5-2024

Advancing Game Development and AI Integration: An Extensible Game Engine With Integrated AI Support for Real-World Deployment and Efficient Model Development

Ryan Anderson
Utah State University, ryan.anderson@usu.edu

ADVANCING GAME DEVELOPMENT AND AI INTEGRATION: AN EXTENSIBLE GAME ENGINE WITH INTEGRATED AI SUPPORT FOR REAL-WORLD DEPLOYMENT AND EFFICIENT MODEL DEVELOPMENT

by

Ryan Anderson

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Mario Harper, Ph.D. Major Professor	Dean Mathias, Ph.D. Committee Member
Steve Petruzza, Ph.D. Committee Member	D. Richard Cutler, Ph.D. Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2024

ABSTRACT

Advancing Game Development and AI Integration: An Extensible Game Engine with Integrated AI Support for Real-World Deployment and Efficient Model Development

by

Ryan Anderson, Master of Science

Utah State University, 2024

Major Professor: Mario Harper, Ph.D.
Department: Computer Science

In pursuit of advancing the field of game development and artificial intelligence (AI), this thesis proposes the creation of Acacia, an innovative and extensible game engine featuring integrated AI support. Acacia supports dual-flagging mechanisms, allowing developers to apply Reinforcement Learning (RL) algorithms and designate entities for RL analysis based on both game state and perceived "reward" criteria. Additionally, developers have precise control over components that are made visible to the RL algorithm, such as only those visible within the camera focus. This granularity refines AI interactions to align with real player behavior. A customizable plugin was developed to enhance flexibility, integrating the ability to implement RL algorithms easily into the game.

As a tangible showcase of Acacia's capabilities, the thesis includes the development of three distinct games spanning various genres, each leveraging the AI plugin. The primary objective is to establish an extensible game engine capable of constructing any 2D game with inherent RL support. Beyond this, Acacia seeks to reduce the limitations of existing solutions, such as Unity, by embracing ML framework agnosticism. Unlike Unity, which confines developers to its particular ML library (Unity RL), Acacia accommodates

frameworks like PyTorch or Tensorflow, as well as fostering adaptability to emerging ML frameworks.

The importance of this project is highlighted by the common challenge of deploying Unity ML algorithms outside the Unity environment, such as in robotics and simulation environment applications. Acacia aims to provide a practical solution, allowing developers to simulate scenarios within the engine and deploy their trained algorithms in real-world applications.

Furthermore, Acacia's architecture enables the creation of multiple instances of the same virtual world, incorporating the capability to disable rendering of all of them at once. In scenarios where rendering is enabled for numerous instances, the system can be utilized to render only a single instance among hundreds, preventing undue strain on system resources. This approach leads to accelerated training times, simulating hundreds of hours within a fraction of the conventional timeframe, markedly enhancing the efficiency of AI model development within the game engine. Acacia is open-source and available at https://github.com/sonorousduck/Acacia.

(60 pages)

PUBLIC ABSTRACT

Advancing Game Development and AI Integration: An Extensible Game Engine with Integrated AI Support for Real-World Deployment and Efficient Model Development

Ryan Anderson

This thesis introduces Acacia, a game engine with built-in artificial intelligence (AI) capabilities. Acacia allows game developers to effortlessly incorporate Reinforcement Learning (RL) algorithms into their creations. By tagging game elements to convey information about the game state or rewards, developers gain precise control over how RL algorithms interact with their games, mirroring real player behavior or providing full knowledge of the game world.

To showcase Acacia's versatility, the thesis presents three games across different genres, each demonstrating the engine's AI plugin. The goal is to establish Acacia as a preferred resource for creating 2D games with RL support without confining developers to specific machine-learning libraries, ensuring adaptability to current and emerging frameworks.

A key advantage of Acacia is its flexibility in deployment, addressing common challenges encountered in other simulators like Unity. Unlike Unity's confined ML library, Acacia enables developers to deploy trained models beyond the simulation environment, even onto real-world objects such as robots.

Moreover, Acacia's architecture facilitates efficient training by enabling multiple instances of the same virtual world and smart rendering optimization. This strategic approach accelerates training times, allowing for hundreds of hours of simulation within a fraction of the usual timeframe, significantly enhancing AI model development efficiency within the game engine.

To my wife, family, and friends

ACKNOWLEDGMENTS

I would like to thank Dr. Mario Harper for all the hours he spent mentoring and guiding me in my research and this thesis. I would also like to thank Dr. Dean Mathias for the many, many questions answered on proper game engine design and rendering, and Dr. Steve Petruzza for support and assistance along the way.

Ryan Anderson

CONTENTS

Page

ABSTRACT . . . iii

PUBLIC ABSTRACT . . . v

ACKNOWLEDGMENTS . . . vii

LIST OF FIGURES . . . x

ACRONYMS . . . xiii

1 INTRODUCTION . . . 1

2 GAME ENGINE . . . 3
- 2.1 Entity-Component-System Architecture . . . 3
- 2.2 Graphics . . . 4
- 2.3 Particle System . . . 5
- 2.4 Audio System . . . 6
- 2.5 Input System . . . 7
- 2.6 Collision Detection . . . 8
  - 2.6.1 Uniform Grids . . . 8
  - 2.6.2 Spatial Partitioning Trees . . . 9
  - 2.6.3 Quadtrees . . . 10
- 2.7 Value of Acacia . . . 11
- 2.8 Reinforcement Learning Use Cases . . . 12

3 Save Money, Get Charged: Facility-Tied Energy Management with Unknown and Unscheduled EV Charging . . . 14
- 3.1 Foundational Algorithms . . . 15
  - 3.1.1 Facility Associated Energy Management Systems . . . 15
  - 3.1.2 Transformers . . . 17
  - 3.1.3 Bayesian Updating . . . 18
- 3.2 EVSE Aware EMS Module Implementation . . . 19
  - 3.2.1 Energy Usage Prediction . . . 19
  - 3.2.2 Battery State of Charge Prediction . . . 20
  - 3.2.3 Car Prediction . . . 22
  - 3.2.4 Battery Charging Simulation + Curtailment Setting . . . 22
- 3.3 Software Implementation . . . 23
- 3.4 Simulation Experiments . . . 24
- 3.5 Results . . . 25
- 3.6 Conclusions and Future Work . . . 26
- 3.7 Acknowledgements . . . 28

4 REINFORCEMENT LEARNING 29
4.1 Deep Q Networks (DQN) 29
4.2 Proximal Policy Optimization (PPO) 30
4.3 Python Scripting 31
4.4 Parallel Instances of an Environment 32
4.5 Developed Games 32
4.5.1 Brick Breaker 32
4.5.2 Crypt 35
4.5.3 SpaceGuy 37
4.6 Conclusions and Future Work 40

REFERENCES 42

APPENDICES 46

CURRICULUM VITAE 47

LIST OF FIGURES

Figure Page

2.1 Demonstration of dynamic layers. In this case, the rendering layer cannot be predetermined to render the player in front of some objects and behind other objects based on position. The dynamic layer allows for objects, such as players, to be rendered behind and in front of objects based on criteria a developer creates (In this case, rendering position is based on the player's Y position.) 5

2.2 A representation of a uniform grid. The left side represents the world partition and the right side depicts the grid locations checked for potential collisions when examining the red circle's position. This reduces the potential collision count from 6 to 3, leading to computational savings. 8

2.3 A representation of a KD-Tree. The left side represents the world partition and the right side is the data structure that is formed. This splits the world depending on the entities found in every frame. 9

2.4 A representation of a quadtree. The left side represents the world partition and the right side is the data structure that is formed. This allows quick traversal to find potential neighbors with which to compare. 10

3.1 Example of a peak load event at the Electric Vehicle and Roadway Research Facility (EVR). Utility charges for the facility power are based on the maximum recorded power draw during the 1-month billing period. During peak times, uncontrolled EV charging events can easily add significant costs. The EMS EVSE module is designed to curtail efficiently, predict, and respond to these high-cost events while allowing vehicles to charge without interruption. 16

3.2 Electric Vehicle and Roadway Research Facility (EVR) and the ABB DC-Fast Charge EVSE units. The EVSE EMS module was developed based on the needs of the EVR facility. This tool was successfully deployed on hardware and integrated with the existing EMS services. Some data collection and logging of full-facility energy usage, charging data, and solar information were conducted as part of the development. 17

3.3 Structure of the transformer architecture used in the EVSE EMS module development. Transformers leverage multi-head attention to parse historical sequence information allowing learning to be more resilient against "forgetting" as it generates new sequences of information. This provides an advantage in EMS applications over similar tools, such as LSTMs, for sequence generation. 19

3.4 Comparisons of energy prediction cases. Four energy usage predictions are shown, contrasting the real usage against transformer predictions over a three-hour window. 20

3.5 Estimations of SoC prior to a charge event starting. 1,000,000 random samples were generated from three distributions representing three classes of people operating their EVs at different SoCs before charging. 21

3.6 Training performance of the EV detection and identification model. Accuracy and F1 are shown over the 80 training epochs, attaining 90% accuracy on validation data and a test accuracy of 87%. 23

3.7 OCPP command that is then sent to a REST API which records all curtailments. The OCPP websocket queries the REST server every few seconds to look for commands it should apply. This command would set the curtailment to 20 kW and have a duration of 60,000 seconds. 24

3.8 Energy usage at the EVR and curtailments on the DC-Fast Charge units installed at the EVR facility. As solar generation provides negative power during peak solar, available power to chargers increases, raising the curtailment (blue). 25

4.1 An image of the Brick Breaker main game screen. Points are earned for destroying bricks and the game is terminated when the ball goes out of the screen on the bottom side. The red represents the walls (including the bottom wall that is used for termination criteria) 33

4.2 Depicted are the values for breaking each brick. The first two rows (purple) are worth 10 points, increasing from there to the last row (blue) worth 40 points. 34

4.3 BrickBreaker's tensorboard visualization of training curves. The left figure indicates the episode's mean length and the right figure indicates the episode's mean reward. The increase in reward and length indicates training. 35

4.4 An image of Crypt on the main gameplay screen. Points are earned by defeating monsters and running further distances. The bats deal 1 damage each hit and the birds deal 3 damage but die on impact. Termination is based on the life count being reduced to zero (Loss), or if the entirety of the stage is run (Victory). 36

4.5 Crypt's tensorboard visualization of training curves. The left figure indicates the episode's mean length and the right figure indicates the episode's mean reward. The increase in reward and length indicates training. 37

4.6 An image of SpaceGuy in gameplay. Three different enemy types are depicted, with the main player in the middle of the screen. The smaller enemies are weaker, move faster, and hit for less impact. The bigger enemies move much slower, deal three damage per hit instead of one, and have significantly more health. The last enemy is the spawner, which spawns little enemies when the player is within its range. The gameplay is terminated when your health runs out, or upon defeating a main boss, which requires finding a key to unlock the door to fight. 38

4.7 SpaceGuy's tensorboard visualization of training curves. The left figure indicates the episode's mean length and the right figure indicates the episode's mean reward. The increase in reward and length indicates training. 40

ACRONYMS

RL	Reinforcement Learning
OpenGL	Open Graphics Library
AI	Artificial Intelligence
NN	Neural Network
ML	Machine Learning
EVR	Electric Vehicle Roadway
EMS	Energy Management System
ITSC	International Conference on Intelligent Transportation Systems
IEEE	Institute of Electrical and Electronics Engineers

CHAPTER 1

INTRODUCTION

This thesis presents the development of Acacia, an extensive 2D game engine equipped with built-in reinforcement learning (RL) capabilities that offer easy extensibility and deployment in and beyond the gaming environment. The core focus of this research is to demonstrate the ability of Acacia to integrate RL algorithms seamlessly into the developed games. Three distinct games were created to demonstrate the engine's capabilities utilizing Acacia's reinforcement learning features.

Game engines constitute various modular components essential for game creation. Some components include physics engines, collision detection systems, audio processors, sprite renderers, font renderers, animation controllers, and particle effect generators. The principle of modularity is critical in the construction of game engines, as it ensures flexibility and minimizes dependencies between different subsystems. Adopting this modular approach, Acacia employs the Entity-Component-System (ECS) architecture throughout its design. ECS was first developed by Looking Glass Studios in their 1998 first-person stealth video game, Thief: The Dark Project. The initial ECS has been slightly modified for many engines, such as Unity, Unreal Engine, and Acacia, to allow for scripting, which is not part of a pure ECS architecture. However, scripting allows for quick prototyping of mechanics and easy extensibility over that of its pure counterpart.

Reinforcement Learning (RL), a subset of machine learning, has garnered significant attention since the emergence of Deep-Q Networks in 2013. [1]. This learning paradigm mimics human decision-making processes, where an agent evaluates actions based on received rewards or penalties. Such a process exists in video game mechanics such as scoring systems, where players are rewarded for achieving specific objectives, impacting their gameplay. RL has applications in diverse domains ranging from games like Go [2] and DOTA 2 [3] to complex real-world tasks like robotics [4–6] and energy management control [7].

Chapter Three of this thesis delves specifically into RL's application in energy management control by presenting a paper published at the IEEE International Conference on Intelligent Transportation Systems (ITSC), showcasing its versatility and potential beyond traditional gaming contexts.

CHAPTER 2
GAME ENGINE

2.1 Entity-Component-System Architecture

The underlying paradigm of this game engine's architecture is the Entity-Component-System (ECS). This paradigm is a versatile data-oriented framework that emulates an object-oriented approach, offering flexibility and modularity. This architecture enables developers to build game entities by attaching specific data components to each one, defining their final behavior, and maintaining clean separation between various aspects of game development.

Consider a scenario where a developer aims to create a dynamic game entity influenced by physics with a designated position on the screen and a unique sprite associated with it. Leveraging the ECS architecture, the developer assembles this entity by attaching specific components, namely the Transform (Position), RigidBody (Physics Enabled), and Sprite components. These components, when combined, encapsulate the entity's characteristics and attributes, such as its position, physical properties, and visual representation.

The advantage of ECS is shown through the cooperation between components and systems. Once the entity is constructed with its components in place, specialized systems, such as the Physics System and Rendering System, take charge of their respective responsibilities. The Physics System ensures precise position updates, while the Rendering System handles the rendering of the entity's sprite at the correct screen location. This modular approach not only simplifies game development but also minimizes interdependence between different systems, promoting a well-organized and scalable codebase.

Furthermore, to augment this robust ECS foundation, a scripting system was integrated. Although not a conventional component of an ECS architecture, the scripting system adds an extra layer of flexibility and ease for game designers. This scripting system

enables the creation of custom behaviors for entities without necessitating the development of dedicated systems for each unique case. It allows designers to inject custom logic, enabling entities to exhibit unique actions, interactions, and responses.

2.2 Graphics

The graphics module within the engine has been designed to offer adaptability, allowing it to seamlessly integrate with a wide range of rendering systems. Its architecture is centered around predefined functions such as Draw, DrawAnimation, DrawString, and DrawInstanced, each of which must be defined as part of the graphics API. Draw defines the functionality of drawing simple and typical sprites, whereas DrawAnimation allows the drawing of layered images, which are used for animations. DrawString allows for the complicated procedure of drawing strings to the screen, which difficulty arises in the different sizes of each character, based on the font that is used. DrawInstanced allows the same sprite to be drawn more than once, specified by an offset. Instance drawing is much more efficient than regular drawing since the data is passed only a single time to the GPU for each batch draw. In particular, the particle system takes advantage of this, as it draws thousands of the same object every frame.

In the current implementation, each of these "draw" functions collects the necessary information and inserts objects into a drawing queue based on a specified drawing layer. This layered approach ensures that objects are rendered in the correct order, providing a visually cohesive output.

Upon completing all drawing calls, a final draw call is executed, orchestrating the rendering of all layers in a synchronized manner. Furthermore, a dynamic layer has been integrated into the system, allowing for in-between-layer drawing. This feature is particularly valuable when characters traverse a scene, dynamically positioning themselves in front of or behind objects depending on their relative positions. This dynamic layer is implemented as a priority queue, sorted by layers, guaranteeing precise rendering between the primary layer renderings as shown in Figure 2.1.

One of the key strengths of this graphics module is its inherent agnosticism toward

the rendering backend engine in use. By defining these core functions, the engine can accommodate rendering engines beyond the default OpenGL implementation. This forward-looking approach enables future developers to create custom graphics libraries tailored to specific needs or to leverage alternative rendering technologies.

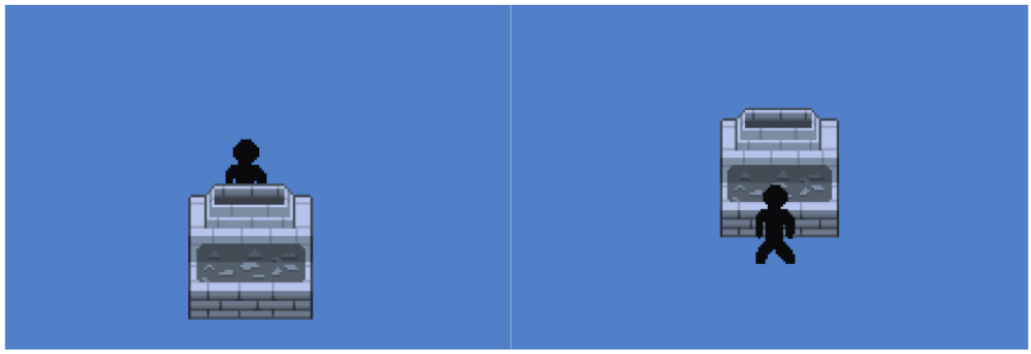

Fig. 2.1: Demonstration of dynamic layers. In this case, the rendering layer cannot be predetermined to render the player in front of some objects and behind other objects based on position. The dynamic layer allows for objects, such as players, to be rendered behind and in front of objects based on criteria a developer creates (In this case, rendering position is based on the player's Y position.)

The graphics module represents a cornerstone of the engine's versatility and extensibility. Its ability to adapt to various rendering systems and its intelligent layer management system ensure that game developers can deliver visually pleasing and immersive experiences easily and precisely.

2.3 Particle System

Rendering particles in games can pose significant efficiency challenges, especially when managing large quantities of particles, ranging from tens to tens of thousands within a single particle effect. For instance, consider the scenario where 100 particles are generated per frame, with initial average render times depicted in Table 2.1

One of the most significant optimizations involved implementing instanced drawing, a

	Debug Mode	Release Mode
Particle Renderer	12,273 us	3409 us
Particle System	95 us	26 us

Table 2.1: Average render times before optimization.

	Debug Mode	Release Mode
Particle Renderer	350 us	190 us
Particle System	95 us	26 us

Table 2.2: Average render times after implementing instanced drawing.

technique that efficiently renders multiple entities with a single render call, thereby optimizing CPU and GPU communication. By utilizing the same vertices for all particles and specifying only the necessary offsets for each instance, a notable performance boost was achieved, as demonstrated in Table 2.2.

Furthermore, to enhance developer accessibility, a variety of generic particle emitter shapes were introduced, including cones, circles, rectangles, and more. These emitters offer options for randomization of start and end speeds, sizes, colors, and alphas, with the Particle System optimizing performance by only updating relevant information when in use. Incorporating these randomizations, including linear interpolation between values, the system was stress-tested with 48,500 particles, yielding the results depicted in Table 2.3.

2.4 Audio System

The audio system was designed as a modular system to allow customizability for the priority of sound effects. Utilizing SDL Audio as the backend, a developer can allocate sound effect buffers upon creation of the game. Each sound effect buffer is given a certain designation- UI, Entity, or music. Per SDL Audio specifications, only a single music buffer is allowed to be playing at one time, but the other sound effect buffers can have up to as

	Debug Mode	Release Mode
Particle Renderer	2666 us	211 us
Particle System	3123 us	674 us

Table 2.3: Average rendering and updating times for 48,500 particles.

many channels as a device can support. This sound effect buffer designation system allows a developer to assign priority to certain sound effects based upon the quantity of buffers designated to the category. For example, if an audio device can handle 256 channels and the game has many entities playing sounds at one time, the developer may want to have a guaranteed sound effect for important UI alerts or effects that convey meaning. This flexibility gives developers more choice on priorities of sound effects, which may vary from game to game.

2.5 Input System

The custom input system serves as a pillar for versatile control across various platforms, including mouse and keyboard, controllers, and AI input systems. Its architecture is designed to allow developers to map controls to specific in-game functions, promoting accessibility and adaptability.

At its core, the system operates through the utilization of lambda functions defined by the developer. These functions serve as the bridge connecting player actions with corresponding in-game actions. For instance, in one of the developed games demonstrated in subsection 4.5.2, Crypt, players can flip gravity, enabling traversal on both the ground and the ceiling. To achieve this, the developer defines a "flipGravity" function, which acts as a central hub for all inputs associated with this mechanic.

To further enhance flexibility, a separate structure is employed to map control inputs to their respective functions. This decouples functionality from direct key bindings, allowing for effortless input binding and remapping. It ensures that players can tailor their control schemes to their preferences without being confined to predefined keybindings.

The true power of this input system emerges from its agnostic nature, accommodating various input devices seamlessly. Whether players opt for traditional mouse and keyboard setups, use diverse controllers with varying layouts, or when the game is controlled by AI-driven input, the system remains adaptable and efficient.

2.6 Collision Detection

Collision detection is a well-studied issue in game engine design, robotics, CAD software, and various other fields. [8–12] It plays a crucial role in ensuring the realism and interactivity of virtual worlds. As the complexity and quantity of objects increase, the computational cost of collision detection per frame also increases significantly. To address this, numerous algorithms have been developed to reduce the number of objects checked for collisions per frame. Some of these algorithms include KD-Trees, Uniform Grids, Spatial Hash Maps, and Quadtrees in 2D, with Octrees being a common choice in 3D environments. Each of these algorithms presents distinct advantages and disadvantages, depending on the specific requirements of the application, but they all offer substantial improvements over the brute force $O(n^2)$ method.

2.6.1 Uniform Grids

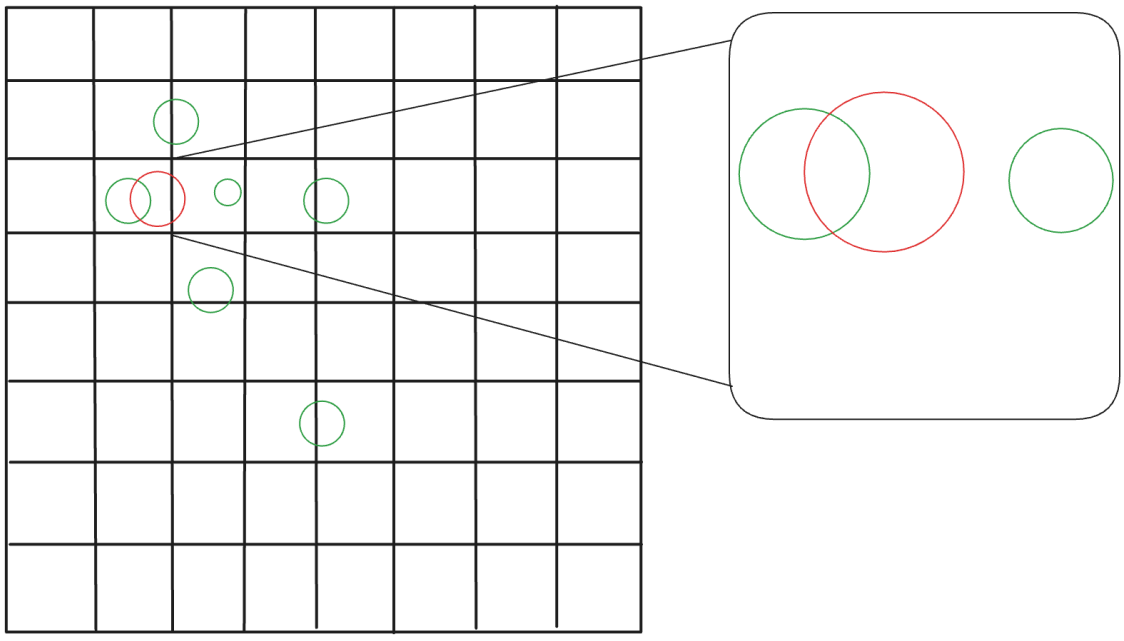

Fig. 2.2: A representation of a uniform grid. The left side represents the world partition and the right side depicts the grid locations checked for potential collisions when examining the red circle's position. This reduces the potential collision count from 6 to 3, leading to computational savings.

Uniform grids are one of the most simple yet effective collision detection algorithms in 2D. In this approach, the world space is partitioned into a grid of uniform cells, with the cell size configurable by the developer. Each entity in the world is placed into the grid cell corresponding to its position. During collision detection, the system only needs to look up the grid cells that intersect with the object's location, significantly reducing the number of potential collisions to check (see Figure 2.2).

Uniform grids excel in scenarios with rapid object movement or evenly distributed objects. However, they may struggle when objects are densely clustered, as it can result in situations where the $O(n^2)$ worst-case scenario is approached.

In Acacia, uniform grids are utilized for 2D games due to their quick construction time and flexibility. An optional Quadtree implementation is also available, allowing developers to choose the collision detection method that best suits their project's needs.

2.6.2 Spatial Partitioning Trees

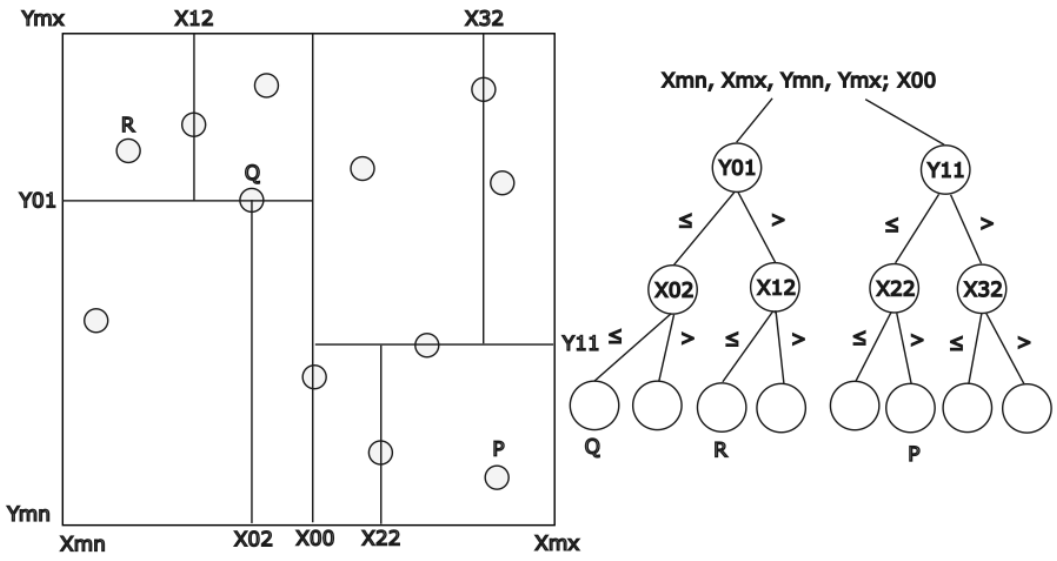

Fig. 2.3: A representation of a KD-Tree. The left side represents the world partition and the right side is the data structure that is formed. This splits the world depending on the entities found in every frame.

Spatial Partitioning Trees, such as KD-Trees, divide the world space into regions and recursively subdivide them into smaller regions. KD-Trees, for instance, split the space along hyperplanes, creating a hierarchical structure (see Figure 2.3). This hierarchical organization facilitates efficient collision checks, as the KD-Tree only needs to inspect objects within its partition and neighboring partitions. Additional research has been done to improve computational efficiency and other speedups. [13–18]

2.6.3 Quadtrees

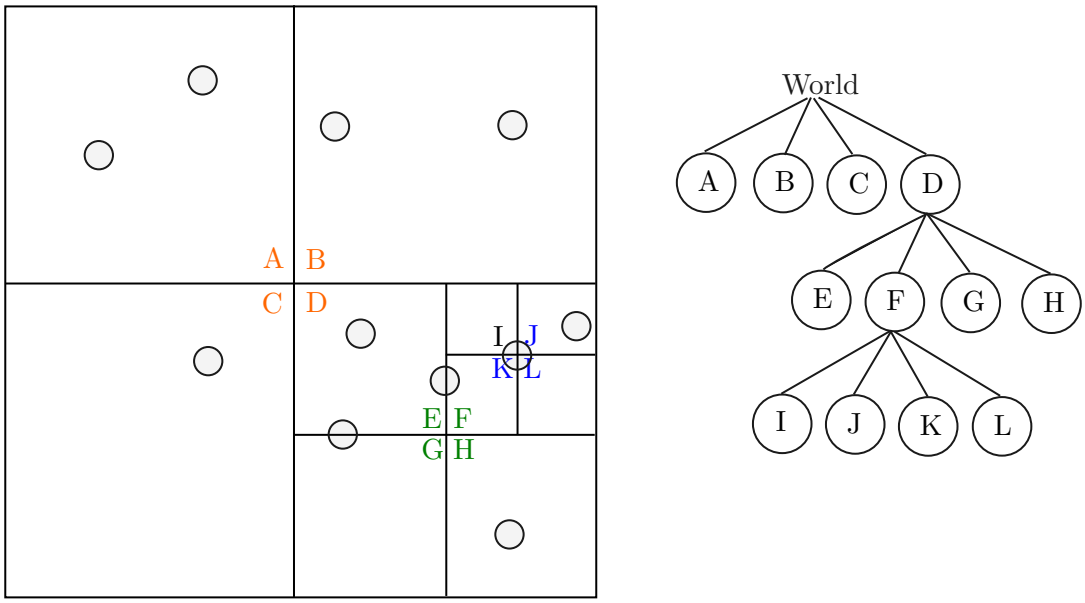

Fig. 2.4: A representation of a quadtree. The left side represents the world partition and the right side is the data structure that is formed. This allows quick traversal to find potential neighbors with which to compare.

Quadtrees, similar to KD-Trees, break the world space into partitions. However, as indicated by the name, it breaks the world into four parts, rather than the two half-spaces that the KD-Tree does. This allows for areas that contain more entities to be much more granular while leaving the overall world partitions where few entities exist simple. A representation of this is found in Figure 2.4.

Quadtrees offer efficient collision detection with an insertion time of $O(log(N))$, where

N represents the number of nodes being inserted into the tree. Building a complete Quadtree takes $O(N * log(N))$ time. Collision detection based on the Quadtree then requires $O(N * log(N))$ for construction and searching each frame. It is important to note that the worst-case scenario of a quadtree is $O(N^2)$. However, this is extremely rare and is typically not a concern in practice.

Quadtrees are well-suited for managing data in sparse data scenarios. Acacia's implementation includes both dynamic and static Quadtrees, optimizing collision detection for moving and static game objects, respectively.

Collision detection is a critical aspect of game engine development, and choosing the right algorithm depends on the specific requirements of the game and its environment. Whether opting for uniform grids, spatial partitioning trees like KD-Trees, Quadtrees, or other algorithms not mentioned here, each approach offers advantages and trade-offs that should be carefully considered.

2.7 Value of Acacia

The goal of developing Acacia is to provide an extensible 2D engine to improve development in RL by enabling easy integration with PyTorch and TensorFlow, and the myriad other support libraries within the Python ML ecosystem. Building the Acacia engine offers distinct advantages over utilizing Unity RL, particularly in the realm of RL integration. While Unity provides a robust environment for game development and includes its Unity-ML solution for RL, several limitations hinder its effectiveness in certain contexts.

One significant drawback of Unity RL is its limited flexibility when it comes to exporting models outside of the Unity environment. This constraint poses challenges for scenarios where deploying trained models into real-world applications is necessary. Unlike Acacia, which prioritizes extensibility and integration with popular deep learning frameworks like PyTorch and TensorFlow, Unity RL's ecosystem can be restrictive in terms of interoperability.

Moreover, Unity's primary focus lies in 3D game development, and while it does support 2D games and simulations, its features are not as tailored or optimized for this specific

domain compared to Acacia. This lack of specialization can lead to inefficiencies and complexities when implementing 2D simulations, especially when coupled with RL algorithms.

Furthermore, Unity's learning curve can be steep, particularly for developers primarily focused on RL implementation. The complexities of the Unity environment, coupled with the intricacies of RL algorithms, can pose significant challenges for developers aiming to leverage RL in their Unity projects.

In contrast, Acacia is purpose-built with a focus on simplicity, extensibility, and compatibility with RL methodologies. By providing a dedicated 2D engine that seamlessly integrates with popular deep learning frameworks, Acacia empowers developers to create and deploy 2D simulations with RL capabilities more efficiently and effectively.

2.8 Reinforcement Learning Use Cases

A significant application of our work in RL was published at the International Conference on IEEE Intelligent Transportation Systems (ITS) Conference, the premier conference in ITS. This research addresses a critical challenge in the realm of sustainable transportation: optimizing electric vehicle (EV) charging infrastructure to mitigate month-to-month charging costs. A Python simulation (which laid some groundwork for this project) was developed to simulate and assess charging limits imposed on EV charging equipment. RL was used to devise strategies that minimized charging expenses for consumers while improving the utilization of charging resources, promoting efficiency and sustainability in EV adoption.

However, the existing simulation framework presents notable limitations, particularly in its visualization capabilities, spatial awareness of vehicle parking, and integration with more reinforcement learning (RL) methodologies. These shortcomings hinder the simulation's ability to fully leverage the power of RL algorithms, which excel in learning optimal decision-making strategies in complex, dynamic environments. Acacia can provide a solution that enhances the simulation's effectiveness in modeling real-world scenarios accurately. Improving simulation capabilities to adapt and learn from experience dynamically can significantly improve the reduction of charging costs.

Acacia is further showcased by three new examples that leverage integrated reinforcement learning. Using the Gymnasium API, these examples demonstrate how developers can standardize environments to behave like common gaming environments, facilitating seamless integration with RL algorithms. This highlights Acacia's adaptability and empowers developers to harness the potential of RL across diverse applications, from gaming simulations to real-world problem-solving in transportation systems.

This research addresses the specific challenges within EV charging infrastructure optimization and exemplifies the broader potential of reinforcement learning in transforming and optimizing complex systems. By bridging the gap between theoretical advancements in RL and practical applications through Acacia, this work contributes to the advancement of many fields in diverse domains.

CHAPTER 3

Save Money, Get Charged: Facility-Tied Energy Management with Unknown and Unscheduled EV Charging

Energy Management Systems (EMS) are crucial for maintaining economically viable and consistent energy usage, particularly for facilities with flexible power loads. With the increase in the availability of electric vehicles (EVs), EMS must balance internal building usage with Electric Vehicle Supply Equipment (EVSE) charge management. Many current EMS balancing of EVSE loads is done through a simple interface (often provided by the manufacturer of the EVSE units) which simply caps the maximum allowed charge rate (curtailment). Loads from EVSEs are generally not anticipated by the EMS (as EV operators at facility-tied EVSE units typically do not schedule a charger), with a power tolerance used to justify a safe curtailment.

Our work details the development of an EVSE management-enabled EMS that changes curtailment throughout the day in anticipation of EV users that may charge their vehicles. This EMS module utilizes the Open Charge Point Protocol (OCPP) protocol and reduces operating costs while ensuring EV operators have a full charge within their charging window. While the addition of an EVSE control module increases the complexity of the EMS, cost spikes in facility operation from short-term, large-power EV charging events (which can occur at any time) are largely mitigated.

The EVSE-management EMS module is demonstrated in the Electric Vehicle and Roadway Research (EVR) facility with the primary objective of reducing energy costs incurred by its significant power needs. The pricing mechanism for the EVR power is based on the peak energy usage observed by the utility provider in the current billing month. Tests consider two DC fast-charge EVSE units (ABB Terra HP Chargers), capable of delivering 350KWh, with many electric vehicles capable of drawing a load ranging from 5-300 kW. This short-duration and high-power event is concerning to the facility (particularly during

peak load times, Figure 3.1) as it can more than triple the monthly power costs in a final bill.

Due to EVSE hardware and communication capabilities, this EMS operates under the assumption that EVSE units may only be curtailed before a charge event is initialized. Once a vehicle is connected to the EVSE, curtailment cannot be modified for the duration of that charge session. The inability to change the delivered power in session requires the EMS to predict power needs and constantly alter curtailments, as EV charge events are not known in advance. The EMS can respond to emergencies or significant energy loads by terminating power delivery to EVSE units during a charge event. This emergency stop is implemented but has not been used.

The original EVR EMS logic and communication lacked the ability to respond to EVSE-related power events. This research details the methods of extending the EMS capabilities to continuously forecast overall energy usage, which informs a predictive agent that sets curtailment limits on all facility-connected EVSE units. This EVSE-predictive EMS module has alleviated significant power events from exceeding peak draws while continuing to allow EV operators using the EVSE units to have a full charge upon vehicle use, reducing the ancillary costs of facility power.

3.1 Foundational Algorithms

3.1.1 Facility Associated Energy Management Systems

Energy management systems are critical for many day-to-day operations and often focus on the reduction of costs while ensuring power is available to all services. Facility EMS, in particular, is flexible in responding to smart grids as demonstrated in cases of industrial and academic use [19] and in intelligent machine-learning-driven peak-shaving [20, 21]. Facility EMS is a critical infrastructure for initiatives toward zero and nearly zero-energy buildings, because facility usage is a significant source of emissions [22] and is often managing the point of charge for many EV operators [23, 24].

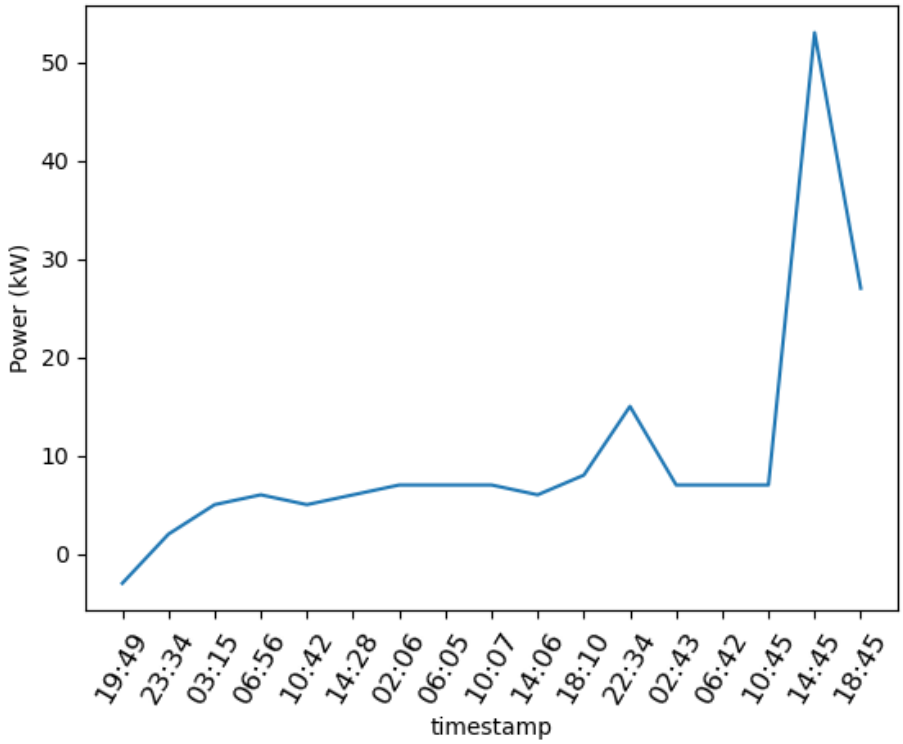

Fig. 3.1: Example of a peak load event at the Electric Vehicle and Roadway Research Facility (EVR). Utility charges for the facility power are based on the maximum recorded power draw during the 1-month billing period. During peak times, uncontrolled EV charging events can easily add significant costs. The EMS EVSE module is designed to curtail efficiently, predict, and respond to these high-cost events while allowing vehicles to charge without interruption.

EVs fundamentally regulate internal power needs through an onboard Battery Management System (BMS), with some BMS able to communicate to remote devices [25, 26] via cloud interface, giving additional data feeds to a facility EMS. Current research explores many potential avenues for peak-shaving with considerations of associated EVSE units [27, 28], however, many of these lack a facility implementation or require Vehicle to Grid (V2G) communication. The EVSE EMS module developed does not require V2G and is reliably demonstrated in the USU Electric Vehicle Roadway (EVR) facility through integration into the facility EMS.

The EVR is designed for the development of technology for dynamic in-road charging, MegaWatt wireless, and fast inductive charging. With the presence of these high-power

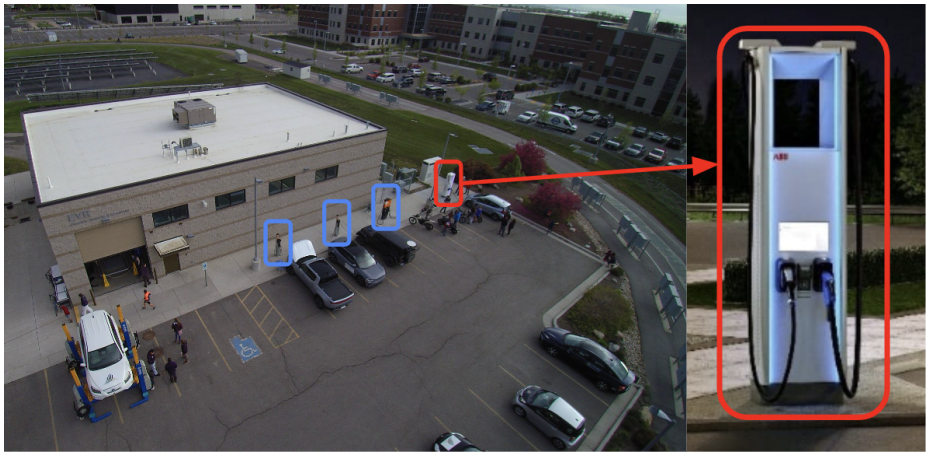

Fig. 3.2: Electric Vehicle and Roadway Research Facility (EVR) and the ABB DC-Fast Charge EVSE units. The EVSE EMS module was developed based on the needs of the EVR facility. This tool was successfully deployed on hardware and integrated with the existing EMS services. Some data collection and logging of full-facility energy usage, charging data, and solar information were conducted as part of the development.

events regularly occurring, the EVR routinely draws significant grid power. These draws must balance the associated solar production and battery storage reserves, taxing the capabilities of a complex facility EMS. Without V2G communication or a schedule of EVSE use, the six EVSE units installed at the EVR can burden the EMS tremendously, particularly the two DC Fast-Charge EVSE units. The EVR EMS must consider the possibility of an EV charge event suddenly occurring, and the magnitude and duration of this event. A learning approach using transformers is employed to correctly and efficiently ensure power needs do not cause peak infringement.

3.1.2 Transformers

Transformers are a sequence-to-sequence machine learning algorithm and have recently garnered much attention for their performance compared to other architectures in various domains, such as language modeling and general sequence prediction [29] [30] [31]. Transformer architectures include some of the most advanced AI toolsets, including ChatGPT [32] and BERT [33], both excellent language models. Transformers learn through masking information in a given sequence and predicting the masked value, which is excellent for predicting

energy usage. This masking then shifts, and it continues to predict one more value until it hits an End of Sentence token (EOS token), which indicates that the model can stop predicting. A second loss function is added to concurrently predict the ordering of the predicted value, which allows it to sequence correctly. Without this, it would become an assortment of unordered, but relatively correct, predictions. The general architecture can be seen in figure 3.3.

This kind of sequence-to-sequence generation is powerful and has been shown to produce more accurate results than LSTM architectures for sequence generation in speech applications [34] as well as train substantially faster [30], making it the AI model of choice. Many current approaches in energy prediction for EMS still rely on LSTM and similar architectures [35, 36]. In the energy prediction implementation of this research, energy typically follows a sporadic and seemingly sinusoidal wave, with the largest peaks occurring during the day. However, these peaks are seemingly stochastic in occurrence, preventing simple learning methods from fitting the energy usage. Thus, a transformer is well-suited to create a more accurate energy usage model.

3.1.3 Bayesian Updating

To accurately predict the probable time required for a charge, it is necessary to predict the starting SOC for the vehicle batteries. Determining the state of charge (SoC) of a vehicle that could potentially connect is conducted through a Naive Bayes model. Bayesian updating is a simple but powerful idea that relies on previous experience to build out a probability distribution and is widely used in EV battery technology for predictions of health, degradation, and charge profiles [37–39]. The Naive Bayes model is defined simply as:

$$P(H|D) = \frac{P(H)P(D|H)}{P(D)}$$

where $P(H)$ is our prior (a likely distribution of SoC), $P(D|H)$ is the probability of an SoC given the likely distribution of SoCs, and $P(D)$ is the probability of a vehicle of a given SoC. Bayesian Updating allows us to keep collecting data and modifying our posterior,

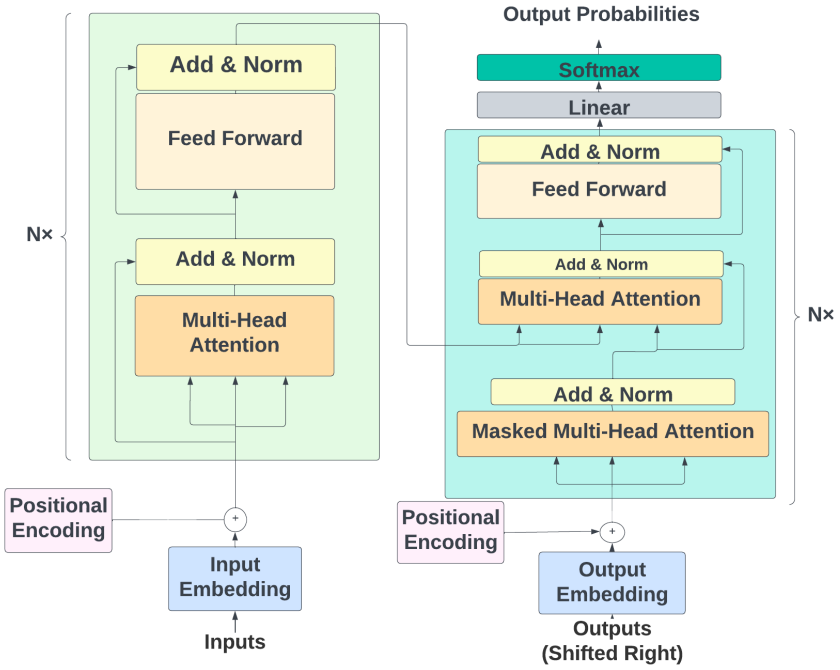

Fig. 3.3: Structure of the transformer architecture used in the EVSE EMS module development. Transformers leverage multi-head attention to parse historical sequence information allowing learning to be more resilient against "forgetting" as it generates new sequences of information. This provides an advantage in EMS applications over similar tools, such as LSTMs, for sequence generation.

$P(H|D)$ while retaining our old data to inform our updates. Bayesian methods allow for value prediction without needing the vast quantity of data that a typical machine-learning algorithm requires. As new data is collected, Bayesian methods can also continually update at a cheaper computational cost.

3.2 EVSE Aware EMS Module Implementation

3.2.1 Energy Usage Prediction

We implemented a transformer for sequence generation of energy usage based on the previous seven hours of energy usage history. Seven hours was chosen as an optimal time window based on a grid-based parameter search. The seven hours of operating data are binned and averaged into segments of ten minutes, associated with the billing time cycle of

the utility provider. The transformer outputs the energy usage forecast for the next three hours at ten-minute intervals. This is compared with observed data for the subsequent three hours, as seen in figure 3.4.

The transformer employed has eight encoder and decoder layers, eight attention heads, a sequence length of 40 values (corresponding to the seven hours of input data), a learning rate of 0.001, and utilized an SGD optimizer. A grid search was performed over all the parameters mentioned above to tune the transformer hyperparameters. The resulting energy usage predictor plays a vital role in the module as it serves as our energy forecast estimate to inform our decisions and act as curtailment safeguards.

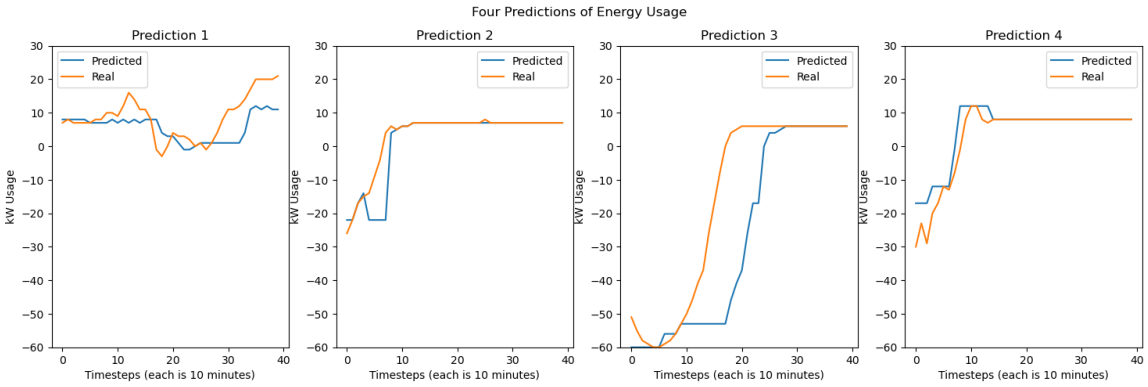

Fig. 3.4: Comparisons of energy prediction cases. Four energy usage predictions are shown, contrasting the real usage against transformer predictions over a three-hour window.

3.2.2 Battery State of Charge Prediction

The EVSE EMS module assumes a vehicle may plug in during the immediate following timestep and the module estimates the SoC of that vehicle based on three general distributions where data could be drawn from: mostly charged, partly charged, and barely charged. While prior researchers tend to show EV charging into two distinct categories (20%-80%, 80%-100%) [40], we chose three classifications to be defined by charges at $\leq 95\%$, $\leq 75\%$, and $\leq 50\%$, respectively as this was more in line with observed data from charge events at the EVR facility. Each of these bins was then represented by its own distribution for which

the likelihoods are then computed and used to normalize the distributions.

For the initial generation of this non-standard distribution, we used data collected over the past ten months of charging data at the EVR along with the data on the cars that have charged.

Battery SoC likelihood is characterized by one of the three possible sub-distributions. From facility data, we see that the primary SoC profile of vehicles when connecting is heavily skewed towards being mostly charged. This is likely due to the workplace nature of the facility and the relatively small commutes experienced by many EV operators using the EVSE units. The separation of historical data into three distributions was able to better fit and explain the charging SoC profiles (Figure 3.5).

Fig. 3.5: Estimations of SoC prior to a charge event starting. 1,000,000 random samples were generated from three distributions representing three classes of people operating their EVs at different SoCs before charging.

To verify our findings, both the original data and a modeled distribution formed from 1,000,000 samples were graphed. As seen in figure 3.5, these results show that the non-standard distribution describes the original distribution well. This enables us to pull from the non-standard distribution in real-time to predict what battery SOC any EV may be at and effectively update and adapt to new information from future vehicles.

3.2.3 Car Prediction

Estimation of the total battery capacity of incoming vehicles is accomplished by visual identification to determine the make/model of the car and expected battery by a lookup table. This visual identification model is practically implemented through security camera footage. (We note that the security camera footage does not show if a vehicle is or will connect to a charger, only that it enters the parking area.) If a vehicle is unable to be identified as an EV, an average battery capacity of 86kWh is assumed based on observed vehicle models at the facility.

Battery capacity is estimated through a car identification model built using two image datasets. The first dataset is a larger, general dataset of cars, VMMRdb [41]. The other is a smaller hand-created dataset, which compiled several images of newer electric vehicles and their battery information, as VMMRdb was released in 2016 and lacked many models of electric vehicles. This smaller hand-created dataset allows the model to classify newer electric vehicles in a "new EV" category, denoting vehicles of battery capacity averaging 86kWh. The intention is to allow the model to collect "new EV" images for hand labeling to improve the quality of the small newer electric vehicle dataset in the future while allowing the VMMRdb dataset to be more versatile for our use.

The vehicle identification model was trained on both datasets to recognize cars using transfer learning using the Regnet [42] model. Training on this combined custom dataset yielded a model with a 90% identification accuracy, with an F1 score of about 0.9, as seen in figure 3.6. This level of accuracy is suitable for the EMS to leverage this tool in use for battery capacity prediction.

3.2.4 Battery Charging Simulation + Curtailment Setting

After the battery SOC, vehicle model, battery capacity, and energy forecast is generated, all derived information is given to a DeepQ reinforcement learning agent (DQRL) that determines the final curtailment. The agent's reward structure is designed to receive a -1 reward for every timestep of 10 minutes that the car is charging and a -100 for any infringement of the current peak. The intention behind this is to incentivize the agent to charge as

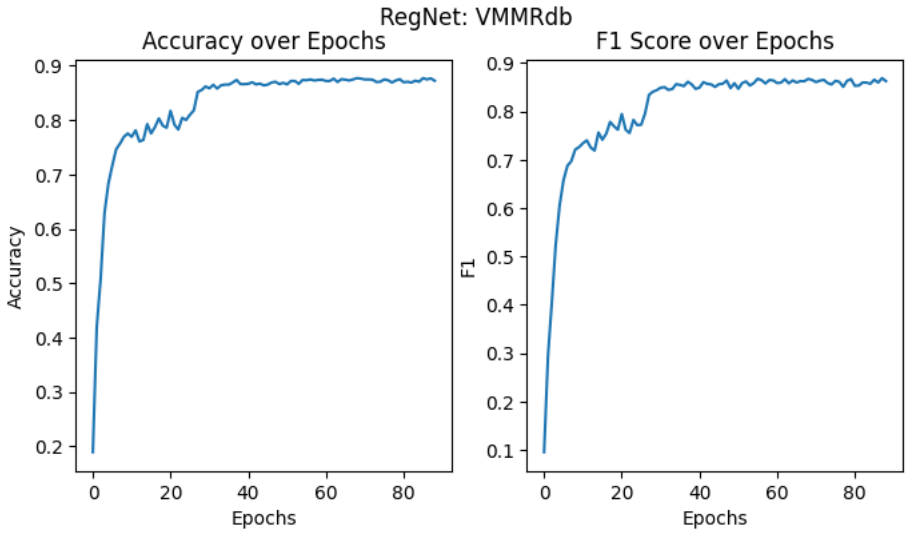

Fig. 3.6: Training performance of the EV detection and identification model. Accuracy and F1 are shown over the 80 training epochs, attaining 90% accuracy on validation data and a test accuracy of 87%.

quickly as possible for increased EV operator satisfaction while preventing the peak from being exceeded. The DeepQ algorithm was preferred over a simpler Monte Carlo algorithm as the DQRL algorithm can also serve as an error adjuster, making it more resilient to unpredictable price shifts. As the transformer will not perfectly predict the 3-hour forecast, the DQRL agent began to learn the discrepancy between the forecast and the amount that it should curtail.

3.3 Software Implementation

Next, curtailment on the actual EVSE was necessary in order to employ these algorithms. A standard protocol was utilized, called Open Charge Point Protocol (OCPP). OCPP defines a generic standard allowing easy integration into various chargers with different functionalities. One such functionality is the ability to curtail the chargers. However, as previously mentioned, in-charge curtailments are not currently possible on the ABB EVSE equipment at the EVR. As such, we used "Overall Curtailments," which curtails the power available for the charger prior to a charge session commencing. Further, we can specify more granular details, such as which side of a charger should be curtailed (if there is more

```
charger_profile_data = {'connectorId': 0,
'chargingProfileId': 0,
'stackLevel': 0,
'chargingProfilePurpose': 'TxDefaultProfile',
'chargingProfileKind': 'Absolute',
'chargingSchedule': {
    'duration': 60000,
    'startSchedule': datetime.utcnow(),
    'chargingRateUnit': "W",
    'chargingSchedulePeriod': [{
        'startPeriod': 0,
        'limit': 20000}]}}
```

Fig. 3.7: OCPP command that is then sent to a REST API which records all curtailments. The OCPP websocket queries the REST server every few seconds to look for commands it should apply. This command would set the curtailment to 20 kW and have a duration of 60,000 seconds.

than one port per charger), curtailment durations, and start times (see figure 3.7).

As seen in figure 3.8, the curtailments on the chargers are correctly functioning and curtailing to specified amounts.

3.4 Simulation Experiments

We test the capabilities in a simulation sandbox server where it is possible to know all details of a charging EV. To introduce uncertainty in the simulation (to better reflect reality), battery capacity prediction from the vehicle identification model, is set to underperform using an accuracy of 80%, reflective of fielded implementation. If the predictor failed to identify the charging vehicle correctly, incorrect information about battery capacities would be used in the DQRL. Battery SOC prediction leveraged the Naive Bayesian updater based on historical data to populate random SoCs. The simulated battery capacity and SoC were combined with the predicted energy usage (transformer model) and the DQRL to set the curtailment which would reward fast EV charging while attempting to honor peak constraints. Facility energy usage was taken directly from historical energy usage data at the EVR. This was to enable the simulation to test peak infringements and subsequent costs (in USD) based on real billing data.

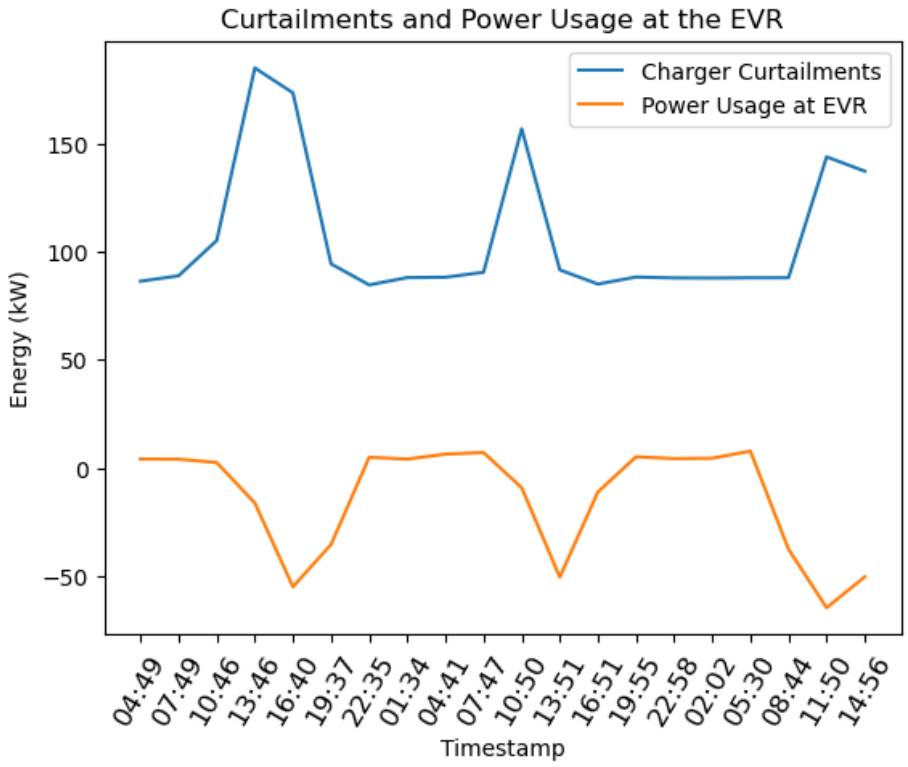

Fig. 3.8: Energy usage at the EVR and curtailments on the DC-Fast Charge units installed at the EVR facility. As solar generation provides negative power during peak solar, available power to chargers increases, raising the curtailment (blue).

The EV charging profile during the charge session was assumed to be a generally linear battery charging curve, with a 90% efficiency. Lithium-Ion batteries typically operate at 99% charge efficiency [43], the simulation assumes a lower battery efficiency due to potential degradation, thermal effects, and other possible unknown factors to build a more conservative estimator. The simulation ran in increments of 10-minute timesteps, with reports generated when a peak was exceeded during a charge session.

3.5 Results

Vehicle charging simulations were run 10,000 times with the average charging time required, the number of times that the charge exceeded the given peak, and the average peak infringement recorded. Peaks were assigned to values between 60-160 kW for each iteration. The EVSE EMS module was compared with a classical static curtailment of

Table 3.1: Results from a randomly generated peak from 60-160kW of 10,000 simulations

Curtailment	Average Time	% Exceeding Peak	Average Amount Exceed Peak
Static: 50 kW	28.81 minutes	0%	0kW
Static: 100 kW	16.85 minutes	36.3%	61.646 kW
Static: 150 kW	13.54 minutes	100%	200.828 kW
Static: 200 kW	12.69 minutes	100%	251.285 kW
EVR EMS	21.40 minutes	7.4%	6.29 kW

low (50kW), medium (100kW), high (150kW), and extremely high curtailments (200kW) as a comparison metric. The results of two separate tests (with and without full vehicle knowledge) can be seen in table 3.1.

Peak infringements increase as the static curtailment limits are raised, as expected. The EVSE EMS module is able to increase the charge completion time by 26% (uncertain vehicle) or 38% (known vehicle). The peak power is exceeded in 7.4% (uncertain vehicle) and 12.3% (known vehicle) of the 10,000 simulated charge sessions. The module's average amount of exceeded power remains far higher, which has uncertain vehicle identification with a 6.29 kW average peak infringement compared to 0.953 kW seen in the known vehicle model. The difference between correctly identifying the vehicle translates to a $66.52 difference in price.

Benchmark implementation of a simple 50 kW static and permanent curtailment is able to keep operating costs at the EVR to $345.60 on months with lower use and high solar generation. Without any curtailment or control strategies implemented, the costs of operation at the EVR can rise in similar months to as much as $834.72, with peak loads hitting 67kW. Our EVSE EMS module does add $74.75 to the operating bill but ensures that multiple vehicles are able to charge, which the simple 50kW static curtailment was unable to accomplish.

3.6 Conclusions and Future Work

The EVR EMS module is shown to reduce the charging time of EVs by 24-38% compared with a base 50kW static curtailment, while dramatically reducing the peak infringe-

Table 3.2: Results from a randomly generated peak from 60-160kW of 10,000 simulations with known car information

Curtailment	Average Time	% Exceeding Peak	Average Amount Exceed Peak
Static: 50 kW	28.79 minutes	0%	0kW
Static: 100 kW	17.09 minutes	36.8%	6.989kW
Static: 150 kW	13.57 minutes	100%	59.154kW
Static: 200 kW	12.8 minutes	100%	119.995kW
EVSE EMS	17.92 minutes	12.3%	0.953kW

ment compared to a similar charge-time static curtailment of 100kW. If incoming vehicles are correctly identified by security camera footage upon entering the facility parking, the average peak infringement is only 0.953kW, equating to a $11.84 increase in the monthly bill. However, unlike baseline cases, all EVs were able to receive a full charge prior to vehicle use.

We note that the peak limits, data for training, and implementation software are specifically targeted for the USU EVR facility. The developed code and methods are sufficiently portable to other EMS systems integrated with an EVSE to be valuable in other system integrations, available at https://github.com/DIRECTLab/Predictive-EMS. Ongoing tasks will add several lower power EVSE units of different makes and models to demonstrate the robustness of the EVR EMS module to heterogeneous communication and control frameworks.

Additional work in creating a more capable EVR EMS module is ongoing. Future improvements include the refinement of the vehicle detection model, especially for newer electric vehicles due to their increased battery and charge capacities. Our car detection and segmentation model will naturally improve unknown cars that are classified manually. Increased image sets will be leveraged to train improved vision models to better recognize unknown cars. Additionally, sufficient data on specific vehicles are beginning to allow battery charge profile predictions. This will enable a more precise understanding of EV power needs and charge times, further reducing the average peak exceeded power.

3.7 Acknowledgements

This material is based upon work supported by the National Science Foundation ASPIRE Engineering Research Center under 1941524.

CHAPTER 4

REINFORCEMENT LEARNING

As demonstrated, reinforcement learning's capacity for optimization and capability to decrease charging costs in electric vehicle charging is relevant and advantageous in various domains. Thus, reinforcement learning support was developed and integrated into Acacia to allow for flexible visualization and simulation capabilities with powerful RL functionality.

4.1 Deep Q Networks (DQN)

Deep Q Networks (DQNs) were the first major development in deep reinforcement learning, created in 2013. [1] Unlike Q-Tables, which learn to map a position in the world to a certain action, it replaces this "Q-Table" with a network as the function approximation instead.

DQNs are suitable for tackling problems with discrete spaces. One of the most well-known RL environments where DQNs excel is CartPole. For an example shown on CartPole, a video has been provided at this link https://youtu.be/i-9bcXxm4zo. In these scenarios, the agent learns to map an observation to a certain action, leading to an optimal or poor outcome. Because the DQN algorithm is an offline policy, it then uses these observations, rewards, and next-state observations, to build a memory bank or experience replay. Using this experience replay, it learns to associate good and bad actions with the state observation the agent is in. Using these associations, it then chooses an action to solve the problem. After building up the experience replay, the algorithm uses its experiences to update the Q-Table as shown in equation 4.1.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)] \tag{4.1}$$

After updating the Q-Table, it then uses equation 4.2 to update the neural network weights to minimize the loss between the target and $Q(s, a)$. [44] [45]

$$L_i(\theta_i) = \mathbb{E}_{a\sim\mu}[(y_i - Q(s, a; \theta_i))^2] \tag{4.2}$$

where

$$y_i = \mathbb{E}_{a'\sim\pi}[r + \gamma max_{a'} Q(s', a'; \theta_{i-1}|S_t = s, A_t = a] \tag{4.3}$$

One characteristic of DQNs is the algorithm is greedy. They will always take the best action for the state that they are in. This, however, presents an issue when the algorithm is first starting to train. Since the algorithm does not know the pairing of the state of the world and the rewards, the greedy algorithm would not know which action is truly the best to take and will take the same action no matter what. To mitigate this, an epsilon is used, which controls how often the agent takes a random action instead of what it considers to be the best action possible. This allows it to create this memory of what actions truly lead to the best action, given the state.

The major downside of this epsilon, however, is if the agent happens to not encounter a state while it is still exploring, it will still run into the memory issue. The agent will then act as if it knew the best action, without the knowledge of what the best actions are. This problem, among others, is what the following algorithm aims to improve on.

4.2 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) works differently than a DQN by acting in probabilities to take certain actions, given a state. [46] PPO was preceded by Trust Region Policy Optimization (TRPO) which was significantly more complicated than DQN but guaranteed monotonic improvement with little tuning of hyperparameters. [47] PPO, which built on TRPO and simplified its loss function, relied on the basic idea that the algorithm has a policy, and based on the reward signal, you calculate the loss based on a gradient. The loss functions are shown in equation 4.4

$$L^{CLIP+H+V} = L^{CLIP} + hH - vL^V \tag{4.4}$$

where

$$L_t^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(p_t(\theta)A_t, clip(p_t(\theta), 1-\epsilon, 1+\epsilon)A_t)] \tag{4.5}$$

where

$$p_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{4.6}$$

and

$$L^V = \hat{\mathbb{E}}_t[(V_\omega(s_t) - V_t^{target})^2] \tag{4.7}$$

where h and v are weighting factors for H (the Entropy bonus) and L^V, and ϵ is defined as a small hyperparameter that controls how large of an update the policy is allowed to receive per update. [48]

These loss functions remove the incentive to update the policy too much in a single iteration, which can be an issue in online learning. If a large return is returned for an action in a certain state, the algorithm is not guaranteed that the next time the same reward will be given. As an example, suppose the agent is a goalie in a soccer match. If the goalie dives to the left given the state of all the players, it may block the goal one time, but the next time the ball might go to the right instead. The algorithm, then, should not assume that the reward will always be the same, which helps with the stability of the algorithm. Furthermore, we want to take the lower bound of what we know is possible for the amount of modifying the policy. If we are conservative with updates, we avoid a higher risk of instability in the algorithm.

4.3 Python Scripting

To facilitate framework agnosticism, Python was integrated as a scripting language for Acacia. PyTorch has an API for using its framework directly from C++, however, Tensorflow has less support, as well as potential new frameworks in the future that may or may not have C++ integration.

For each game, there is upfront work to pass the state, rewards, and actions back and

forth between the Python interpreter and Acacia. This requires the developer to define the classes that will make up these three elements, which then allows the Python interpreter to embed these classes into itself and use them. This also allows Python to pass back an action object that the engine can then interpret and use.

4.4 Parallel Instances of an Environment

In the realm of reinforcement learning, the agent undergoes a learning process akin to that of a human player, gradually refining its strategies through trial and error within the game environment. However, as the agent becomes more proficient, its progress can slow due to infrequent failures. To expedite this learning curve, a technique known as parallelized training is employed. This involves creating multiple instances of the game environment, each running independently. By decoupling rendering from the training process, significant resources are freed up which allows for more instantiations of numerous environments concurrently. Consequently, the agent can undergo training across a multitude of scenarios simultaneously, accelerating the learning process. What might have taken weeks or months to train in a single environment can now be accomplished in a matter of seconds, enabling agents to rapidly evolve their skills and strategies. This parallelized training approach not only optimizes training time but also enhances the agent's adaptability and robustness by exposing it to a diverse range of scenarios and challenges.

4.5 Developed Games

4.5.1 Brick Breaker

Brick Breaker is a classic arcade-style game that challenges players to control a paddle and bounce a ball to achieve the highest score possible. The objective is to break the bricks arranged at the top of the screen using the bouncing ball. While the traditional version of the game features multiple levels, the example discussed here was intentionally designed as a single level to showcase the engine's capabilities and its compatibility with reinforcement learning techniques.

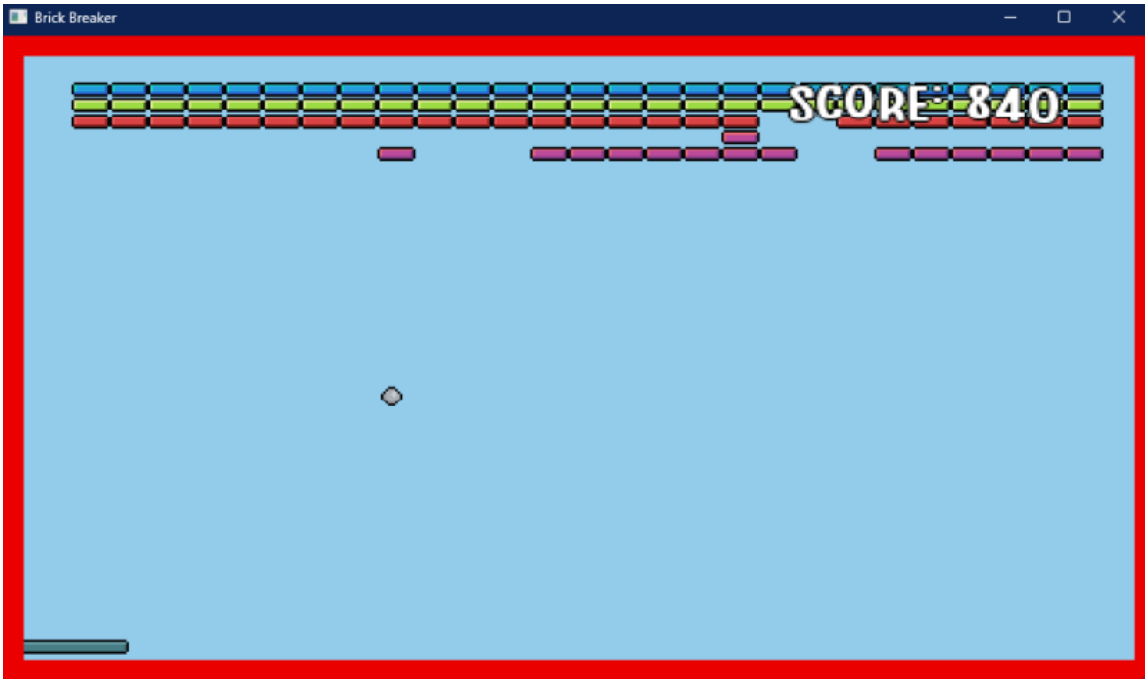

Fig. 4.1: An image of the Brick Breaker main game screen. Points are earned for destroying bricks and the game is terminated when the ball goes out of the screen on the bottom side. The red represents the walls (including the bottom wall that is used for termination criteria)

To accumulate points in Brick Breaker, players must destroy the bricks. A substantial score bonus is awarded for successfully clearing all bricks from the game screen. Additionally, an element of unpredictability adds to the gameplay, as some bricks may randomly release power-ups when broken. These power-ups temporarily increase the player's paddle size, providing an advantage for a short period.

The simplicity of the game's action space makes Brick Breaker an ideal testbed for reinforcement learning experiments. Players have only a few possible actions at their disposal: moving the paddle left or right, launching the ball if it is still connected to the paddle, or taking no action. This control scheme facilitates the development and evaluation of reinforcement learning algorithms within the game environment, making it a valuable tool for research and experimentation. An example of the game is given in Figure 4.1.

Brick Breaker has a relatively simple state space. To reduce the observation space, the paddle position, ball position, and ball direction are given as the state space. The state space is a vector of shape (6,1). The action space consists of 4 discrete actions. Either the

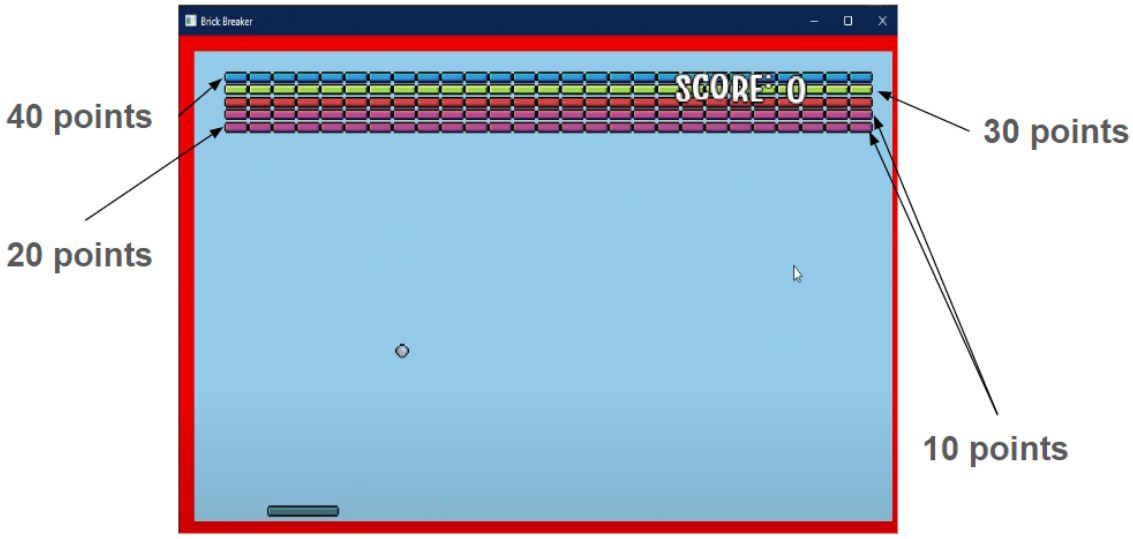

Fig. 4.2: Depicted are the values for breaking each brick. The first two rows (purple) are worth 10 points, increasing from there to the last row (blue) worth 40 points.

paddle will move to the left, move to the right, do nothing, or shoot the ball. If the ball isn't attached to the paddle, shooting the ball does nothing. This allows the algorithm to control when and where to release the ball. Upon release, the ball will travel directly up before bouncing and changing directions slightly based on the bounce off of the paddle.

During the development of the algorithm, several different reward structures were tested. One of the reward structures rewarded points for breaking bricks, as well as for the time that the ball spent alive. Figure 4.2 gives a representation of the point values that bricks have and were used throughout each of the reward structures.

Another reward structure rewarded the direct behavior of bouncing the ball off of the paddle. The idea was that rewarding the algorithm for bouncing the ball would incentivize the algorithm to keep the ball alive, which would naturally increase the score from breaking bricks that a player would regularly experience. However, from each of these reward structures, the one that did the best was the simplest. Simply rewarding the player based on brick destruction incentivized the agent to keep the ball alive for as long as possible as well as get as many points as it could.

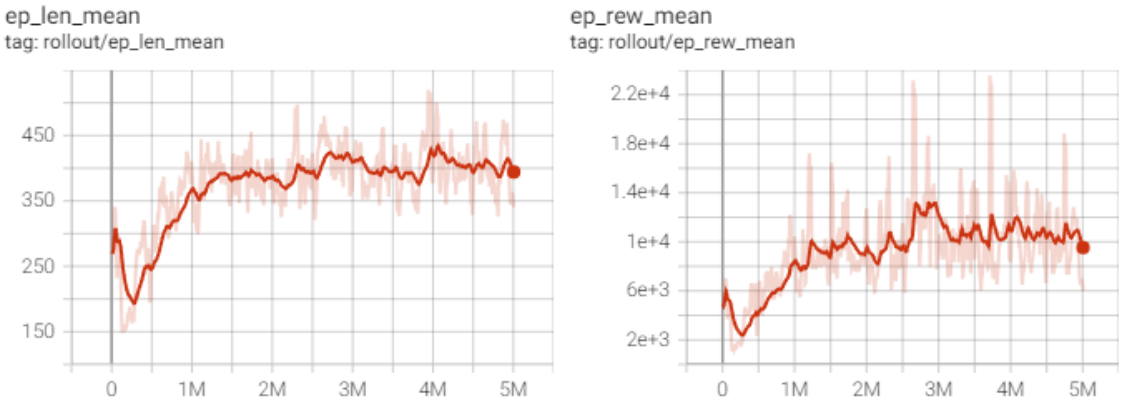

Fig. 4.3: BrickBreaker's tensorboard visualization of training curves. The left figure indicates the episode's mean length and the right figure indicates the episode's mean reward. The increase in reward and length indicates training.

A demonstration of the playing ability of the agent is found at the following YouTube link: https://youtu.be/eXrTgITTCJg, with the corresponding training curves found in Figure 4.3. Training was accomplished, as indicated by the graphs and visualizations, however, improvements can be further developed by augmenting the state that it receives and allowing for more training time. It performs much better than a random agent, but there is room for improvement.

4.5.2 Crypt

Crypt originated as a game jam project, created collaboratively by myself and two colleagues during the HackUSU event in 2022. Originally developed within the Unity game engine, the decision to re-create the same game within Acacia served two key purposes. First, it served as a valuable test of the engine's adaptability, demonstrating its ability to replicate diverse game experiences. Second, it provided a means to validate the implementation of essential engine features that are commonly used in game development.

Crypt is a side scroller that allows the player to flip gravity as they run and defeat monsters. The game's scoring system is multifaceted, combining the distance the player can cover with their ability to eliminate monsters. Additionally, there are two types of monsters: birds that fly into the player and explode, inflicting substantial damage but are weak and

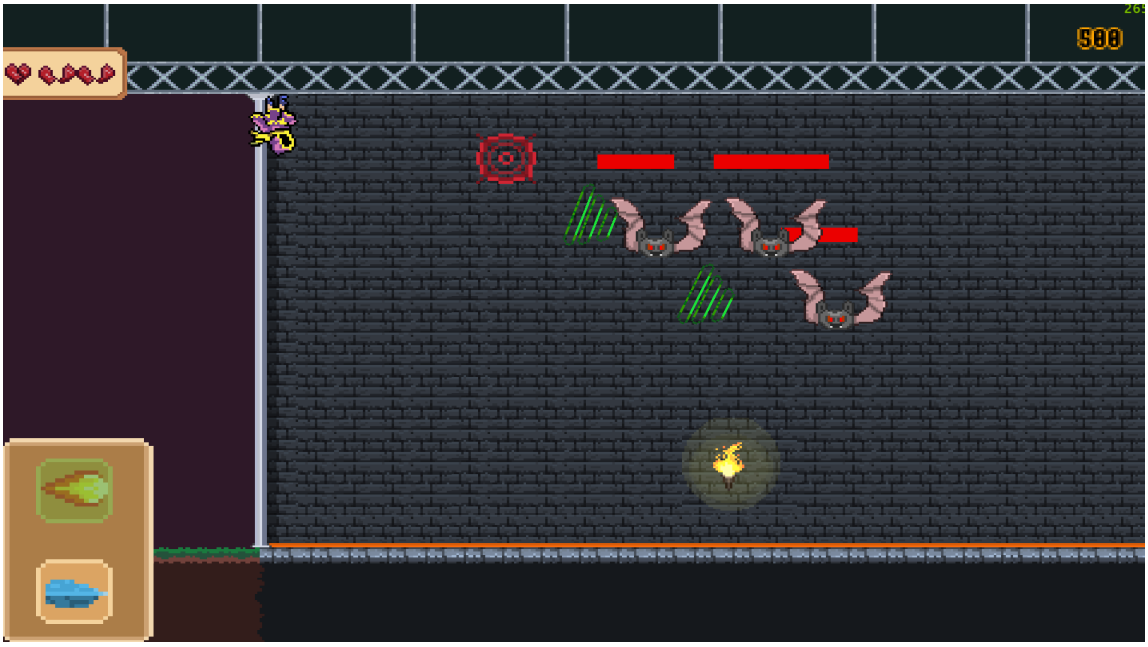

Fig. 4.4: An image of Crypt on the main gameplay screen. Points are earned by defeating monsters and running further distances. The bats deal 1 damage each hit and the birds deal 3 damage but die on impact. Termination is based on the life count being reduced to zero (Loss), or if the entirety of the stage is run (Victory).

relatively low-scoring monsters (worth 10 points), and bats, which pursue the player, shoot at them, and are worth significantly more points (worth 30 points). An example of the game is given in Figure 4.4.

In Crypt, the state space is more complicated than Brick Breaker. In Brick Breaker, the agent doesn't need to know where the bricks are, since by chance it will most likely hit the bricks if it keeps the ball alive. However, in Crypt, since enemies are not always present, knowing the enemies' positions is important to keep itself alive. Thus, the state in the Crypt is a vector of shape (44, 1) which contains the player's position, the aiming cursor position, the position of five enemies, and the position of five of the bullets that are on the screen (both the enemies' and the player's bullets). The reward is based on what a real player's reward would be. Reward is given for the distance that the player can run, as well as small rewards for killing enemies. The exploding birds are worth 10 points, since they only take one hit to destroy, whereas the bats are worth 30 points since they have more health (taking three hits to destroy). If the player manages to complete the game,

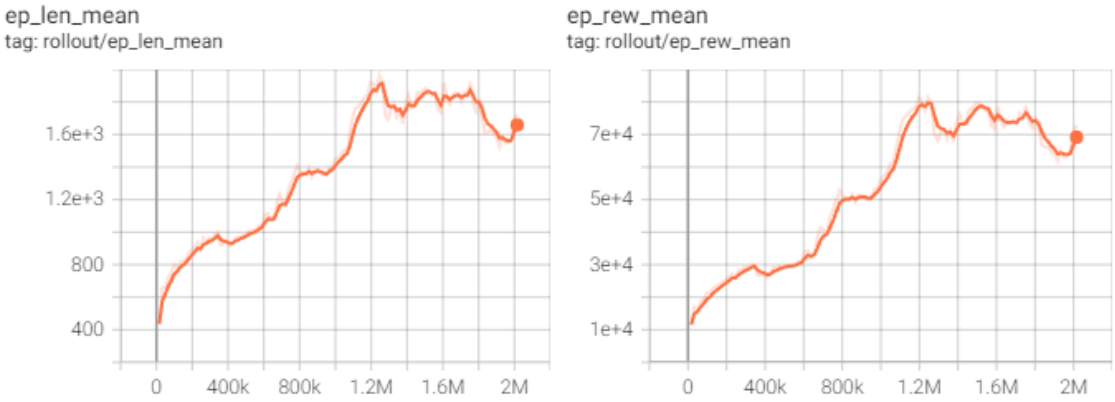

Fig. 4.5: Crypt's tensorboard visualization of training curves. The left figure indicates the episode's mean length and the right figure indicates the episode's mean reward. The increase in reward and length indicates training.

a bonus is given. Another reward structure that was attempted was giving a diminishing reward based on player health. However, this felt extraneous as it might reward or punish certain behaviors because it happened to get hit when it performed a potentially good action. Thus, by allowing the player's death to stop increasing the reward, rather than punishing for damage, it would allow the agent to still incentivize avoidance of damage. Demonstrations of the agent playing can be found here: https://youtu.be/IIRlGGWr_pA with the training curves shown in Figure 4.5.

As observed in the video, the agent typically learned to shoot straight in front of itself and flip gravity to hit areas in the middle of the screen. There are a few instances of targeted aiming, but typically this was the observed behavior. For future improvements, punishments for low accuracy can be implemented to allow the agent to be more accurate and intelligent with its behavior.

4.5.3 SpaceGuy

SpaceGuy is a game originally created by Jeffrey Anderson in Unity (Github link: https://github.com/foggeydoughtnut/SpaceGuy-Unity). With his permission, as well as it being open source, this became the final game implemented in Acacia. Among the three games developed, SpaceGuy was the most intricate.

Fig. 4.6: An image of SpaceGuy in gameplay. Three different enemy types are depicted, with the main player in the middle of the screen. The smaller enemies are weaker, move faster, and hit for less impact. The bigger enemies move much slower, deal three damage per hit instead of one, and have significantly more health. The last enemy is the spawner, which spawns little enemies when the player is within its range. The gameplay is terminated when your health runs out, or upon defeating a main boss, which requires finding a key to unlock the door to fight.

SpaceGuy is a top-down shooter adventure, in which the player commands a spacecraft equipped with two firing methods. The first is a laser weapon, which can be fired continuously and has unlimited ammunition. This weapon is the weakest but ensures that the player is never left without a means of defense.

Additionally, the player has a missile launcher, which provides the player with a cache of six powerful missiles, which are refreshed on death, or upon picking up missile resupply cache powerups. These powerups are found throughout the game world.

As players navigate the game environment, they will encounter various upgrades that enhance their ship's capabilities. These upgrades include a multishot spread weapon, a rapid-fire weapon, missile refills, health potions, and speed boosts. These enhancements are critical in the goal of exploring the game world, finding the boss key, and eventually destroying the level's boss. An example of SpaceGuy is given in Figure 4.6.

SpaceGuy is the most complicated and unique out of all the games developed as it

requires the player to navigate the world. This navigation can't be directly rewarded, as the player needs to find the key before progressing to the boss room. However, much like a regular player, the AI doesn't know that there exists a key or boss room and must stumble upon it by accident. To help it have the chance to ever find the key, when the key is within camera viewing distance, its state position is included in the agent's received state. Much like Crypt, other received states include the player's position and rotation and enemy locations that are within the camera view.

The reward structure is purely based on enemy kills in SpaceGuy, with the boss being worth significantly more than any other enemy in the game. There are two regular enemy types, along with a spawner. The spawner rewards the AI (or player) 30 points and spawns the weakest enemy type whose point value eventually diminishes to zero points. Without this reduction in point value, an agent would conceivably learn to allow the spawner to infinitely spawn enemies and the player could just increase their points based on those enemies. The weakest enemies are worth 10 points, and the larger and more dangerous enemies are worth 70 points. In SpaceGuy, the player also has three lives, with 10 health each life. For each lost life, the player receives a negative 100-point deduction. This is to highly discourage deaths, however slight damage here and there doesn't have a direct effect on its behavior.

Just like the previous two games, demonstrations can be found at the following link: https://youtu.be/gegD1gcCAz8?si=j1Lu7OMU5y7iB9NZ with the corresponding training curves shown in Figure 4.7

Future improvements for SpaceGuy require first, millions of iterations of training to beat the game. Due to the large world that a player can traverse, a significant amount of time is required for the agent to consistently behave in a good manner, as well as recognize the importance of the key and navigate subsequently to the goal. This can be reinforced by rewarding the AI based on its distance to the key, and then to the distance to the boss room, which is another avenue of potential future improvements for beating the game.

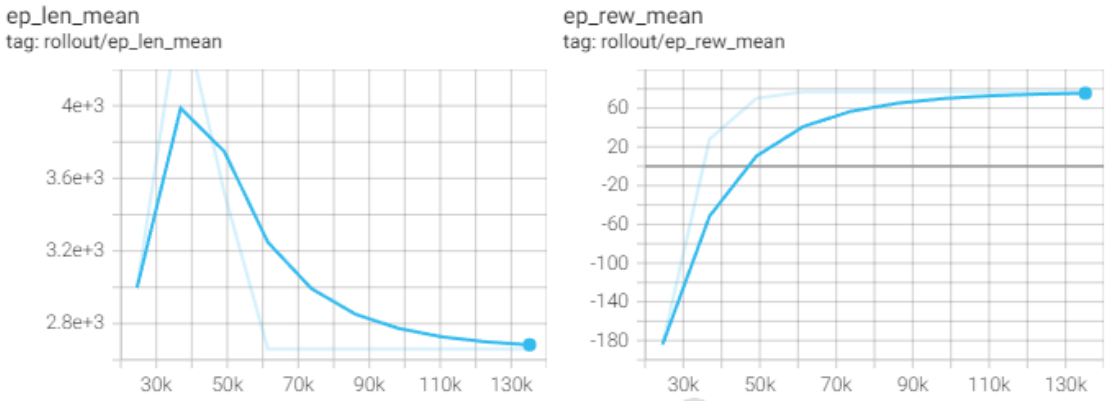

Fig. 4.7: SpaceGuy's tensorboard visualization of training curves. The left figure indicates the episode's mean length and the right figure indicates the episode's mean reward. The increase in reward and length indicates training.

4.6 Conclusions and Future Work

Acacia stands as a versatile game engine with remarkable adaptability, capable of accommodating a wide array of 2D simulation environments. By implementing Python scripting capabilities and built-in support, alongside example scripts for popular machine learning frameworks like PyTorch and Stable Baselines3, Acacia facilitates seamless training of simulations. The engine's ability to generate multiple instances of the same world enables automated simultaneous training of numerous agents. Acacia also ensures compatibility with vectorized environments (such as those utilized in Stable Baselines), along with other frameworks supporting multi-agent training.

The primary contribution of this thesis lies in simplifying the training process within a simulation environment, supporting both Python and C++ as scripting languages. Leveraging Python scripting and the engine's design akin to a Gymnasium-style API, users can harness a multitude of common reinforcement learning libraries. This flexibility extends to the choice of machine learning frameworks, enabling easy access to trained models outside the simulation environment. This capability holds significant potential, allowing for the training of agents within the simulated environment and subsequent deployment onto real-world systems, such as robots and transportation.

Another significant contribution of Acacia lies in its status as an open-source, highly

adaptable game engine. By adhering to the ECS design paradigm, Acacia offers developers the flexibility to modify underlying modules, tailoring them to meet specific project requirements. This modular approach not only facilitates customization but also promotes code reusability and scalability, enabling developers to create complex and unique game experiences with ease.

The open-source nature of Acacia (https://github.com/sonorousduck/Acacia) further amplifies its value. By providing unrestricted access to the engine's source code, developers can take ownership of their projects and shape them according to their vision. This freedom to modify and extend Acacia's capabilities allows for the creation of games and simulations that precisely model specific requirements, rather than being constrained by a rigid, pre-defined architecture.

Looking ahead, future development efforts will focus on enhancing the engine's physics engine to better simulate real-world scenarios, particularly in the realm of robotics development. Additionally, ongoing research will explore utilizing Acacia as a platform for simulation demonstration and training, offering improved visualization and encapsulation compared to traditional Python-based simulations. This approach underscores Acacia's commitment to advancing not only game development but also broader applications in simulation and training across various domains.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[4] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 737–744.

[5] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 411–444, 2022.

[6] B. Singh, R. Kumar, and V. P. Singh, "Reinforcement learning in robotic applications: a comprehensive survey," *Artificial Intelligence Review*, pp. 1–46, 2022.

[7] R. Anderson and M. Harper, "Save money, get charged: Facility-tied energy management with unknown and unscheduled ev charging," in *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*, 2023, pp. 986–991.

[8] S. Kockara, T. Halic, K. Iqbal, C. Bayrak, and R. Rowe, "Collision detection: A survey," in *2007 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2007, pp. 4046–4051.

[9] C. Ericson, *Real-time collision detection*. Crc Press, 2004.

[10] M. C. Lin, D. Manocha, J. Cohen, and S. Gottschalk, "Collision detection: Algorithms and applications," *Algorithms for robotic motion and manipulation*, pp. 129–142, 1997.

[11] Y. J. Heo, D. Kim, W. Lee, H. Kim, J. Park, and W. K. Chung, "Collision detection for industrial collaborative robots: A deep learning approach," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 740–746, 2019.

[12] G. Szauer, *Game Physics Cookbook: Discover over 100 easy-to-follow recipes to help you implement efficient game physics and collision detection in your games*. Packt Publishing Ltd, 2017.

[13] J. Schauer and A. Nüchter, "Collision detection between point clouds using an efficient kd tree implementation," *Advanced Engineering Informatics*, vol. 29, no. 3, pp. 440–458, 2015.

[14] J. Cao and M. Wang, "A fast and generalized broad-phase collision detection method based on kd-tree spatial subdivision and sweep-and-prune," *IEEE Access*, 2023.

[15] K. Zhou, Q. Hou, R. Wang, and B. Guo, "Real-time kd-tree construction on graphics hardware," *ACM Transactions on Graphics (TOG)*, vol. 27, no. 5, pp. 1–11, 2008.

[16] O. Procopiuc, P. K. Agarwal, L. Arge, and J. S. Vitter, "Bkd-tree: A dynamic scalable kd-tree," in *Advances in Spatial and Temporal Databases: 8th International Symposium, SSTD 2003, Santorini Island, Greece, July 2003. Proceedings 8*. Springer, 2003, pp. 46–65.

[17] S. Lefebvre and H. Hoppe, "Perfect spatial hashing," *ACM Transactions on Graphics (TOG)*, vol. 25, no. 3, pp. 579–588, 2006.

[18] M. Eitz and G. Lixu, "Hierarchical spatial hashing for real-time collision detection," in *IEEE International Conference on Shape Modeling and Applications 2007 (SMI '07)*, 2007, pp. 61–70.

[19] S. M. Putri and D. Maizana, "Optimal smart grid management system in campus building," *Jurnal Nasional Teknik Elektro*, pp. 139–143, 2020.

[20] S. Chapaloglou, A. Nesiadis, P. Iliadis, K. Atsonios, N. Nikolopoulos, P. Grammelis, C. Yiakopoulos, I. Antoniadis, and E. Kakaras, "Smart energy management algorithm for load smoothing and peak shaving based on load forecasting of an island's power system," *Applied energy*, vol. 238, pp. 627–642, 2019.

[21] Ç. Iris and J. S. L. Lam, "A review of energy efficiency in ports: Operational strategies, technologies and energy management systems," *Renewable and Sustainable Energy Reviews*, vol. 112, pp. 170–182, 2019.

[22] K. Voss, I. Sartori, and R. Lollini, "Nearly-zero, net zero and plus energy buildings," *REHVA Journal, Dec*, 2012.

[23] O. I. Asensio, C. Z. Apablaza, M. C. Lawson, and S. E. Walsh, "A field experiment on workplace norms and electric vehicle charging etiquette," *Journal of Industrial Ecology*, vol. 26, no. 1, pp. 183–196, 2022.

[24] J. H. Lee, D. Chakraborty, S. J. Hardman, and G. Tal, "Exploring electric vehicle charging patterns: Mixed usage of charging infrastructure," *Transportation Research Part D: Transport and Environment*, vol. 79, p. 102249, 2020.

[25] W. Li, M. Rentemeister, J. Badeda, D. Jöst, D. Schulte, and D. U. Sauer, "Digital twin for battery systems: Cloud battery management system with online state-of-charge and state-of-health estimation," *Journal of energy storage*, vol. 30, p. 101557, 2020.

[26] M. Shen and Q. Gao, "A review on battery management system from the modeling efforts to its multiapplication and integration," *International Journal of Energy Research*, vol. 43, no. 10, pp. 5042–5075, 2019.

[27] A. Mehrabi, H. K. Nunna, A. Dadlani, S. Moon, and K. Kim, "Decentralized greedy-based algorithm for smart energy management in plug-in electric vehicle energy distribution systems," *IEEE Access*, vol. 8, pp. 75 666–75 681, 2020.

[28] Y. Wu, Z. Wang, Y. Huangfu, A. Ravey, D. Chrenko, and F. Gao, "Hierarchical operation of electric vehicle charging station in smart grid integration applications—an overview," *International Journal of Electrical Power & Energy Systems*, vol. 139, p. 108005, 2022.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[30] L. Dong, S. Xu, and B. Xu, "Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition," in *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2018, pp. 5884–5888.

[31] K. Irie, A. Zeyer, R. Schlüter, and H. Ney, "Language modeling with deep transformers," *arXiv preprint arXiv:1905.04226*, 2019.

[32] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[34] S. Karita, N. Chen, T. Hayashi, T. Hori, H. Inaguma, Z. Jiang, M. Someki, N. E. Y. Soplin, R. Yamamoto, X. Wang *et al.*, "A comparative study on transformer vs rnn in speech applications," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2019, pp. 449–456.

[35] D. Kim, D. Kwon, L. Park, J. Kim, and S. Cho, "Multiscale lstm-based deep learning for very-short-term photovoltaic power generation forecasting in smart city energy management," *IEEE Systems Journal*, vol. 15, no. 1, pp. 346–354, 2020.

[36] M. Palak, G. Revati, and A. Sheikh, "Smart building energy management using deep learning based predictions," in *2021 North American Power Symposium (NAPS)*. IEEE, 2021, pp. 1–6.

[37] G. Dong, W. Han, and Y. Wang, "Dynamic bayesian network-based lithium-ion battery health prognosis for electric vehicles," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 11, pp. 10 949–10 958, 2020.

[38] R. Sekhar, P. Shah, S. Panchal, M. Fowler, and R. Fraser, "Distance to empty soft sensor for ford escape electric vehicle," *Results in Control and Optimization*, vol. 9, p. 100168, 2022.

[39] X. Tang, C. Zou, K. Yao, J. Lu, Y. Xia, and F. Gao, "Aging trajectory prediction for lithium-ion batteries via model migration and bayesian monte carlo method," *Applied Energy*, vol. 254, p. 113591, 2019.

[40] E. D. Kostopoulos, G. C. Spyropoulos, and J. K. Kaldellis, "Real-world study for the optimal charging of electric vehicles," *Energy Reports*, vol. 6, pp. 418–426, 2020.

[41] F. Tafazzoli, H. Frigui, and K. Nishiyama, "A large and diverse dataset for improved vehicle make and model recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, pp. 1–8.

[42] N. Schneider, F. Piewak, C. Stiller, and U. Franke, "Regnet: Multimodal sensor registration using deep neural networks," in *2017 IEEE intelligent vehicles symposium (IV)*. IEEE, 2017, pp. 1803–1810.

[43] T. L. Gibson and N. A. Kelly, "Solar photovoltaic charging of lithium-ion batteries," in *2009 IEEE Vehicle Power and Propulsion Conference*, 2009, pp. 310–316.

[44] A. Oppermann, "A deep dive into deep q-learning," Dec 2021. [Online]. Available: https://builtin.com/artificial-intelligence/deep-q-learning

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

[46] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[47] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[48] D. Bick and M. Wiering, "Towards delivering a coherent self-contained explanation of proximal policy optimization," Ph.D. dissertation, Master's thesis, 2021.[Online]. Available: https://fse. studenttheses. ub . . . , 2021.

APPENDICES

CURRICULUM VITAE

James Ryan Anderson

Published Conference Papers

- Stealth Centric A*: Bio-Inspired Navigation for Ground Robots, Ryan Anderson, Taylor Anderson, Carter Bailey, Jeffrey Anderson, and Mario Harper, in *Proc. IEEE Int. Conf. on Robotic Computing (IRC)*, 2023.

- Save Money, Get Charged: Facility-Tied Energy Management with Unknown and Unscheduled EV Charging, Ryan Anderson and Mario Harper, in *Proc. IEEE Int. Conf. on Transportation Systems (ITSC)*, 2023.

Published Journal Articles

- Charger Reservation Web Application, Ryan Anderson, Jeffrey Anderson, Taylor Anderson, and Mario Harper, *Software Impacts*, vol. 18, pp. 100589, 2023

- Stealth Centric Autonomous Robot Simulator (SCARS), Jeffrey Anderson, Ryan Anderson, Taylor Anderson, Carter Bailey, and Mario Harper, *Software Impacts*, vol. 16, pp. 100497, 2023

- Power and transportation collection and visualization, Ryan Anderson, Taylor Anderson, and Mario Harper, *Software Impacts*, vol. 14, pp. 100386, 2022