and feature encoding for the learning agent, was pri-

marily driven by domain knowledge. We chose to develop intelligent agents with meta-reinforcement learning due to its ability to explicitly handle temporal dynamics and evolving market conditions. Economists developed stylized facts using quantitative observations of real markets, and we applied those facts to evaluate the data generated by ABMMS.

By synthesizing common patterns that appeared in multiple of the presented applications we can identify effective strategies for leveraging domain knowledge in deep learning systems. One of the most prominent patterns was learning from simulation. Simulations provide a convenient mechanism for encoding domain knowledge, and can easily support multiple learning paradigms, such as supervised or reinforcement learning (as exemplified by CASI and ABMMS respectively). Semi-supervised and unsupervised learning can make use of unlabeled data, which is much easier to collect than the labeled data required by supervised learning. Domain knowledge-driven evaluation techniques are a crucial element of many applications, including AMP-GAN and ABMMS. Post-hoc model interpretation techniques are particularly useful since they can construct absolute and relative frames of reference that allow us to understand model performance in context [173].

Though the role of domain knowledge in data preprocessing and model fitting has gradually shrunk in most deep learning applications, it remains critical for problem formulation, data collection, model selection, and result interpretation. Machine learning applications formulated without domain knowledge are likely to be solutions in search of problems. Data collected without domain knowledge are likely to miss relevant features that may be necessary to reach acceptable performance. Models selected without domain knowledge often ignore known relationships in the data,

leading to lower performance. Result interpretation without domain knowledge is nearly impossible for many applications, especially if the problem formalism for that application diverges from the common archetypes explored by the machine learning community.

In this dissertation, I presented three deep learning applications that target challenging real-world problems. These applications were successful due, in part, to appropriate leverage of domain knowledge at key steps of development. By combining the strengths of domain knowledge and deep learning, the machine learning community is now able to tackle a broader range of applications than ever before. Based on the success of the presented applications, future work should continue to investigate simulation-driven learning, supervised learning alternatives, and post-hoc evaluation methods.

Bibliography

[1] Frédéric Abergel, Marouane Anane, Anirban Chakraborti, Aymen Jedidi, and Ioane Muni Toke. *Limit order books*. Cambridge University Press, 2016.

[2] JM Ageitos, A Sánchez-Pérez, P Calo-Mata, and TG Villa. "Antimicrobial peptides (AMPs): Ancient compounds that represent novel weapons in the fight against bacteria". In: *Biochemical Pharmacology* 133 (2017), pp. 117–138.

[3] Piyush Agrawal and Gajendra PS Raghava. "Prediction of antimicrobial potential of a chemically modified peptide from its tertiary structure". In: *Frontiers in Microbiology* 9 (2018), p. 2551.

[4] J. Alves, M. Lombardi, and C. J. Lada. "The mass function of dense molecular cores and the origin of the IMF". In: *Astronomy and Astrophysics* 462 (Jan. 2007), pp. L17–L21. DOI: 10.1051/0004-6361:20066389. eprint: arXiv:astro-ph/0612126.

[5] Sarabjot S Anand, David A Bell, and John G Hughes. "The role of domain knowledge in data mining". In: *Proceedings of the fourth international conference on Information and knowledge management*. 1995, pp. 37–43.

[6] James Zou Anvita Gupta. "Feedback GAN for DNA optimizes protein functions". In: *Nature Machine Intelligence* 1.2 (2019), pp. 105–111.

[7] H. G. Arce, M. A. Borkin, A. A. Goodman, J. E. Pineda, and C. N. Beaumont. "A Bubbling Nearby Molecular Cloud: COMPLETE Shells in Perseus". In: *The Astrophysical Journal* 742, 105 (Dec. 2011), p. 105. DOI: 10.1088/0004-637X/742/2/105. eprint: 1109.3368.

[8] H. G. Arce, M. A. Borkin, A. A. Goodman, J. E. Pineda, and M. W. Halle. "The COMPLETE Survey of Outflows in Perseus". In: *The Astrophysical Journal* 715 (June 2010), pp. 1170–1190. DOI: 10.1088/0004-637X/715/2/1170. eprint: 1005.1714.

[9] SM Niaz Arifin and Gregory R Madey. "Verification, Validation, and Replication Methods for Agent-Based Modeling and Simulation: Lessons Learned the Hard Way!" In: *Concepts and Methodologies for Modeling and Simulation*. Springer, 2015, pp. 217–242.

[10] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein generative adversarial networks". In: *International conference on machine learning*. 2017, pp. 214–223.

[11] Markus Baldauf and Joshua Mollner. "Trading in fragmented markets". In: *Journal of Financial and Quantitative Analysis* 56.1 (Feb. 2021), pp. 93–121.

[12] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag. "Collaborative hyperparameter tuning". In: *International conference on machine learning*. 2013, pp. 199–207.

[13] Rainier Barrett, Shaoyi Jiang, and Andrew D White. "Classifying antimicrobial and multifunctional peptides with Bayesian network models". In: *Peptide Science* 110.4 (2018), e24079.

[14] Christopher N Beaumont, Alyssa A Goodman, Sarah Kendrew, Jonathan P Williams, and Robert Simpson. "The Milky Way Project: leveraging citizen science and machine learning to detect interstellar bubbles". In: *The Astrophysical Journal Supplement Series* 214.1 (2014), p. 3.

[15] Christopher N Beaumont, Jonathan P Williams, and Alyssa A Goodman. "Classifying Structures in the Interstellar Medium with Support Vector Machines: The G16. 05-0.57 Supernova Remnant". In: *The Astrophysical Journal* 741.1 (2011), p. 14.

[16] Kristin P Bennett and Colin Campbell. "Support vector machines: hype or hallelujah?" In: *Acm Sigkdd Explorations Newsletter* 2.2 (2000), pp. 1–13.

[17] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *Journal of Machine Learning Research* 13.Feb (2012), pp. 281–305.

[18] Thomas Blaschke, Marcus Olivecrona, Ola Engkvist, Jürgen Bajorath, and Hongming Chen. "Application of generative autoencoder in de novo molecular design". In: *Molecular Informatics* 37.1-2 (2018), p. 1700123.

[19] Richard Bookstaber, Michael D Foley, and Brian F Tivnan. "Toward an understanding of market resilience: market liquidity and heterogeneity in the investor decision cycle". In: *Journal of Economic Interaction and Coordination* 11.2 (2016), pp. 205–227.

[20] Richard Bookstaber, Mark Paddrik, and Brian Tivnan. "An agent-based model for financial vulnerability". In: *Journal of Economic Interaction and Coordination* 13.2 (2018), pp. 433–466.

[21] Jean-Philippe Bouchaud, Marc Mézard, and Marc Potters. "Statistical properties of stock order books: empirical results and models". In: *Quantitative finance* 2 (2002), pp. 251–256.

[22] R. D. Boyden, S. S. R. Offner, E. W. Koch, and E. W. Rosolowsky. "Assessing the Impact of Astrochemistry on Molecular Cloud Turbulence Statistics". In: *The Astrophysical Journal* 860, 157 (June 2018), p. 157. DOI: 10.3847/1538-4357/aac76d. eprint: 1805.09775.

[23] Jeremy P Bradshaw. "Cationic antimicrobial peptides". In: *BioDrugs* 17.4 (2003), pp. 233–240.

[24] Volker Brendel and HG Busse. "Genome structure described by formal languages". In: *Nucleic Acids Research* 12.5 (1984), pp. 2561–2568.

[25] Andrew Brock, Jeff Donahue, and Karen Simonyan. "Large scale gan training for high fidelity natural image synthesis". In: *arXiv preprint arXiv:1809.11096* (2018).

[26] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. "Neural photo editing with introspective adversarial networks". In: *arXiv preprint arXiv:1609.07093* (2016).

[27] Stephen J Brown. "The efficient markets hypothesis: The demise of the demon of chance?" In: *Accounting & Finance* 51.1 (2011), pp. 79–95.

[28] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. "Multi-agent reinforcement learning: An overview". In: *Innovations in multi-agent systems and applications-1* (2010), pp. 183–221.

[29] Arthur le Calvez and Dave Cliff. "Deep learning can replicate adaptive traders in a limit-order-book financial market". In: *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2018, pp. 1876–1883.

[30] Alice Capecchi and Jean-Louis Reymond. "Peptides in chemical space". In: *Medicine in Drug Discovery* 9 (2021), p. 100081.

[31] Zhengyou Zhang Cha Zhang. *A Survey of Recent Advances in Face Detection*. Tech. rep. June 2010. URL: https://www.microsoft.com/en-us/research/publication/a-survey-of-recent-advances-in-face-detection/.

[32] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. "Mode regularized generative adversarial networks". In: *arXiv preprint arXiv:1612.02136* (2016).

[33] Jinyin Chen, Yangyang Wu, Chengyu Jia, Haibin Zheng, and Guohan Huang. "Customizable Text Generation via Conditional Text Generative Adversarial Network". In: *Neurocomputing* (2019).

[34] Shuan Chen and Hyun Uk Kim. "Designing Novel Functional Peptides by Manipulating a Temperature in the Softmax Function Coupled with Variational Autoencoder". In: IEEE. 2019, pp. 6010–6012.

[35] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. "Infogan: Interpretable representation learning by information maximizing generative adversarial nets". In: *Advances in neural information processing systems*. 2016, pp. 2172–2180.

[36] E. et al. Churchwell. "The Bubbling Galactic Disk". In: *The Astrophysical Journal* 649 (Oct. 2006), pp. 759–778. DOI: 10.1086/507015.

[37] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).

[38] Dave Cliff. "Minimal-intelligence agents for bargaining behaviors in market-based environments". In: *Hewlett-Packard Labs Technical Reports* (1997).

[39] Dave Cliff. "BSE: A Minimal Simulation of a Limit-Order-Book Stock Exchange". In: *Proceedings of the 30th European Modeling and Simulation Symposium (EMSS 2018)* (2018), pp. 194–203.

[40] Dave Cliff and Janet Bruten. "Zero is Not Enough: On The Lower Limit of Agent Intelligence For Continuous Double Auction Markets". In: *Hewlett-Packard Labs Technical Reports* (1997).

[41] Peter JA Cock, Tiago Antao, Jeffrey T Chang, Brad A Chapman, Cymon J Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, and Michiel J. L. de Hoon. "Biopython: freely available Python tools for computational molecular biology and bioinformatics". In: *Bioinformatics* 25.11 (2009), pp. 1422–1423.

[42] Charles D. Collver Jr. *An application of agent-based modeling to market structure policy: the case of the U.S. Tick Size Pilot Program and market maker profitability*. 2017. URL: https://www.sec.gov/marketstructure/research/increasing-the-mpi-combined.pdf.

[43] UniProt Consortium. "UniProt: a worldwide hub of protein knowledge". In: *Nucleic Acids Research* 47.D1 (2019), pp. D506–D515.

[44] Rama Cont. "Empirical properties of asset returns: stylized facts and statistical issues". In: *Quantitative Finance* 1.1 (2001), pp. 223–236.

[45] Rama Cont, Sasha Stoikov, and Rishi Talreja. "A stochastic model for order book dynamics". In: *Operations research* 58.3 (2010), pp. 549–563.

[46] A. J. Cunningham, A. Frank, A. C. Quillen, and E. G. Blackman. "Outflow-driven Cavities: Numerical Simulations of Intermediaries of Protostellar Turbulence". In: *The Astrophysical Journal* 653 (Dec. 2006), pp. 416–424. DOI: 10.1086/508762. eprint: astro-ph/0603014.

[47] Anik Daigle, Gilles Joncas, and Marc Parizeau. "Automatic detection of expanding HI shells in the Canadian galactic plane survey data". In: *The Astrophysical Journal* 661.1 (2007), p. 285.

[48] Anik Daigle, Gilles Joncas, Marc Parizeau, and Marc-Antoine Miville-Deschênes. "Automatic Detection of Expanding H i Shells Using Artificial Neural Networks". In: *Publications of the Astronomical Society of the Pacific* 115.808 (2003), p. 662.

[49] Vincent Darley. *A NASDAQ market simulation: insights on a major market from the science of complex adaptive systems*. Vol. 1. World Scientific, 2007.

[50] Payel Das, Tom Sercu, Kahini Wadhawan, Inkit Padhi, Sebastian Gehrmann, Flaviu Cipcigan, Vijil Chenthamarakshan, Hendrik Strobelt, Cicero dos Santos, Pin-Yu Chen, Yi Yan Yang, Jeremy Tan, James Hedrick, Jason Crain, and Aleksandra Mojsilovic. "Accelerating Antimicrobial Discovery with Controllable Deep Generative Models and Molecular Dynamics". In: *arXiv preprint arXiv:2005.11248* (2020).

[51] Payel Das, Kahini Wadhawan, Oscar Chang, Tom Sercu, Cicero Dos Santos, Matthew Riemer, Vijil Chenthamarakshan, Inkit Padhi, and Aleksandra Mojsilovic. "Pepcvae: Semi-supervised targeted design of antimicrobial peptide sequences". In: *arXiv preprint arXiv:1810.07743* (2018).

[52] Ishita Dasgupta, Jane Wang, Silvia Chiappa, Jovana Mitrovic, Pedro Ortega, David Raposo, Edward Hughes, Peter Battaglia, Matthew Botvinick, and Zeb Kurth-Nelson. "Causal reasoning from meta-reinforcement learning". In: *arXiv preprint arXiv:1901.08162* (2019).

[53] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. "Deep direct reinforcement learning for financial signal representation and trading". In: *IEEE transactions on neural networks and learning systems* 28.3 (2016), pp. 653–664.

[54] Mathilde R Desselle, Ruth Neale, Karl A Hansford, Johannes Zuegg, Alysha G Elliott, Matthew A Cooper, and Mark AT Blaskovich. *Institutional profile: community for open antimicrobial drug discovery–crowdsourcing new antibiotics and antifungals*. 2017.

[55] Sripad Krishna Devalla, Prajwal K Renukanand, Bharathwaj K Sreedhar, Giridhar Subramanian, Liang Zhang, Shamira Perera, Jean-Martial Mari, Khai Sing Chin, Tin A Tun, Nicholas G Strouthidis, et al. "DRUNET: a dilated-residual U-Net deep learning network to segment optic nerve head tissues in optical coherence tomography images". In: *Biomedical optics express* 9.7 (2018), pp. 3244–3265.

[56] David Rushing Dewhurst, Colin M Van Oort, John H Ring IV, Tyler J Gray, Christopher M Danforth, and Brian F Tivnan. "Scaling of inefficiencies in the US equity markets: Evidence from three market indices and more than 2900 securities". In: *arXiv preprint arXiv:1902.04691* (2019).

[57] Rick Di Mascio, Anton Lines, and Narayan Y Naik. "Alpha decay". In: *SFS Finance Cavalcade* (2016).

[58] JD Diaz, Kenji Bekki, Duncan A Forbes, Warrick J Couch, Michael J Drinkwater, and Simon Deeley. "Classifying the formation processes of S0 galaxies using Convolutional Neural Networks". In: *Monthly Notices of the Royal Astronomical Society* (2019).

[59] Sander Dieleman, Kyle W Willett, and Joni Dambre. "Rotation-invariant convolutional neural networks for galaxy morphology prediction". In: *Monthly notices of the royal astronomical society* 450.2 (2015), pp. 1441–1459.

[60] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. "Adversarial feature learning". In: *arXiv preprint arXiv:1605.09782* (2016).

[61] Guozhu Dong and Huan Liu. *Feature engineering for machine learning and data analytics*. CRC Press, 2018.

[62] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. "Rl ^{2}: Fast reinforcement learning via slow reinforcement learning". In: *arXiv preprint arXiv:1611.02779* (2016).

[63] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[64] Matthew Duffin and John Cartlidge. "Agent-based model exploration of latency arbitrage in fragmented financial markets". In: *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2018, pp. 2312–2320.

[65] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. "Adversarially learned inference". In: *arXiv preprint arXiv:1606.00704* (2016).

[66] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *ArXiv e-prints* (Mar. 2016). eprint: 1603.07285.

[67] Soňa Ehlerová and Jan Palouš. “Triggered star formation in expanding shells”. In: *Monthly Notices of the Royal Astronomical Society* 330.4 (2002), pp. 1022–1026.

[68] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. “Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures”. In: *International Conference on Machine Learning*. ICML, 2018, pp. 1407–1416.

[69] J Doyne Farmer, Paolo Patelli, and Ilija I Zovko. “The predictive power of zero intelligence in financial markets”. In: *Proceedings of the National Academy of Sciences* 102.6 (2005), pp. 2254–2259.

[70] C. Federrath. “Inefficient star formation through turbulence, magnetic fields and feedback”. In: *Monthly Notices of the Royal Astronomical Society* 450 (July 2015), pp. 4035–4042. DOI: 10.1093/mnras/stv941. eprint: 1504.03690.

[71] William Fedus, Ian Goodfellow, and Andrew M Dai. “MaskGAN: better text generation via filling in the_”. In: *arXiv preprint arXiv:1801.07736* (2018).

[72] Jonathon B Ferrell, Jacob M Remington, Colin M Van Oort, Mona Sharafi, Reem Aboushousha, Yvonne Janssen-Heininger, Severin T Schneebeli, Matthew J Wargo, Safwan Wshah, and Jianing Li. “A Generative Approach toward Precision Antimicrobial Peptide Design”. In: *BioRxiv* (2020).

[73] Christopher D Fjell, Robert EW Hancock, and Artem Cherkasov. “AMPer: a database and an automated discovery tool for antimicrobial peptides”. In: *Bioinformatics* 23.9 (2007), pp. 1148–1155.

[74] M. A. Frerking, W. D. Langer, and R. W. Wilson. “The relationship between carbon monoxide abundance and visual extinction in interstellar clouds”. In: *The Astrophysical Journal* 262 (Nov. 1982), pp. 590–605. DOI: 10.1086/160451.

[75] Daniel Friedman. “The double auction market institution: A survey”. In: *The Double Auction Market Institutions, Theories, and Evidence*. Routledge, 2018, pp. 3–26.

[76] Kunihiko Fukushima and Sei Miyake. “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition”. In: *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[77] Ryan J Gallagher, Morgan R Frank, Lewis Mitchell, Aaron J Schwartz, Andrew J Reagan, Christopher M Danforth, and Peter Sheridan Dodds. “Generalized word shift graphs: a method for visualizing and explaining pairwise comparisons between texts”. In: *EPJ Data Science* 10.1 (2021), p. 4.

[78] Francois Ghoulmie, Rama Cont, and Jean-Pierre Nadal. “Heterogeneity and feedback in an agent-based market model”. In: *Journal of Physics: condensed matter* 17.14 (2005), S1259.

[79] Mario Gimona. “Protein linguistics—a grammar for modular protein assembly?” In: *Nature Reviews Molecular Cell Biology* 7.1 (2006), pp. 68–73.

[80] Sambit K Giri, Garrelt Mellema, Keri L Dixon, and Ilian T Iliev. “Bubble size statistics during reionization from 21-cm tomography”. In: *Monthly Notices of the Royal Astronomical Society* 473.3 (2017), pp. 2949–2964.

[81] Steven Gjerstad and John Dickhaut. “Price formation in double auctions”. In: *Games and economic behavior* 22.1 (1998), pp. 1–29.

[82] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[83] Dhananjay K Gode and Shyam Sunder. “Allocative efficiency of markets with zero-intelligence traders: Market as a partial substitute for individual rationality”. In: *Journal of political economy* 101.1 (1993), pp. 119–137.

[84] Giorgi Gogoladze, Maia Grigolava, Boris Vishnepolsky, Mindia Chubinidze, Patrice Duroux, Marie-Paule Lefranc, and Malak Pirtskhalava. “DBAASP: database of antimicrobial activity and structure of peptides”. In: *FEMS microbiology letters* 357.1 (2014), pp. 63–68.

[85] Gabriel Goh. “Why Momentum Really Works”. In: *Distill* (2017). http://distill.pub/2017/momentum. DOI: 10.23915/distill.00006.

[86] Bárbara Gomes, Marcelo T Augusto, Mário R Felício, Axel Hollmann, Octávio L Franco, Sónia Gonçalves, and Nuno C Santos. “Designing improved active peptides for therapeutic approaches against infectious diseases”. In: *Biotechnology Advances* 36.2 (2018), pp. 415–429.

[87] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. “Automatic chemical design using a data-driven continuous representation of molecules”. In: *ACS Central Science* 4.2 (2018), pp. 268–276.

[88] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. “Generative adversarial nets”. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[89] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. “Maxout networks”. In: *arXiv preprint arXiv:1302.4389* (2013).

[90] A. A. Goodman, E. W. Rosolowsky, M. A. Borkin, J. B. Foster, M. Halle, J. Kauffmann, and J. E. Pineda. "A role for self-gravity at multiple length scales in the process of star formation". In: *Nature* 457 (Jan. 2009), pp. 63–66. DOI: 10.1038/nature07609.

[91] Osamu Gotoh. "An improved algorithm for matching biological sequences". In: *Journal of molecular biology* 162.3 (1982), pp. 705–708.

[92] Martin D Gould, Mason A Porter, Stacy Williams, Mark McDonald, Daniel J Fenn, and Sam D Howison. "Limit order books". In: *Quantitative Finance* 13.11 (2013), pp. 1709–1742.

[93] Volker Grimm, Uta Berger, Finn Bastiansen, Sigrunn Eliassen, Vincent Ginot, Jarl Giske, John Goss-Custard, Tamara Grand, Simone K Heinz, Geir Huse, et al. "A standard protocol for describing individual-based and agent-based models". In: *Ecological modelling* 198.1-2 (2006), pp. 115–126.

[94] Volker Grimm, Uta Berger, Donald L DeAngelis, J Gary Polhill, Jarl Giske, and Steven F Railsback. "The ODD protocol: a review and first update". In: *Ecological modelling* 221.23 (2010), pp. 2760–2768.

[95] Volker Grimm, Steven F Railsback, Christian E Vincenot, Uta Berger, Cara Gallagher, Donald L DeAngelis, Bruce Edmonds, Jiaqi Ge, Jarl Giske, Juergen Groeneveld, et al. "The ODD protocol for describing agent-based and other simulation models: A second update to improve clarity, replication, and structural realism". In: *Journal of Artificial Societies and Social Simulation* 23.2 (2020).

[96] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, and Gang Wang. "Recent advances in convolutional neural networks". In: *arXiv preprint arXiv:1512.07108* (2015).

[97] XC Guo, JH Yang, CG Wu, CY Wang, and YC Liang. "A novel LS-SVMs hyper-parameter selection based on particle swarm optimization". In: *Neurocomputing* 71.16-18 (2008), pp. 3211–3215.

[98] Andrew G Haldane and Robert M May. "Systemic risk in banking ecosystems". In: *Nature* 469.7330 (2011), pp. 351–355.

[99] Peter J Hammond. "Rationality in economics". In: *Rivista internazionale di scienze sociali* 105.3 (1997), pp. 247–288.

[100] Joel Hasbrouck. *Empirical market microstructure: The institutions, economics, and econometrics of securities trading*. Oxford University Press, 2007.

[101] Alex Hawkins-Hooker, Florence Depardieu, Sebastien Baur, Guillaume Couairon, Arthur Chen, and David Bikard. "Generating functional protein variants with variational autoencoders". In: *BioRxiv* (2020).

[102] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[103] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. "Gans trained by a two time-scale update rule converge to a local nash equilibrium". In: *arXiv preprint arXiv:1706.08500* (2017).

[104] Kai Hilpert, Rudolf Volkmer-Engert, Tess Walter, and Robert EW Hancock. "High-throughput generation of small antibacterial peptides with improved activity". In: *Nature Biotechnology* 23.8 (2005), pp. 1008–1012.

[105] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[106] Elad Hoffer, Itay Hubara, and Daniel Soudry. "Train longer, generalize better: closing the generalization gap in large batch training of neural networks". In: *Advances in Neural Information Processing Systems*. 2017, pp. 1731–1741.

[107] Heike Hofmann, Karen Kafadar, and Hadley Wickham. "Letter-value plots: Boxplots for large data". In: *The American Statistican* (2011).

[108] Axel Hollmann, Melina Martinez, Patricia Maturana, Liliana C Semorile, and Paulo C Maffia. "Antimicrobial peptides: interaction with model and biological membranes and synergism with chemical antibiotics". In: *Frontiers in chemistry* 6 (2018), p. 204.

[109] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. "Meta-learning in neural networks: A survey". In: *arXiv preprint arXiv:2004.05439* (2020).

[110] Yong Hu, Kang Liu, Xiangzhou Zhang, Lijun Su, EWT Ngai, and Mei Liu. "Application of evolutionary computation for rule discovery in stock algorithmic trading: A literature review". In: *Applied Soft Computing* 36 (2015), pp. 534–551.

[111] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. "Toward controlled generation of text". In: JMLR. org. 2017, pp. 1587–1596.

[112] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. "Snapshot ensembles: Train 1, get M for free". In: *arXiv preprint arXiv:1704.00109* (2017).

[113] David H Hubel and Torsten N Wiesel. "Receptive fields and functional architecture of monkey striate cortex". In: *The Journal of physiology* 195.1 (1968), pp. 215–243.

[114] Alpha Vantage Inc. *Alpha Vantage*. 2021. URL: https://www.alphavantage.co/.

[115] Alpha Vantage Inc. *Alpha Vantage API Documentation: Intraday (Extended History)*. 2021. URL: https://www.alphavantage.co/documentation/#intraday.

[116] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. 2015, pp. 448–456.

[117] Christian Janiesch, Patrick Zschech, and Kai Heinrich. "Machine learning and deep learning". In: *Electronic Markets* (2021), pp. 1–11.

[118] Stanisław Jastrzebski, Devansh Arpit, Nicolas Ballas, Vikas Verma, Tong Che, and Yoshua Bengio. "Residual Connections Encourage Iterative Inference". In: *arXiv preprint arXiv:1710.04773* (2017).

[119] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. "3D convolutional neural networks for human action recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 35.1 (2013), pp. 221–231.

[120] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. "Junction tree variational autoencoder for molecular graph generation". In: *arXiv preprint arXiv:1802.04364* (2018).

[121] Artur Kadurin, Sergey Nikolenko, Kuzma Khrabrov, Alex Aliper, and Alex Zhavoronkov. "druGAN: an advanced generative adversarial autoencoder model for de novo generation of new molecules with desired molecular properties in silico". In: *Molecular pharmaceutics* 14.9 (2017), pp. 3098–3104.

[122] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. "On large-batch training for deep learning: Generalization gap and sharp minima". In: *arXiv preprint arXiv:1609.04836* (2016).

[123] Meenakshi Khosla, Keith Jamison, Amy Kuceyeski, and Mert Sabuncu. "3D Convolutional Neural Networks for Classification of Functional Connectomes". In: *arXiv preprint arXiv:1806.04209* (2018).

[124] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[125] Diederik P Kingma and Max Welling. "Auto-encoding variational Bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[126] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. "Self-normalizing neural networks". In: *Advances in Neural Information Processing Systems*. 2017, pp. 972–981.

[127] V. Könyves, C. Kiss, A. Moór, Z. T. Kiss, and L. V. Tóth. "Catalogue of far-infrared loops in the Galaxy". In: *Astronomy and Astrophysics* 463 (Mar. 2007), pp. 1227–1234. DOI: 10.1051/0004-6361:20065438. eprint: astro-ph/0610465.

[128] M. R. Krumholz, R. I. Klein, and C. F. McKee. "Radiation-Hydrodynamic Simulations of Collapse and Fragmentation in Massive Protostellar Cores". In: *The Astrophysical Journal* 656 (Feb. 2007), pp. 959–979. DOI: 10.1086/510664. eprint: arXiv:astro-ph/0609798.

[129] Max Kuhn and Kjell Johnson. *Feature engineering and selection: A practical approach for predictive models*. CRC Press, 2019.

[130] Manish Kumar, Ruchi Verma, and Gajendra PS Raghava. "Prediction of mitochondrial proteins using support vector machine and hidden Markov model". In: *Journal of Biological Chemistry* 281.9 (2006), pp. 5357–5363.

[131] Tatu Laine. "Quantitative analysis of financial market infrastructures: further perspectives on financial stability". In: (2015).

[132] Tian Lan, Yuanyuan Li, Jonah Kimani Murugi, Yi Ding, and Zhiguang Qin. "RUN: Residual U-Net for Computer-Aided Detection of Pulmonary Nodules without Candidate Selection". In: *arXiv preprint arXiv:1805.11856* (2018).

[133] Pat Langley. *The changing science of machine learning*. 2011.

[134] Francois Lanusse, Quanbin Ma, Nan Li, Thomas E Collett, Chun-Liang Li, Siamak Ravanbakhsh, Rachel Mandelbaum, and Barnabas Poczos. "CMU DeepLens: Deep Learning For Automatic Image-based Galaxy-Galaxy Strong Lens Finding". In: *arXiv preprint arXiv:1703.02642* (2017).

[135] R. B. Larson. "Turbulence and star formation in molecular clouds." In: *Monthly Notices of the Royal Astronomical Society* 194 (Mar. 1981), pp. 809–826. DOI: 10.1093/mnras/194.4.809.

[136] Ties Latendorf, Ulrich Gerstel, Zhihong Wu, Joachim Bartels, Alexander Becker, Andreas Tholey, and Jens-Michael Schröder. "Cationic intrinsically disordered antimicrobial peptides (CIDAMPs) represent a new paradigm of innate defense with a potential for novel anti-infectives". In: *Scientific reports* 9.1 (2019), pp. 1–25.

[137] Viktória Lázár, Ana Martins, Réka Spohn, Lejla Daruka, Gábor Grézal, Gergely Fekete, Mónika Számel, Pramod K Jangir, Bálint Kintses, Bálint Csörgő, Ákos Nyerges, Ádám Györkei, András Kincses, András Dér, Fruzsina R Walter, Mária A Deli, Edit Urbán, Zsófia Hegedűs, Gábor Olajos, Orsolya Méhi, Balázs Bálint, István Nagy, Tamás A Martinek, Balázs Papp, and Csaba Pál.

"Antibiotic-resistant bacteria show widespread collateral sensitivity to antimicrobial peptides". In: *Nature Microbiology* 3.6 (2018), p. 718.

[138] Blake LeBaron. "A builder's guide to agent-based financial markets". In: *Quantitative finance* 1 (2001), pp. 254–261.

[139] Blake LeBaron. "Agent-based computational finance". In: *Handbook of computational economics* 2 (2006), pp. 1187–1233.

[140] Blake LeBaron. "Active and passive learning in agent-based financial markets". In: *Eastern Economic Journal* 37.1 (2011), pp. 35–43.

[141] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[142] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[143] E. J. Lee, P. Chang, and N. Murray. "Time-varying Dynamical Star Formation Rate". In: *The Astrophysical Journal* 800, 49 (Feb. 2015), p. 49. DOI: 10.1088/0004-637X/800/1/49. eprint: 1406.4148.

[144] H. Li, D. Li, L. Qian, D. Xu, P. F. Goldsmith, A. Noriega-Crespo, Y. Wu, Y. Song, and R. Nan. "Outflows and Bubbles in Taurus: Star-formation Feedback Sufficient to Maintain Turbulence". In: *The Astrophysical Journal Supplement Seriess* 219, 20 (Aug. 2015), p. 20. DOI: 10.1088/0067-0049/219/2/20. eprint: 1507.06512.

[145] Junyi Li, Xintong Wang, Yaoyang Lin, Arunesh Sinha, and Michael Wellman. "Generating realistic stock market order streams". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. AAAI, 2020, pp. 727–734.

[146] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E Gonzalez, Michael I Jordan, and Ion Stoica. "RLlib: Abstractions for Distributed Reinforcement Learning". In: *arXiv preprint arXiv:1712.09381* (2017).

[147] Maggie Lieu, Luca Conversi, Bruno Altieri, and Benoît Carry. "Detecting solar system objects with convolutional neural networks". In: *Monthly Notices of the Royal Astronomical Society* 485.4 (2019), pp. 5831–5842.

[148] Chris J. Lintott, Kevin Schawinski, Anže Slosar, Kate Land, Steven Bamford, Daniel Thomas, M. Jordan Raddick, Robert C. Nichol, Alex Szalay, Dan Andreescu, Phil Murray, and Jan Vandenberg. "Galaxy Zoo: morphologies derived from visual inspection of galaxies from the Sloan Digital Sky Survey". In: *Monthly Notices of the Royal Astronomical Society* 389 (Sept. 2008), pp. 1179–1189. DOI: 10.1111/j.1365-2966.2008.13689.x. eprint: 0804.4483.

[149] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. "An intriguing failing of convolutional neural networks and the coordconv solution". In: *Advances in Neural Information Processing Systems*. 2018, pp. 9605–9616.

[150] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. "Deep Learning Face Attributes in the Wild". In: Dec. 2015.

[151] Andrew W Lo. "The adaptive markets hypothesis". In: *The Journal of Portfolio Management* 30.5 (2004), pp. 15–29.

[152] Christopher Loose, Kyle Jensen, Isidore Rigoutsos, and Gregory Stephanopoulos. "A linguistic model for the rational design of antimicrobial peptides". In: *Nature* 443.7113 (2006), pp. 867–869.

[153] Ilya Loshchilov and Frank Hutter. "CMA-ES for hyperparameter optimization of deep neural networks". In: *arXiv preprint arXiv:1604.07269* (2016).

[154] Ananth Madhavan. "Market microstructure: A survey". In: *Journal of financial markets* 3.3 (2000), pp. 205–258.

[155] Margit Mahlapuu, Joakim Håkansson, Lovisa Ringstad, and Camilla Björn. "Antimicrobial peptides: an emerging category of therapeutic agents". In: *Frontiers in cellular and infection microbiology* 6 (2016), p. 194.

[156] Burton G Malkiel and Eugene F Fama. "Efficient capital markets: A review of theory and empirical work". In: *The journal of Finance* 25.2 (1970), pp. 383–417.

[157] Alexandra K Marr, William J Gooderham, and Robert EW Hancock. "Antibacterial peptides for therapeutic use: obstacles and realistic outlook". In: *Current Opinion in Pharmacology* 6.5 (2006), pp. 468–472.

[158] Tshilidzi Marwala. "Rational Choice and Artificial Intelligence". In: *arXiv preprint arXiv:1703.10098* (2017).

[159] Dominic Masters and Carlo Luschi. "Revisiting Small Batch Training for Deep Neural Networks". In: *arXiv preprint arXiv:1804.07612* (2018).

[160] Deepika Mathur, Sandeep Singh, Ayesha Mehta, Piyush Agrawal, and Gajendra PS Raghava. "In silico approaches for predicting the half-life of natural and modified peptides in blood". In: *PLOS ONE* 13.6 (2018), e0196829.

[161] Daniel Maturana and Sebastian Scherer. "VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2015, pp. 922–928.

[162] John McDermott. "Domain knowledge and the design process". In: *18th Design Automation Conference*. IEEE. 1981, pp. 580–588.

[163] Frank McGroarty, Ash Booth, Enrico Gerding, and VL Raju Chinthalapati. "High frequency trading strategies, market fragility and price spikes: an agent based model perspective". In: *Annals of Operations Research* 282.1 (2019), pp. 217–244.

[164] A. Men'shchikov. "A multi-scale filament extraction method: getfilaments". In: *Astronomy and Astrophysics* 560, A63 (Dec. 2013), A63. DOI: 10.1051/0004-6361/201321885. eprint: 1309.2170.

[165] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. "Which training methods for GANs do actually converge?" In: *arXiv preprint arXiv:1801.04406* (2018).

[166] Dave Michaels and Alexander Osipovich. *Appeals Court Rules for Stock Exchanges in Fee Fight With SEC*. June 16, 2020. URL: https://www.wsj.com/articles/appeals-court-rules-for-stock-exchanges-in-fee-fight-with-sec-11592322391.

[167] Mehdi Mirza and Simon Osindero. "Conditional generative adversarial nets". In: *arXiv preprint arXiv:1411.1784* (2014).

[168] Catherine Mooney, Niall J Haslam, Thérèse A Holton, Gianluca Pollastri, and Denis C Shields. "PeptideLocator: prediction of bioactive peptides in protein sequences". In: *Bioinformatics* 29.9 (2013), pp. 1120–1126.

[169] Catherine Mooney, Niall J Haslam, Gianluca Pollastri, and Denis C Shields. "Towards the improved discovery and design of functional peptides: common features of diverse classes permit generalized prediction of bioactivity". In: *PLOS ONE* 7.10 (2012), e45012.

[170] Catherine Mooney, Yong- Hong Wang, and Gianluca Pollastri. "SCLpred: protein subcellular localization prediction by N-to-1 neural networks". In: *Bioinformatics* 27.20 (2011), pp. 2812–2819.

[171] Alex T Müller, Gisela Gabernet, Jan A Hiss, and Gisbert Schneider. "modlAMP: Python for antimicrobial peptides". In: *Bioinformatics* 33.17 (2017), pp. 2753–2755.

[172] Nikhil Muralidhar, Mohammad Raihanul Islam, Manish Marwah, Anuj Karpatne, and Naren Ramakrishnan. "Incorporating prior domain knowledge into deep neural networks". In: *2018 IEEE International Conference on Big Data (Big Data)*. IEEE. 2018, pp. 36–45.

[173] W James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. "Interpretable machine learning: definitions, methods, and applications". In: *arXiv preprint arXiv:1901.04592* (2019).

[174] Deepesh Nagarajan, Tushar Nagarajan, Natasha Roy, Omkar Kulkarni, Sathyabaarathi Ravichandran, Madhulika Mishra, Dipshikha Chakravortty, and Nagasuma Chandra. "Computational antimicrobial peptide design and evaluation against multidrug-resistant clinical isolates of bacteria". In: *Journal of Biological Chemistry* 293.10 (2018), pp. 3492–3509. ISSN: 0021-9258. DOI: https://doi.org/10.1074/jbc.M117.805499.

[175] Vaishnavh Nagarajan and J Zico Kolter. "Gradient descent GAN optimization is locally stable". In: *Advances in neural information processing systems*. 2017, pp. 5585–5595.

[176] G. Narayanan, R. Snell, and A. Bemis. "Molecular outflows identified in the FCRAO CO survey of the Taurus Molecular Cloud". In: *Monthly Notices of the Royal Astronomical Society* 425 (Oct. 2012), pp. 2641–2667. DOI: 10.1111/j.1365-2966.2012.21579.x. eprint: 1206.5708.

[177] Yomna Nasser, Peter Eckersley, Yann Bayle, Owain Evans, Gennie Gebhart, and Dustin Schwenk. *Measuring the Progress of AI Research*. https://www.eff.org/ai/metrics. 2017.

[178] Anova Financial Networks. *Low Latency Financial Connectivity*. 2021. URL: https://anovanetworks.com.

[179] Maureen O'hara. *Market microstructure theory*. Wiley, 1997.

[180] S. S. R. Offner and H. G. Arce. "Impact of Winds from Intermediate-mass Stars on Molecular Cloud Structure and Turbulence". In: *The Astrophysical Journal* 811, 146 (Oct. 2015), p. 146. DOI: 10.1088/0004-637X/811/2/146. eprint: 1508.07008.

[181] S. S. R. Offner and J. Chaban. "Impact of Protostellar Outflows on Turbulence and Star Formation Efficiency in Magnetized Dense Cores". In: *The Astrophysical Journal* 847, 104 (Oct. 2017), p. 104. DOI: 10.3847/1538-4357/aa8996. eprint: 1709.01086.

[182] S. S. R. Offner, P. C. Clark, P. Hennebelle, N. Bastian, M. R. Bate, P. F. Hopkins, E. Moraux, and A. P. Whitworth. "The Origin and Universality of the Stellar Initial Mass Function". In: *Protostars and Planets VI* (2014), pp. 53–75. DOI: 10.2458/azu_uapress_9780816531240-ch003. eprint: 1312.5326.

[183] S. S. R. Offner, M. M. Dunham, K. I. Lee, H. G. Arce, and D. B. Fielding. "The Turbulent Origin of Outflow and Spin Misalignment in Multiple Star Systems". In: *The Astrophysical Journal Letters* 827, L11 (Aug. 2016), p. L11. DOI: 10.3847/2041-8205/827/1/L11. eprint: 1606.08445.

[184] S. S. R. Offner, E. J. Lee, A. A. Goodman, and H. Arce. "Radiation-hydrodynamic Simulations of Protostellar Outflows: Synthetic Observations and Data Comparisons". In: *The Astrophysical Journal* 743, 91 (Dec. 2011), p. 91. DOI: 10.1088/0004-637X/743/1/91. eprint: 1110.5640.

[185] S. S. R. Offner and Y. Liu. "Turbulent action at a distance due to stellar feedback in magnetized clouds". In: *Nature Astronomy* 2 (Sept. 2018), pp. 896–900. DOI: 10.1038/s41550-018-0566-1. eprint: 1809.03513.

[186] Mark Paddrik, Roy Hayes, William Scherer, and Peter Beling. "Effects of limit order book information level on market stability metrics". In: *Journal of Economic Interaction and Coordination* 12.2 (2017), pp. 221–247.

[187] Adrian Pagan. "The econometrics of financial markets". In: *Journal of empirical finance* 3.1 (1996), pp. 15–102.

[188] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* 22.10 (2010), pp. 1345–1359.

[189] Guim Perarnau, Joost Van De Weijer, Bogdan Raducanu, and Jose M Álvarez. "Invertible conditional gans for image editing". In: *arXiv preprint arXiv:1611.06355* (2016).

[190] Malak Pirtskhalava, Andrei Gabrielian, Phillip Cruz, Hannah L Griggs, R Burke Squires, Darrell E Hurt, Maia Grigolava, Mindia Chubinidze, George Gogoladze, Boris Vishnepolsky, Vsevolod Alekseev, Alex Rosenthal, and Michael Tartakovsky. "DBAASP v. 2: an enhanced database of structure and antimicrobial/cytotoxic activity of natural and synthetic peptides". In: *Nucleic Acids Research* 44.D1 (2016), pp. D1104–D1112.

[191] Marc Potters and Jean-Philippe Bouchaud. "More statistical properties of order books and price impact". In: *Physica A: Statistical Mechanics and its Applications* 324.1-2 (2003), pp. 133–140.

[192] David Martin Powers. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation". In: (2011).

[193] Chris Preist and Maarten van Tol. "Adaptive agents in a persistent shout double auction". In: *Proceedings of the first international conference on Information and computation economies*. ACM, 1998, pp. 11–18.

[194] JR Primack, A Dekel, DC Koo, S Lapiner, D Ceverino, RC Simons, GF Snyder, M Bernardi, Z Chen, H Domínguez-Sànchez, et al. "Deep learning identifies high-z galaxies in a central blue nugget phase in a characteristic mass range". In: *The Astrophysical Journal* 858.2 (2018), p. 114.

[195] Abid Qureshi, Nishant Thakur, Himani Tandon, and Manoj Kumar. "AVPdb: a database of experimentally validated antiviral peptides targeting medically important viruses". In: *Nucleic Acids Research* 42.D1 (2014), pp. D1147–D1153.

[196] William Rand and Uri Wilensky. "Verification and validation through replication: A case study using Axelrod and Hammond's ethnocentrism model". In: *North American Association for Computational Social and Organization Sciences (NAACSOS)* (2006), pp. 1–6.

[197] Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Peter Chen, John Canny, Pieter Abbeel, and Yun Song. "Evaluating protein transfer learning with TAPE". In: *Advances in Neural Information Processing Systems*. 2019, pp. 9689–9701.

[198] KVR Reddy, RD Yedery, and C Aranha. "Antimicrobial peptides: premises and promises". In: *International Journal of Antimicrobial Agents* 24.6 (2004), pp. 536–547.

[199] N. A. Ridge, J. Di Francesco, H. Kirk, D. Li, A. A. Goodman, J. F. Alves, H. G. Arce, M. A. Borkin, P. Caselli, J. B. Foster, M. H. Heyer, D. Johnstone, D. A. Kosslyn, M. Lombardi, J. E. Pineda, S. L. Schnee, and M. Tafalla. "The COMPLETE Survey of Star-Forming Regions: Phase I Data". In: *The Astronomical Journal* 131 (June 2006), pp. 2921–2933. DOI: 10.1086/503704. eprint: arXiv:astro-ph/0602542.

[200] Michael Rollins and Dave Cliff. "Which Trading Agent is Best? Using a Threaded Parallel Simulation of a Financial Market Changes the Pecking-Order". In: *arXiv preprint arXiv:2009.06905* (2020).

[201] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[202] Kevin Roth, Aurelien Lucchi, Sebastian Nowozin, and Thomas Hofmann. "Stabilizing training of generative adversarial networks through regularization". In: *Advances in neural information processing systems*. 2017, pp. 2018–2028.

[203] Serge Ruden, Annika Rieder, Thomas Schwartz, Ralf Mikut, Kai Hilpert, and Irina Chis Ster. "Synergy pattern of short cationic antimicrobial peptides against multidrug-resistant Pseudomonas aeruginosa". In: *Frontiers in Microbiology* 10 (2019), p. 2740.

[204] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[205] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[206] Hossein Sabzian, Mohammad Ali Shafia, Ali Maleki, Seyeed Mostapha Seyeed Hashemi, Ali Baghaei, and Hossein Gharib. "Theories and practice of agent based modeling: Some practical implications for economic planners". In: *arXiv preprint arXiv:1901.08932* (2019).

[207] Mrinmaya Sachan, Kumar Avinava Dubey, Tom M Mitchell, Dan Roth, and Eric P Xing. "Learning pipelines with limited data and domain knowledge: A study in parsing physics problems". In: *Advances in Neural Information Processing Systems* 31 (2018), pp. 140–151.

[208] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training gans". In: *Advances in neural information processing systems*. 2016, pp. 2234–2242.

[209] Ivy Schmerken. *Quants Demand More Efficient Alpha Generation Technology Platform*. Accessed 2021-03-16. July 10, 2008. URL: https://web.archive.org/web/20110718225953/http://www.wallstreetandtech.com/asset-management/showArticle.jhtml?articleID=208808533.

[210] J Schmidhuber. "Making the world differentiable: On using fully recurrent self-supervised neural networks for dynamic reinforcement learning and planning in non-stationary environments". In: *Institut für Informatik, Technische Universität München. Technical Report FKI-126* 90 (1990).

[211] Jürgen Schmidhuber. "A possibility for implementing curiosity and boredom in model-building neural controllers". In: 1991, pp. 222–227.

[212] L Julian Schvartzman and Michael P Wellman. "Stronger CDA strategies through empirical game-theoretic analysis and reinforcement learning". In: *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 249–256.

[213] U.S. Securities and Exchanges Commission. *Findings Regarding The Market Events of May 6, 2010*. 2010. URL: https://www.sec.gov/files/marketevents-report.pdf.

[214] U.S. Securities and Exchanges Commission. *How to Submit Comments*. 2021. URL: https://www.sec.gov/rules/submitcomments.htm.

[215] U.S. Securities and Exchanges Commission. *Tick Size Pilot Program*. 2021. URL: https://www.sec.gov/ticksizepilot.

[216] US Securities and Exchanges Commission. "Regulation National Market System". In: (2005).

[217] Tom Sercu, Sebastian Gehrmann, Hendrik Strobelt, Payel Das, Inkit Padhi, Cicero Dos Santos, Kahini Wadhawan, and Vijil Chenthamarakshan. "Interactive Visual Exploration of Latent Space (IVELS) for peptide auto-encoder model selection". In: (2019).

[218] Martin Sewell. "Characterization of financial time series". In: *Rn* 11.01 (2011), p. 01.

[219] R. J. Simpson, M. S. Povich, S. Kendrew, C. J. Lintott, E. Bressert, K. Arvidsson, C. Cyganowski, S. Maddison, K. Schawinski, R. Sherman, A. M. Smith, and G. Wolf-Chase. "The Milky Way Project First Data Release: a bubblier Galactic disc". In: *Monthly Notices of the Royal Astronomical Society* 424 (Aug. 2012), pp. 2442–2460. DOI: 10.1111/j.1365-2966.2012.20770.x. eprint: 1201.6357.

[220] Justin Sirignano and Rama Cont. "Universal features of price formation in financial markets: perspectives from deep learning". In: *Quantitative Finance* 19.9 (2019), pp. 1449–1459.

[221] Eric Smith, J Doyne Farmer, László Gillemot, and Supriya Krishnamurthy. "Statistical theory of the continuous double auction". In: *Quantitative finance* 3 (2003), pp. 481–514.

[222] Samuel L Smith, Pieter-Jan Kindermans, and Quoc V Le. "Don't Decay the Learning Rate, Increase the Batch Size". In: *arXiv preprint arXiv:1711.00489* (2017).

[223] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems*. 2012, pp. 2951–2959.

[224] Kimmo Soramäki, Morten L Bech, Jeffrey Arnold, Robert J Glass, and Walter E Beyeler. "The Topology of Interbank Payment Flows". In: *Federal Reserve Bank of New York: Staff Reports* (2006).

[225] Erik Sorensen, Ryan Ozzello, Rachael Rogan, Ethan Baker, Nate Parks, and Wei Hu. "Meta-Learning of Evolutionary Strategy for Stock Trading". In: *Journal of Data Analysis and Information Processing* 8.2 (2020), pp. 86–98.

[226] Réka Spohn, Lejla Daruka, Viktória Lázár, Ana Martins, Fanni Vidovics, Gábor Grézal, Orsolya Méhi, Bálint Kintses, Mónika Számel, Pramod K Jangir, Bálint Csörgő, Ádám Györkei, Zoltán Bódi, Anikó Faragó, László Bodai, Imre Földesi, Diána Kata, Gergely Maróti, Bernadett Pap, Roland Wirth, Papp Balázs, and Csaba Pál. "Integrated evolutionary analysis reveals antimicrobial peptides with limited resistance". In: *Nature Communications* 10.1 (2019), pp. 1–13.

[227] Akash Srivastava, Lazar Valkov, Chris Russell, Michael U Gutmann, and Charles Sutton. "Veegan: Reducing mode collapse in gans using implicit variational learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 3308–3318.

[228] Steve Stotter, John Cartlidge, and Dave Cliff. "Behavioural investigations of financial trading agents using Exchange Portal (ExPo)". In: *Transactions on Computational Collective Intelligence XVII*. Springer, 2014, pp. 22–45.

[229] Harish Subramanian, Subramanian Ramamoorthy, Peter Stone, and Benjamin J Kuipers. "Designing safe, profitable automated stock trading agents using evolutionary algorithms". In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM, 2006, pp. 1777–1784.

[230] Rodrigue Talla Kuate. "Hierarchical reinforcement learning for trading agents". PhD thesis. 2016.

[231] Gerald Tesauro and Jonathan L Bredin. "Strategic sequential bidding in auctions using dynamic programming". In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2002, pp. 591–598.

[232] Gerald Tesauro and Rajarshi Das. "High-performance bidding agents for the continuous double auction". In: *Proceedings of the 3rd ACM Conference on Electronic Commerce*. ACM, 2001, pp. 206–209.

[233] Shaini Thomas, Shreyas Karnik, Ram Shankar Barai, Vaidyanathan K Jayaraman, and Susan Idicula-Thomas. "CAMP: a useful resource for research on antimicrobial peptides". In: *Nucleic Acids Research* 38.suppl_1 (2009), pp. D774–D780.

[234] Brian F Tivnan, David Rushing Dewhurst, Colin M Van Oort, John H Ring IV, Tyler J Gray, Brendan F Tivnan, Matthew TK Koehler, Matthew T McMahon, David M Slater, Jason G Veneman, et al. "Fragmentation and inefficiencies in US equity markets: Evidence from the Dow 30". In: *PloS one* 15.1 (2020), e0226968.

[235] Brian F Tivnan, Matthew TK Koehler, David Slater, Jason Veneman, and Brendan F Tivnan. "Towards a model of the US stock market: How important is the securities information processor?" In: *2017 Winter Simulation Conference (WSC)*. IEEE. 2017, pp. 1181–1192.

[236] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. "Learning spatiotemporal features with 3d convolutional networks". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4489–4497.

[237] Andrejs Tucs, Duy Phuoc Tran, Akiko Yumoto, Yoshihiro Ito, Takanori Uzawa, and Koji Tsuda. "Generating Ampicillin-Level Antimicrobial Peptides with Activity-Aware Generative Adversarial Networks". In: *ACS Omega* 5.36 (2020), pp. 22847–22851. DOI: 10.1021/acsomega.0c02088. eprint: https://doi.org/10.1021/acsomega.0c02088.

[238] Colin M Van Oort, Duo Xu, Stella SR Offner, and Robert A Gutermuth. "Casi: A convolutional neural network approach for shell identification". In: *The Astrophysical Journal* 880.2 (2019), p. 83.

[239] Colin M. Van Oort. "Market Efficiency in US Stock Markets: A Study of the Dow 30 and the S&P 30". In: *Graduate College Dissertations and Theses* (2018).

[240] Colin M. Van Oort. *CASI-2D*. https://doi.org/10.5281/zenodo.2695533. Feb. 2019. DOI: 10.5281/zenodo.2695533.

[241] Colin M. Van Oort. *Agent Based Market Microstructure Simulation*. Accessed 2021/04/20. 2021. URL: https://gitlab.com/computational-finance-lab/abmms.

[242] Colin M. Van Oort. *AMP-GAN*. https://gitlab.com/vail-uvm/amp-gan, Accessed 2020/08/30.

[243] Colin M. Van Oort, Jonathon B. Ferrell, Jacob M. Remington, Safwan Wshah, and Jianing Li. "AMPGAN v2: Machine Learning-Guided Design of Antimicrobial Peptides". In: *Journal of Chemical Information and Modeling* 61.5 (2021). PMID: 33787250, pp. 2198–2207. DOI: 10.1021/acs.jcim.0c01441. eprint: https://doi.org/10.1021/acs.jcim.0c01441.

[244] Colin M. Van Oort, Brian F. Tivnan, and Safwan Wshah. "Adaptive Agents and Data Quality in Agent-Based Financial Markets". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* (2021). Under review, Submitted 2021/06/11.

[245] Xavier Vives. "How fast do rational agents learn?" In: *The Review of Economic Studies* 60.2 (1993), pp. 329–347.

[246] Khuong Vo, Dang Pham, Mao Nguyen, Trung Mai, and Tho Quan. "Combination of domain knowledge and deep learning for sentiment analysis". In: *International Workshop on Multi-disciplinary Trends in Artificial Intelligence*. Springer. 2017, pp. 162–173.

[247] Perukrishnen Vytelingum, Dave Cliff, and Nicholas R Jennings. "Strategic bidding in continuous double auctions". In: *Artificial Intelligence* 172.14 (2008), pp. 1700–1729.

[248] Faiza Hanif Waghu, Ram Shankar Barai, Pratima Gurung, and Susan Idicula-Thomas. "CAMPR3: a database on sequences, structures and signatures of antimicrobial peptides". In: *Nucleic acids research* 44.D1 (2016), pp. D1094–D1097.

[249] Elaine Wah and Michael P Wellman. "Latency arbitrage in fragmented markets: A strategic agent-based analysis". In: *Algorithmic Finance* 5.3-4 (2016), pp. 69–93.

[250] Elaine Wah, Mason Wright, and Michael P Wellman. "Welfare effects of market making in continuous double auctions". In: *Journal of Artificial Intelligence Research* 59 (2017), pp. 613–650.

[251] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. "Learning to reinforcement learn". In: *arXiv preprint arXiv:1611.05763* (2016).

[252] P. Wang, Z.-Y. Li, T. Abel, and F. Nakamura. "Outflow Feedback Regulated Massive Star Formation in Parsec-Scale Cluster-Forming Clumps". In: *The Astrophysical Journal* 709 (Jan. 2010), pp. 27–41. DOI: 10.1088/0004-637X/709/1/27. eprint: 0908.4129.

[253] Yanbin Wang, Zhu-Hong You, Shan Yang, Xiao Li, Tong-Hai Jiang, and Xi Zhou. "A high efficient biological language model for predicting protein–protein interactions". In: *Cells* 8.2 (2019), p. 122.

[254] J. P. Williams, E. J. de Geus, and L. Blitz. "Determining structure in molecular clouds". In: *The Astrophysical Journal* 428 (June 1994), pp. 693–712. DOI: 10.1086/174279.

[255] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. "The marginal value of adaptive gradient methods in machine learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 4148–4158.

[256] Jacob Witten and Zack Witten. "Deep learning regression model for antimicrobial peptide design". In: *BioRxiv* (2019), p. 692681.

[257] Aaron Wray, Matthew Meades, and Dave Cliff. "Automated Creation of a High-Performing Algorithmic Trader via Deep Learning on Level-2 Limit Order Book Data". In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2020, pp. 1067–1074.

[258] Xiaorong Xiang, Ryan Kennedy, Gregory Madey, and Steve Cabaniss. "Verification and validation of agent-based scientific simulation models". In: *Agent-directed simulation conference*. Vol. 47. The European Modeling and Simulation Symposium, 2005, p. 55.

[259] Xuan Xiao, Pu Wang, Wei-Zhong Lin, Jian-Hua Jia, and Kuo-Chen Chou. "iAMP-2L: a two-level multi-label classifier for identifying antimicrobial peptides and their functional types". In: *Analytical Biochemistry* 436.2 (2013), pp. 168–177.

[260] Xiaozheng Xie, Jianwei Niu, Xuefeng Liu, Zhengsu Chen, Shaojie Tang, and Shui Yu. "A Survey on Incorporating Domain Knowledge into Deep Learning for Medical Image Analysis". In: *Medical Image Analysis* (2021), p. 101985.

[261] D. Xu and S. S. R. Offner. "Assessing the Performance of a Machine Learning Algorithm in Identifying Bubbles in Dust Emission". In: *The Astrophysical Journal* 851, 149 (Dec. 2017), p. 149. DOI: 10.3847/1538-4357/aa9a42. eprint: 1711.03480.

[262] Hong Yan and Robert EW Hancock. "Synergistic interactions between mammalian antimicrobial defense peptides". In: *Antimicrobial Agents and Chemotherapy* 45.5 (2001), pp. 1558–1560.

[263] Jianwei Yang, Anitha Kannan, Dhruv Batra, and Devi Parikh. "Lr-gan: Layered recursive generative adversarial networks for image generation". In: *arXiv preprint arXiv:1703.01560* (2017).

[264] Zijiang Yang, Reda Al-Bahrani, Andrew CE Reid, Stefanos Papanikolaou, Surya R Kalidindi, Wei-keng Liao, Alok Choudhary, and Ankit Agrawal. "Deep learning based domain knowledge integration for small datasets: Illustrative applications in materials informatics". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–8.

[265] Changchang Yin, Rongjian Zhao, Buyue Qian, Xin Lv, and Ping Zhang. "Domain Knowledge guided deep learning with electronic health records". In: *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2019, pp. 738–747.

[266] Guozhi Yu, Desiree Y Baeder, Roland R Regoes, and Jens Rolff. "Combination effects of antimicrobial peptides". In: *Antimicrobial Agents and Chemotherapy* 60.3 (2016), pp. 1717–1724.

[267] Ting Yu, Tony Jan, Simeon Simoff, and John Debenham. “Incorporating prior domain knowledge into inductive machine learning”. In: *Unpublished doctoral dissertation Computer Sciences* (2007), pp. 30–44.

[268] Zhengxin Zhang, Qingjie Liu, and Yunhong Wang. “Road extraction by deep residual u-net”. In: *IEEE Geoscience and Remote Sensing Letters* (2018).

[269] Maria S Zharkova, Dmitriy S Orlov, Olga Yu Golubeva, Oleg B Chakchir, Igor E Eliseev, Tatiana M Grinchuk, and Olga V Shamova. “Application of antimicrobial peptides of the innate immune system in combination with conventional antibiotics–a novel way to combat antibiotic resistance?” In: *Frontiers in Cellular and Infection Microbiology* 9 (2019), p. 128.

[270] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists*. "O’Reilly Media, Inc.", 2018.

[271] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. “Generative visual manipulation on the natural image manifold”. In: Springer. 2016, pp. 597–613.

[272] Wentao Zhu, Yufang Huang, Hui Tang, Zhen Qian, Nan Du, Wei Fan, and Xiaohui Xie. “AnatomyNet: Deep 3D Squeeze-and-excitation U-Nets for fast and fully automated whole-volume anatomical segmentation”. In: *arXiv preprint arXiv:1808.05238* (2018).