CptS 570 MACHINE LEARNING HOMEWORK-4

Sharath Kumar Karnati

27th November 2024

Questions

Question 1: Implementation of Q-Learning Algorithm and Experimentation

Solution : Given a Gridworld has 10xt10 = 100 differen states. I designed the Q-learning algorithm using e-greedy and Boltzman exploration techniques. With 5000 iterations per epoch, I ran 20000 iterations. It started converg- ing around 12000-13000 iteration.

1. Convergence, Experiment/(ϵ -greedy- 0.1)

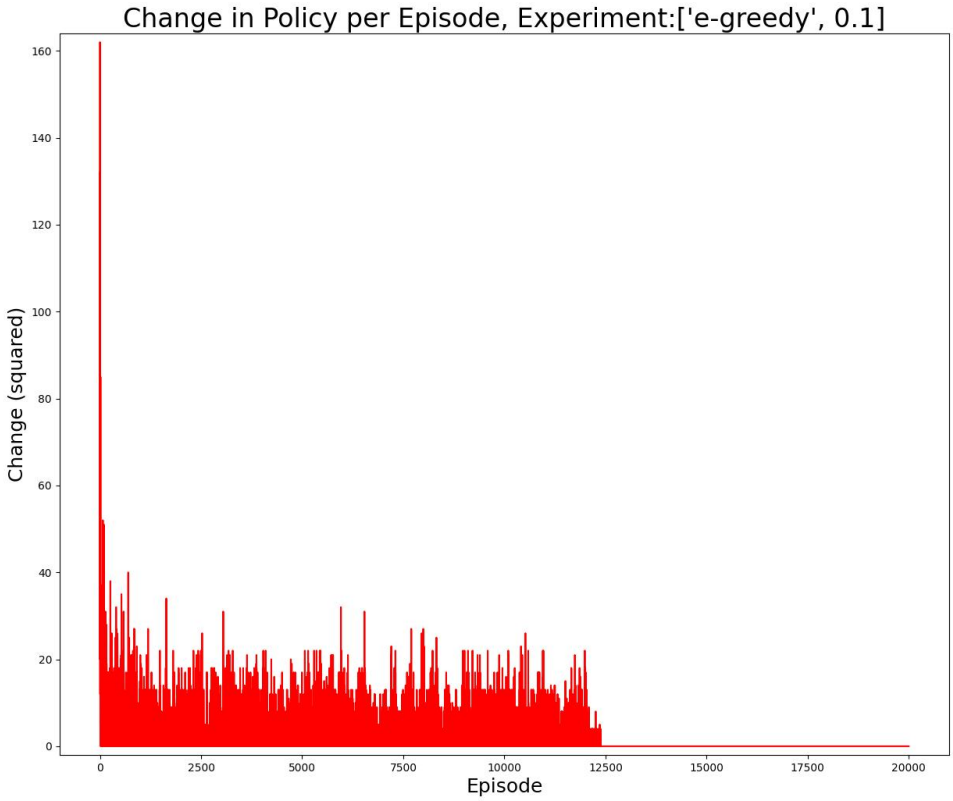

Figure 1: e-greedy, 0.1

2. Convergence, Experiment/(ϵ -greedy- 0.2)

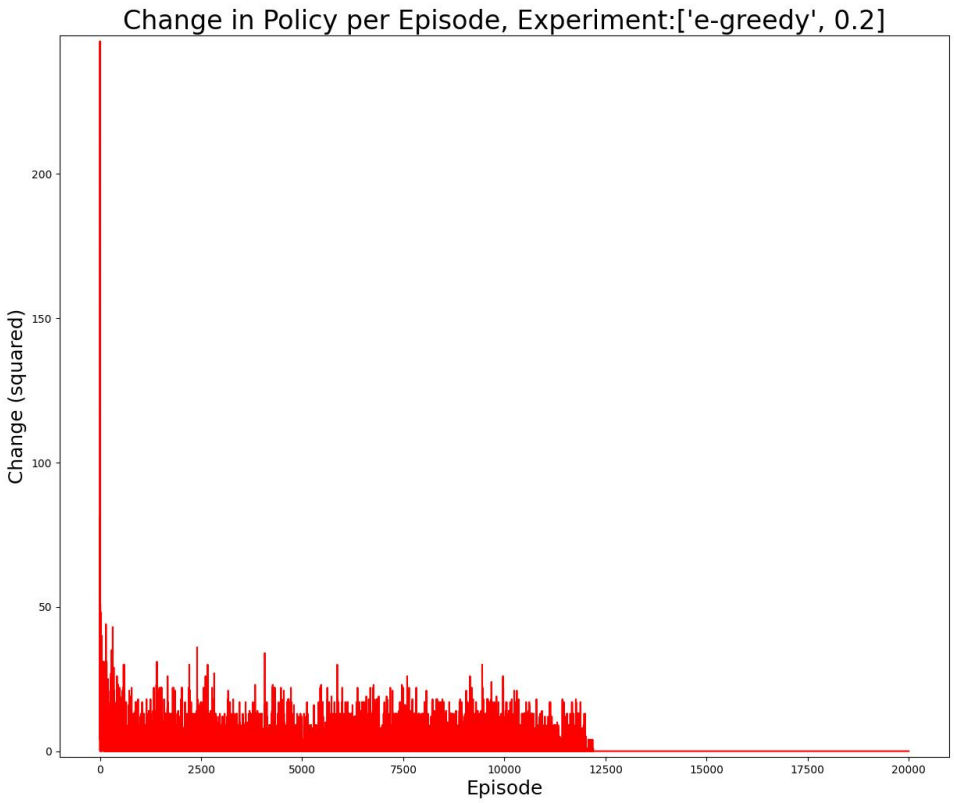

Figure 2: e-greedy, 0.2

3. Convergence, Experiment/(ϵ -greedy- 0.3)

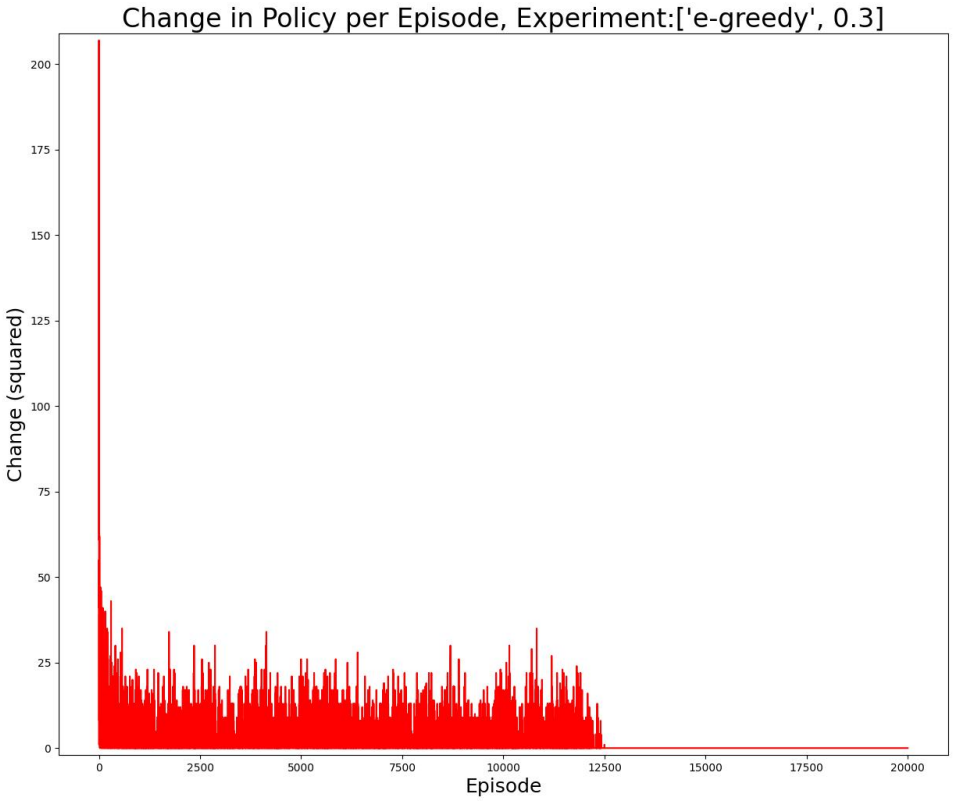

Figure 3: e-greedy, 0.3

4. Convergence, Experiment/(Boltzmann, 1)

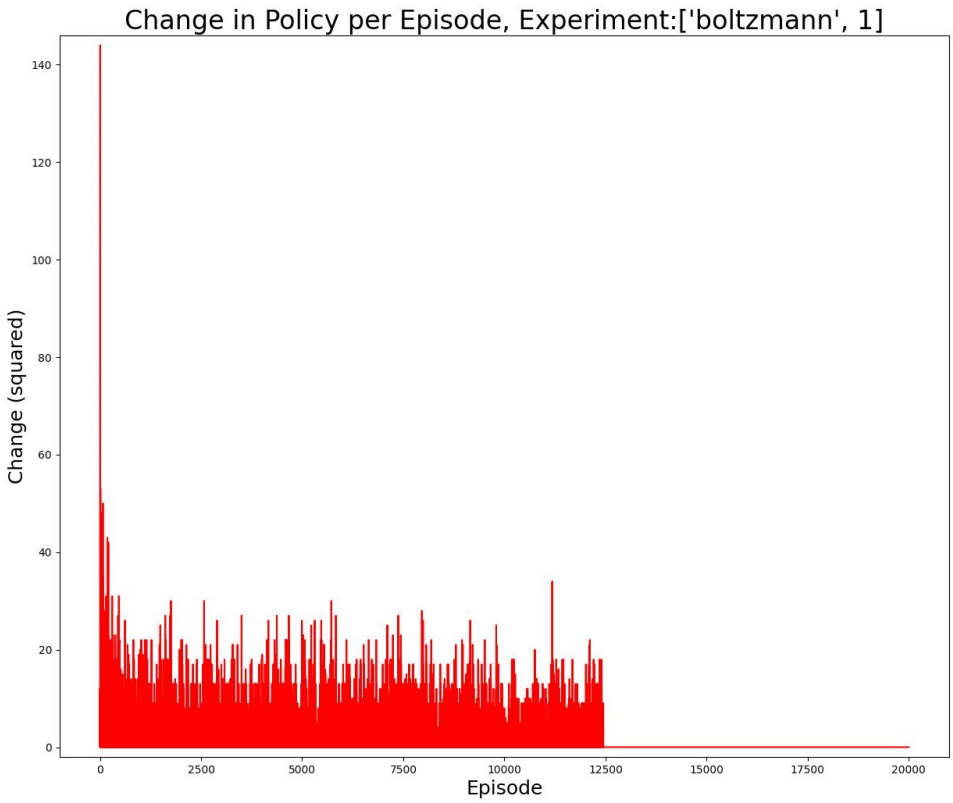

Figure 4: boltzmann, 1

5. Convergence, Experiment/(Boltzmann, 5)

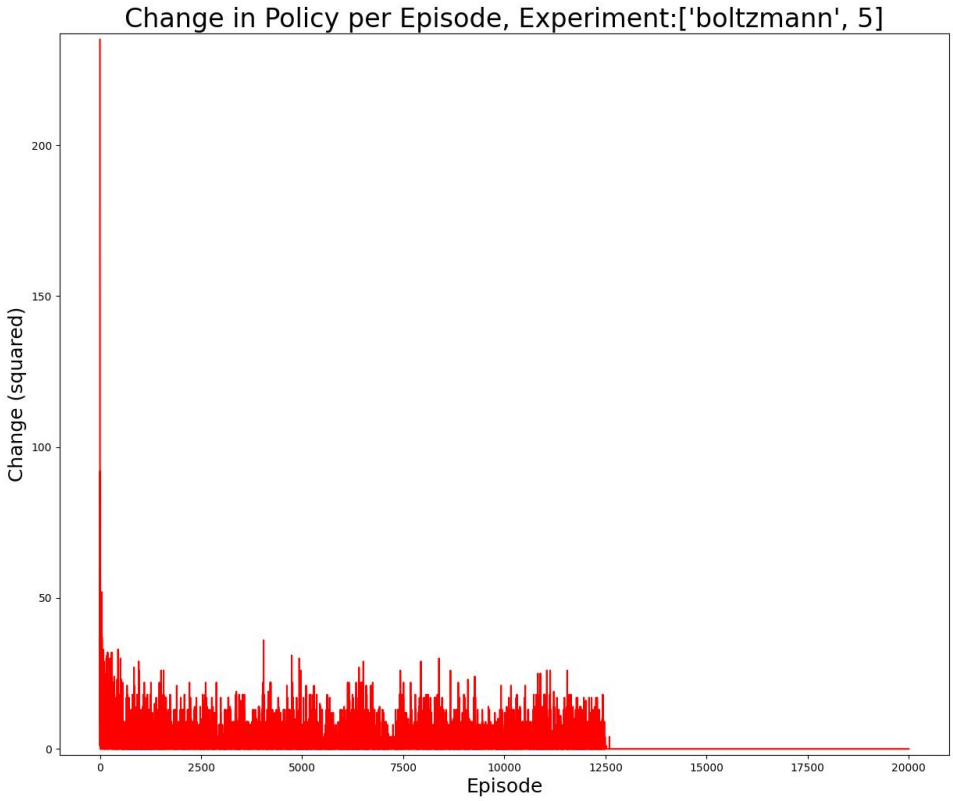

Figure 5: boltzmann, 5

6. Convergence, Experiment/(Boltzmann, 10)

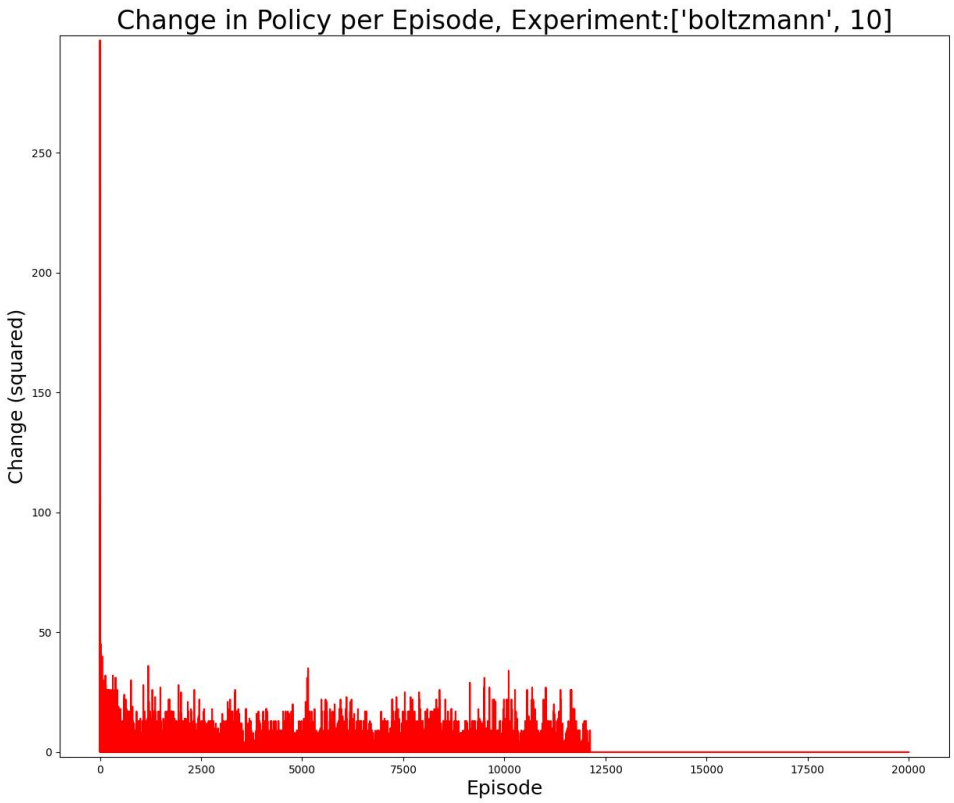

Figure 6: boltzmann, 10

7. Convergence, Experiment/(Boltzmann, 20)

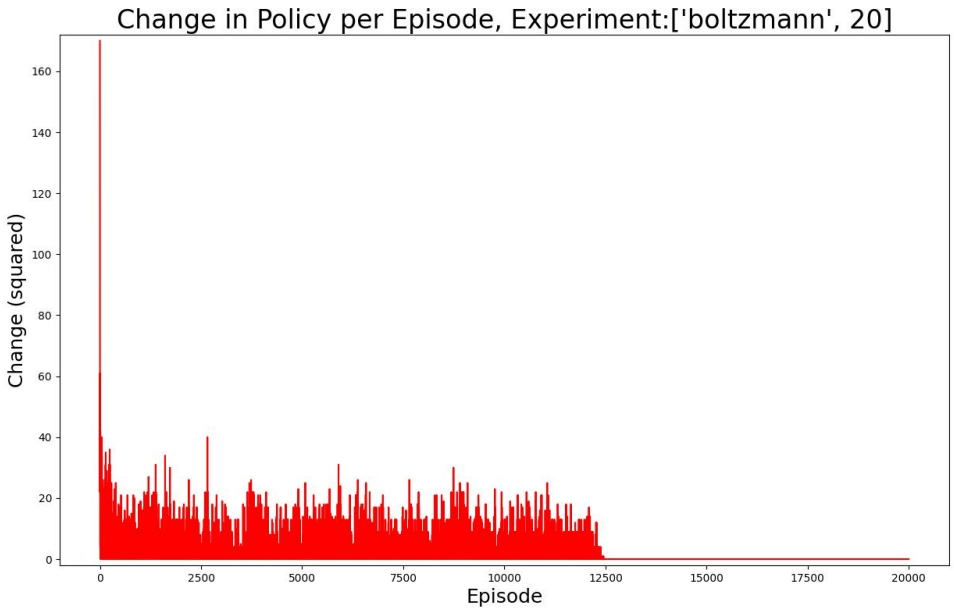

Figure 7: boltzmann, 20

Question 2: CNN

Solution:

The model trained on the Fashion MNIST dataset using a Convolutional Neural Network (CNN) architecture with 4 convolutional layers, followed by an average pooling layer and a fully connected layer. The model was trained for 10 epochs, and the training accuracy improved from 82.07% in the first epoch to 86.11% in the tenth. The testing accuracy followed a similar trend, starting at 81.38% and reaching 85.37% by the end of the training. Throughout the training process, the model consistently showed improvements in accuracy, with the best performance achieved in the later epochs.

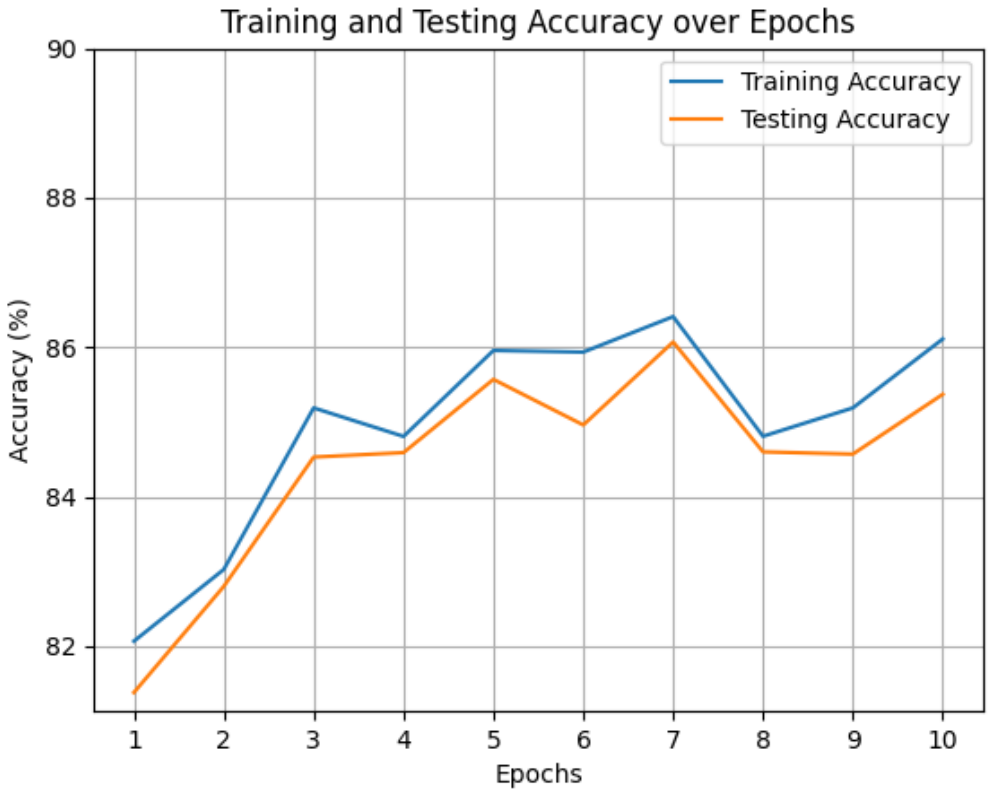

Figure 8: Testing vs Training

Question 3: D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Franois Crespo, Dan Dennison: Hidden Technical Debt in Machine Learning Systems. NIPS 2015: 2503-2511

Summary:

The paper "Hidden Technical Debt in Machine Learning Systems" by Sculley et al. explores the challenges of maintaining machine learning (ML) systems, emphasizing the often-overlooked technical debt that arises after deployment. It highlights the complexity of monitoring and maintaining ML systems compared to traditional software, owing to their unique characteristics and dependencies.

The authors discuss how ML systems differ from conventional software in that changes to one component often propagate across the system, making it highly sensitive to input variations and feature modifications. To address this, techniques like model isolation and ensembling can help mitigate errors. Additionally, monitoring predicted behavior through methods like slice-by-slice analysis can improve system stability.

ML systems are also susceptible to cascading errors caused by frequent small adjustments to meet changing demands. Reusing the same model for corrections can alleviate this issue. Visibility issues in multi-system environments, stemming from unregistered model users, create tight coupling. Implementing strict service-level agreements can help resolve this challenge.

Feedback loops are a critical issue in ML systems, where models directly influence the training data used in subsequent iterations. To counter this, data and models should be managed separately. Hidden feedback loops, where two systems unintentionally interact, can also arise and are harder to detect.

The paper addresses anti-patterns like "glue code" and the "pipeline jungle," where evolving systems introduce deprecated paths and deadlocks. These can be mitigated by encapsulating black-box functionalities using well-defined APIs and fostering collaboration between ML research and engineering teams.

Another challenge is abstraction within ML systems, which often involves multilingual configurations, fragile prototypes, and unrefined code. Unlike traditional systems, ML systems rely on intricate parameter tuning, demanding reusable, well-documented, and fault-tolerant solutions. Thorough testing and monitoring are essential, particularly in dynamic real-world applications. Issues like reproducibility, process inconsistencies, and cultural adaptation further complicate ML system maintenance.

Data dependencies, often harder to detect than code dependencies, present additional risks. Input instability, caused by interconnected models, exacerbates these issues. Version control systems that preserve reliable discoveries can mitigate dependency risks. Redundant features and unnecessary complexity, offering minimal benefit, should be identified and managed using feature evaluation techniques. Automated systems to track and annotate dependencies can address the lack of static analysis in ML systems.

The paper concludes that building robust ML systems requires focusing on algorithmic testability, managing data and system dependencies, evaluating changes effectively, and ensuring knowledge transfer. By addressing these challenges, ML systems can be made more sustainable, maintainable, and resilient.

Question 4: Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley: The ML test score: A rubric for ML production readiness and technical debt reduction. BigData 2017: 1123-1132

Summary:

The paper *"The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction"* by Breck et al. addresses the challenges of ensuring reliability in machine learning (ML) systems, both in large-scale applications and smaller experimental settings. It highlights the technical debt that arises when ML systems are deployed into production and proposes a comprehensive framework consisting of 28 testing and monitoring use cases applicable to production-level systems. These use cases are grouped into four main categories:

- Feature and Data Testing
- Model Development Testing
- ML Infrastructure Testing
- ML Monitoring Testing

The paper outlines the stages of ML system development, starting from data preparation and feature engineering to deployment and monitoring in production. Unlike traditional programming, where input data is processed by static code to produce outputs, ML systems are heavily influenced by both training and testing data. This dependency introduces unique challenges in maintaining system performance across use cases. A survey of production systems used by companies like Google revealed that 80% of teams lacked comprehensive test coverage for their ML systems, though significant improvements were observed in the 20% of teams with testing practices in place.

The framework begins with **Feature and Data Testing**, emphasizing the importance of understanding feature statistics, schema validation, and predictive power analysis. For example, calculating correlation coefficients can help identify the significance of individual features. The paper also highlights practical considerations like RAM usage, inference latency, and privacy management, which are crucial for effective feature engineering. Implementing these tests simplifies the process of adding new features and conducting unit testing.

The next focus is on **Model Development Testing**, which includes maintaining proper version control for models to facilitate updates, rollbacks, and review processes. Metrics like offline proxies and impact measurements help evaluate model performance. Testing with diverse data and optimizing hyperparameters are critical steps in refining model behavior before production deployment.

In the **Infrastructure Setup** section, the authors describe tools and processes for deploying ML models across environments. This includes pipeline testing, model specification validation, and ensuring debugability to streamline development workflows. Models should provide feedback throughout the deployment process and allow for modular testing to save time during iterations.

The final category, **ML Monitoring Testing**, emphasizes the importance of ongoing performance evaluation in real-world scenarios. As models encounter changing data, monitoring systems must ensure reliability through adherence to a set of best practices. Retraining, anomaly detection, and adherence to key rules are vital for maintaining system integrity over time.

Overall, the paper presents a robust checklist and criteria for building reliable and maintainable ML systems. By integrating systematic testing into the development life cycle, the proposed framework addresses critical aspects of technical debt and ensures production readiness for ML applications.