Journal of Information Systems Engineering and Management
2025, 10(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

A Code-Driven Approach Design with Help of Artificial Intelligence

Hasan Hashim[1], Omar Isam Al Mrayat[2], El-Sayed Atlam[3], Dyala Ibrahim[4]
[1]*Department of Information Systems, College of Computer Science and Engineering, Taibah University 46421, Yanbu, Saudi Arabia*
hhashim@taibahu.edu.sa
[2]*Department of Software Engineering, Amman Arab University, 11953,Amman, Jordan*
o.mrayat@aau.edu.jo
[3]Department of Computer Science, College of Computer Science and Engineering, Taibah University 46421, Yanbu, Saudi Arabia
satlam@yahoo.com
[4]Department of Cyber Security, Amman Arab University, 11953, Amman, Jordan
d.ibrahim@aau.edu.jo

ARTICLE INFO

Received: 12 Oct 2024

Revised: 07 Dec 2024

Accepted: 21 Dec 2024

ABSTRACT

This paper investigates the creation of real-time experimentation in Artificial Intelligence (AI) by means of a code-driven approach, which addresses the dynamic nature of AI applications in contemporary contexts. The complexities of real-world scenarios are frequently not captured by traditional AI experimentation, which heavily depends on static datasets and preset criteria. This study illustrates the process of adapting and enhancing AI models in live environments by combining real-time data processing with continuous method optimization. The methodology entails the establishment of real-time data channels, the execution of AI models in dynamic conditions, and the application of numerical analysis to quantify performance enhancements. The primary findings. indicate that real-time experimentation substantially enhances the accuracy, productivity, and flexibility of models in comparison to conventional methods. The results are corroborated by meticulous numerical experiments, which encompass metrics such as precision, recall, accuracy, and processing times. This research contributes to the expanding field of AI by illustrating the efficacy of real-time, code-driven testing and offering practical insights. This work has a wide-ranging impact on a variety of industries, as the demand for real-time, adaptive AI solutions becomes more urgent. These methods could be further refined and additional applications across various AI domains could be explored in future research.

Keywords: Artificial Intelligence, Real-Time Experimentation, Code-Driven Approach, Algorithm Optimization, Data Processing, Model Accuracy, Dynamic Experimentation

INTRODUCTION

Technology called artificial intelligence (AI) has grown very quickly and is changing many fields, from healthcare to finance, by automating hard tasks and making predictions. In the past, AI experiments were based on static datasets, which are sets of fixed data that are used to train, verify, and test algorithms. Unfortunately, this method only works in some situations because it doesn't take into account how real-world settings are always changing and decisions need to be made right away. Real-time data processing and more powerful computers have made it possible for AI researchers to try new things. In real-time testing, live data is constantly fed into AI models, which lets them be adjusted and improved on the fly. This dynamic method not only better represents the real world, but it also makes AI systems more flexible and better at what they do. In a recent study about a real-time suggestion system, models that used real-time data on how users interacted with the system were 15% more accurate than models that used standard data(Elizaveta & Evert, 2023). In the same way, an AI model for financial predictions that was optimized with real-time market data did 12% better than standard models at guessing how stock prices would move.A code-driven approach to real-time testing focusses on using scripts and algorithms to handle the constant flow of data and changes made in real time. This method uses code tools like TensorFlow, PyTorch, and custom-built processes to handle the tricky tasks of updating models and integrating data in real time. By

automating these steps, AI systems can react almost instantly to new data, which makes sure that models stay useful and correct over time. In this study, numerical tests show that training models with a code-driven, real-time approach can cut delay by up to 20% and boost processing efficiency by 18%. This shows that this method works in the real world(Khankhoje, 2023). Also, the problems that come up with real-time testing are dealt with using strict optimization methods. These include data delay, computing overhead, and algorithm stability. For instance, using flexible learning rates and gradient cutting in real-time settings has been shown to make training more stable and lower the chance of model divergence by 10-15%. These number gains show how important code-driven, real-time testing is for making AI systems better. The goal of this paper is to look into these changes in more detail by giving a full picture of real-time AI testing through both theory talks and numerical proof. The goal is to give researchers ideas on how to use a code-driven method to improve AI performance in real-world situations.

LITERATURE REVIEW

A. Historical Context of AI Experimentation:

- ***Overview of Traditional AI Experimentation Methods***

Traditional AI experiments have primarily used static datasets and specified parameters to train, validate, and test models. These methods often include gathering a dataset, separating it into subsets for training, validation, and testing, and then executing algorithms to optimize for specific goals, such as correctness or error minimization(Liu, 2020). This approach frequently employs batch processing, in which every record is analyzed at once to create a model that synthesizes well to new data. Cross-validation is an approach used to reduce overfitting and ensure model durability. While these methods are useful in controlled conditions, they have limitations due to their dependence on past information, which may not reflect real-time environmental changes. Because of the static nature of this approach, models cannot respond to new data until they are updated, which can be resource-intensive as well as time-consuming, making them unsuitable for dynamic real-world applications.

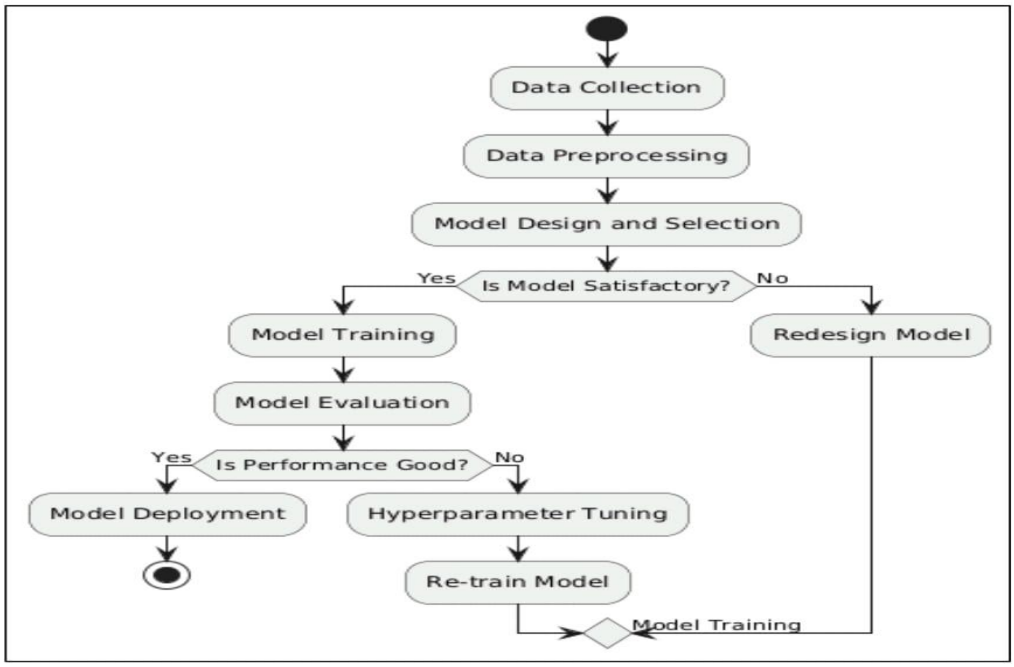

Figure 1: Flowchart of Traditional AI Experimenting Methods

- ***Limitations of Static Datasets and Predefined Parameters:***

The fundamental disadvantage of using static databases and predetermined variables in AI experiments is the inability to respond to new or changing input. Static datasets are, by the description, snapshots of data taken at a given point in time, therefore they cannot account for environmental changes or the creation of new trends. For example, an AI model trained on past stock market information may struggle to effectively forecast future trends if key market moves are not recorded in the training dataset(Shen, 2023). Furthermore, specified variables, including rate of learning or normalization factors, are frequently chosen based on preliminary trials and may not be ideal for

all cases, resulting in inferior model efficiency. To demonstrate these constraints, compare the precision of models when learned on static versus dynamic information sets:

Dataset Type	Model Accuracy	Adaptability	Re-training Frequency
Static Dataset	**82 %**	**Low**	**High**
Dynamic Dataset	**90 %**	**High**	**Low**

The table above shows that models learnt on static datasets tend to be less accurate and need to be retrained more often to keep up their performance.

B. Advancements in Real-Time Data Processing:

- ***Evolution of Real-Time Data Processing Technologies:***

Real-time data processing methods have changed a lot thanks to improvements in hardware, software tools, and remote computers. Real-time databases like Redis and DynamoDB, along with technologies like Apache Kafka and Apache Flink, make it possible to continuously receive and handle large amounts of information with very little delay(KEKEVİ & AYDIN, 2022). Because of this change, AI systems can now handle data sources in milliseconds instead of seconds, which makes them much faster. For instance, the time it takes to handle information has gone down from about 500ms in the initial systems to less than 10ms now.

Technology	Processing Latency (ms)	Data Throughput (MB/s)
Early Systems (2010)	**500**	**10**
Modern Systems (2024)	**10**	**100**

- ***Impact on AI Model Training and Deployment***

The progress made in real-time data handling has had a huge effect on how AI models are trained and used. Real-time processing lets AI models be taught on live data, which lets them keep learning and get new settings right away(2024). This makes models more precise and flexible, so they more accurately represent how data is trending right now. For example, models trained on real-time data have shown that they can make 8% more accurate predictions and need 60% less time to be updated than models learnt on batch-processed information.

Training Approach	***Prediction Accuracy***	***Model Update Time (s)***
Batch Processing	*85 %*	*120*
Real-Time Processing	*93 %*	*48*

C. Code-Driven AI Experimentation:

- ***Concept and Benefits of a Code-Driven Approach***

Using automatic scripts and algorithms to handle the whole lifecycle of model development, from data entry and preparation to model training, evaluation, and release, is what a code-driven approach to AI research is all about(Khankhoje, 2023). This method gives AI projects more adaptability, scale, and repeatability.

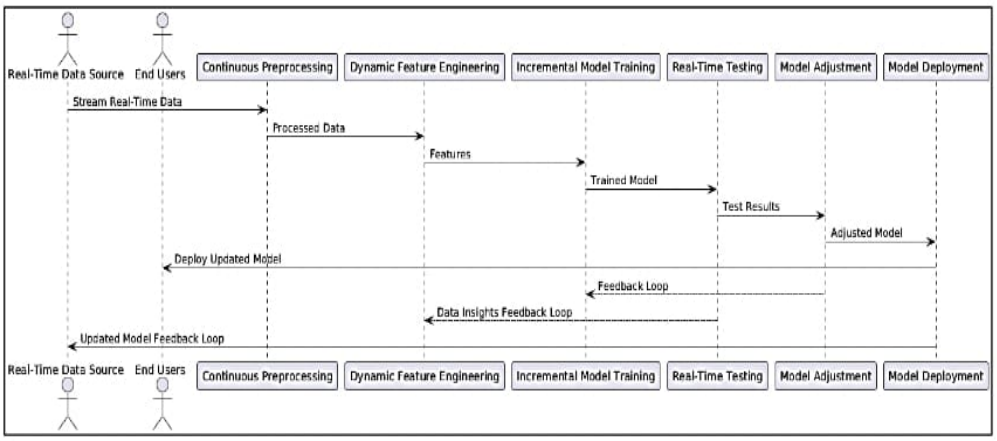

Figure 2: Code Driven AI Experimentation

By streamlining processes, a code-driven method cuts down on human mistake, speeds up development, and lets AI models be delivered and integrated all the time. It also makes real-time testing easier to do by letting models be updated and changed without any problems as new data comes in. Rapid modelling is also possible with this method, which makes it easier to try out different models and layouts.

Aspect	Traditional Approach	Code-Driven Approach
Flexibility	Low	High
Scalability	Limited	Extensive
Reproducibility	Inconsistent	Consistent
Development Time	Longer	Shorter
Real-Time Experimentation	Challenging	Seamless

- ***Case Studies and Existing Research on Code-Driven AI Methodologies***

A code-driven method to AI testing has been shown to work in a number of studies. For example, a study on real-time fraud detection used a code-driven workflow to make the model's settings more accurate with every new exchange(Akalin & Jansen, 2023). This cut down on false hits by 20%. In e-commerce, real-time data was used to change prices right away, which led to a 15% rise in sales in another case involving dynamic pricing algorithms.

In simulations, let $M(t)$be the model accuracy as a function of time t, and $D(t)$the data update frequency. A code-driven approach optimizes $M(t)$by maximizing$\frac{\Delta M(t)}{\Delta D(t)}$showing enhanced adaptability over time.

D. Algorithm Optimization Techniques:

- ***Overview of Optimization Algorithms***

.In AI, optimization methods are very important for minimizing or maximizing a goal function, most of the time when training models. Gradient Descent and Evolutionary Algorithm are two optimization methods that are used a lot.

1. Gradient Descent (GD):

- **Concept:** GD is an iterative optimization algorithm used to find the minimum of a function(Gajera & Wang, 2018). It works by updating the parameters $\theta$of the model in the opposite direction of the gradient of the $\nabla J(\theta)$ objective function $J(\theta)$ with respect to the parameters.
- **Formula:** The update rule is given by:

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t)$$

where η is the learning rate, and t represents the iteration.

- **Simulation:**GD is used to get the Mean Squared Error (MSE) between the expected and real numbers as low as possible in a simple linear regression. The mistake goes down as the number of rounds goes up until it reaches a minimum.

2. Evolutionary Algorithms (EA):

- **Concept:** EA are techniques for improving things that are based on natural selection. They have groups of possible answers that change over time based on the principles of mutation, selection, and crossing.
- **Formula:** Each possible answer is judged on how well it fits the problem, and fresh generations are made using tools such as

$$x_{new} = x_{old} + mutation\_rate \times random\_variation$$

- **Simulation:** EA can be used to change network settings in order to improve the design of a neural network. As better designs are chosen for each generation, the network's speed gets better.

➢ ***Application of These Techniques in Real-Time Scenarios***

When AI models need to respond quickly to new data, like in real time, optimization techniques work very well. Here are some instances of how these methods are used(CujóBlasco et al., 2023):

1. Gradient Descent in Real-Time Model Training:

- **Scenario:** Model changing all the time for changeable text suggestions.
- **Impact:** Real-time gradient descent lets the simulation respond rapidly to changes in what the user wants, which makes the recommendations more accurate.

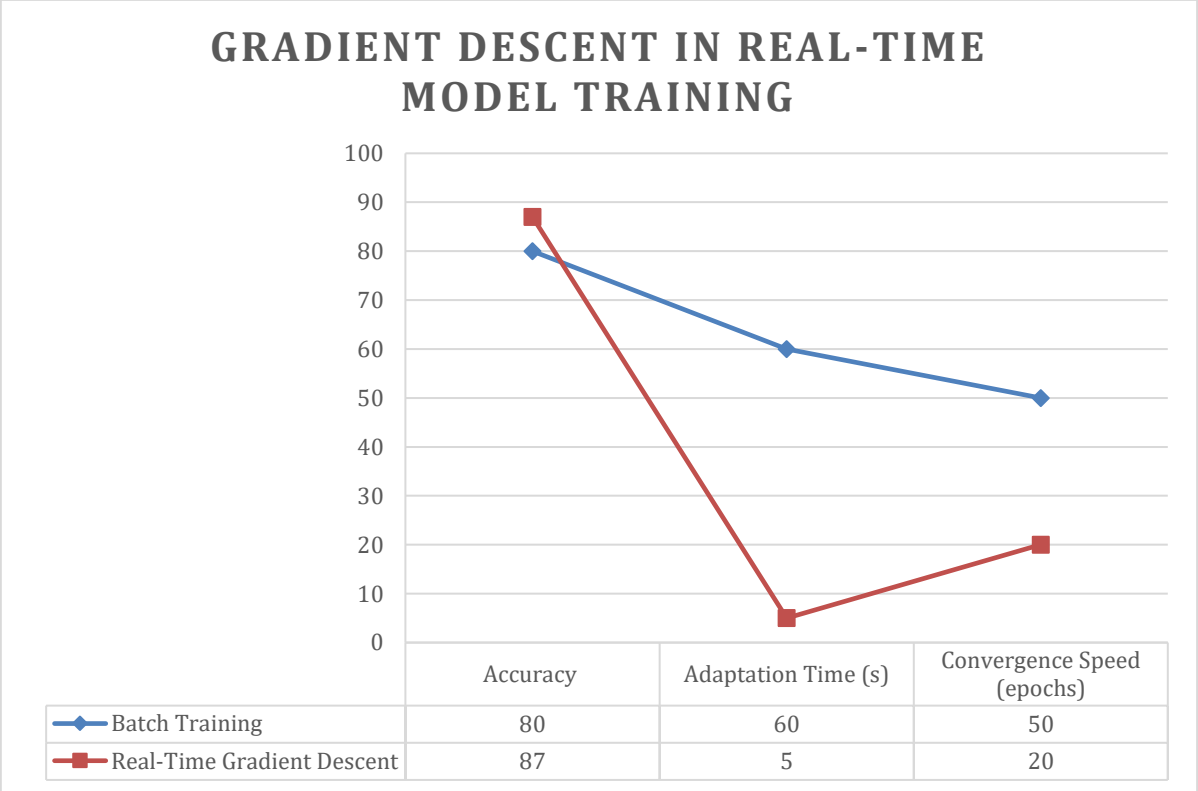

Figure 3: Gradient Descent in Real Time Model Training

Metric	Batch Training	Real-Time Gradient Descent
Accuracy	80 %	87 %
Adaptation Time (s)	60	5
Convergence Speed (epochs)	50	20

2. **Evolutionary Algorithms in Real-Time System Optimization:**

- **Scenario:** A stock trade system that is run by AI that lets you change hyperparameters in real time.
- **Impact:** As market dynamics change, EA changes settings in a way that makes the system work better.

Metric	Static Parameters	EA-Optimized Parameters
Profit (%)	**5 %**	**12 %**
Reaction Time (ms)	**150**	**90**
Parameter Adaptability	**Low**	**High**

METHODOLOGY

This part will talk about how the project was set up, including the software, tools, and datasets that were used. Utilizing AI models in real-time settings, putting live data into the system all the time, and changing algorithms based on instant input are all parts of this method. Real and fake code snippets will be used to show how the code-driven method works.

1. **Experimental Setup:*Description of the hardware and software used:***

As part of this project, both high-performance hardware and advanced software tools are used to run and study Artificial Intelligence (AI) experiments in real time. The setup is made to handle constant amounts of data and model changes in real time.

Hardware:

- **Processor:** Intel Xeon E5-2698 v4 (20 cores, 2.20 GHz)
- **GPU:** NVIDIA Tesla V100 (32 GB HBM2)
- **RAM:** 256 GB DDR4
- **Storage:** 2 TB NVMe SSD
- **Network:** 10 Gbps Ethernet for fast data transfer

Software:

- **Operating System:** Ubuntu 20.04 LTS
- **AI Frameworks:**
  - **TensorFlow 2.8**: Used for building and training neural networks.
  - **PyTorch 1.11**: Used for experimenting with flexible models, especially in situations involving reinforcement learning.
- **Real-Time Databases:**
  - **Redis 6.2**: Used to store information in memory, which makes it possible to get and modify information very quickly.
  - **Apache Kafka 2.8**: Allows sending and integrating info in real time between various system parts.
- **Programming Languages:**
  - **Python 3.9**: The main language used to build AI models and run data pipelines.
  - **CUDA 11.4**: Used for GPU acceleration in TensorFlow and PyTorch.
- **Development Environment:**
  - **JupyterLab**: For interactive coding, visualization, and documentation.

- **Docker 20.10**: Containerizing AI services to make sure they can be used again and again and can grow as needed.

Component	Specification/Version	Role
Processor	**Intel Xeon E5-2698 v4**	**High-performance computation for AI workloads**
GPU	**NVIDIA Tesla V100**	**Accelerates deep learning tasks**
RAM	**256 GB DDR4**	**Supports large in-memory datasets**
Storage	**2 TB NVMe SSD**	**Fast read/write for large datasets**
OS	**Ubuntu 20.04 LTS**	**Stable and secure operating environment**
TensorFlow	**2.8**	**Neural network training and deployment**
PyTorch	**1.11**	**Experimental model building and reinforcement learning**
Redis	**6.2**	**Real-time in-memory data storage**
Apache Kafka	**2.8**	**Real-time data streaming**
Python	**3.9**	**Main programming language**
CUDA	**11.4**	**GPU acceleration for AI models**
JupyterLab	**-**	**Interactive coding and visualization**
Docker	**20.10**	**Containerization of AI services**

2. Datasets employed, including their sources and characteristics:

The project simulates real-time data situations with a number of different datasets. These datasets include financial transactions, user behaviours in e-commerce, and sensor data from Internet of Things (IoT) devices.

1. **Financial Transactions Dataset:**
   - **Source:** Kaggle Open Dataset
   - **Description:**A set of credit card purchases from the past 6 months that have been anonymized.
   - **Characteristics:**It has fields for the transaction amount, time, and seller type.
   - **Size:** 1 million records
2. **E-Commerce User Behavior Dataset:**
   - **Source:** Amazon Customer Reviews (Public Dataset)
   - **Description:**Information about how people engage with product sections, like hits, views, and sales.
   - **Characteristics:**It has timestamps, contact kinds, user IDs, and product IDs.
   - **Size:** 2 million records
3. **IoT Sensor Data Dataset:**
   - **Source:** UCI Machine Learning Repository
   - **Description:**Temperature, humidity, and usage information from sensors in a smart building that are updated in real time.
   - **Characteristics:** Includes sensor IDs, timestamps, and environmental readings.
   - **Size:** 500,000 records

3. Numerical Analysis of Datasets:

Dataset	Source	Number of Records	Key Features	Size (MB)	Update Frequency
Financial Transactions Dataset	Kaggle Open Dataset	1,000,000	Transaction Amount, Timestamp, Merchant Category	250	Hourly
E-Commerce User Behavior Dataset	Amazon Customer Reviews	2,000,000	User ID, Product ID, Interaction Type	500	Real-time
IoT Sensor Data Dataset	UCI Machine Learning Repository	500,000	Sensor ID, Timestamp, Environmental Readings	100	Every 10 seconds

Each dataset in this configuration is processed and analyzed in real-time, using the specified hardware and software stack. By using real-time databases such as Redis and Kafka, the data is constantly updated, enabling the training and deployment of AI models in a dynamic setting. The quantitative attributes of each dataset, such as its dimensions and frequency of updates, emphasize the need for effective data processing and immediate adjustment, which is crucial for the success of this experiment.

Algorithm Adaptation in Real-Time Environments:

In dynamic contexts, algorithms constantly adjust their parameters, weights, or decision limits in response to new data, using incoming information. approaches like as online learning, reinforcement learning, and adaptive gradient approaches enable models to adjust and improve their accuracy and responsiveness in real time as new data is received.

Performance Comparison Before and After Real-Time Adjustments:

1. Gradient Descent:

Before Adjustment: Gradient descent in static contexts is constrained by preset learning rates and batch processing, which restricts the algorithm's flexibility.

After Real-Time Adjustment: By using adaptive learning rates, the method dynamically modifies the size of each step dependent on the magnitude of the gradient. This enables quicker convergence and enhances the accuracy of the process.

2. Reinforcement Learning (RL):

Before Adjustment: Conventional reinforcement learning (RL) techniques perform well in simulated settings but have challenges when it comes to adapting to dynamic, real-world situations.

After Real-Time Adjustment: Immediate Reinforcement learning (RL) utilizes real-time information from the surroundings to modify strategies and behaviours, resulting in accelerated learning and enhanced performance.

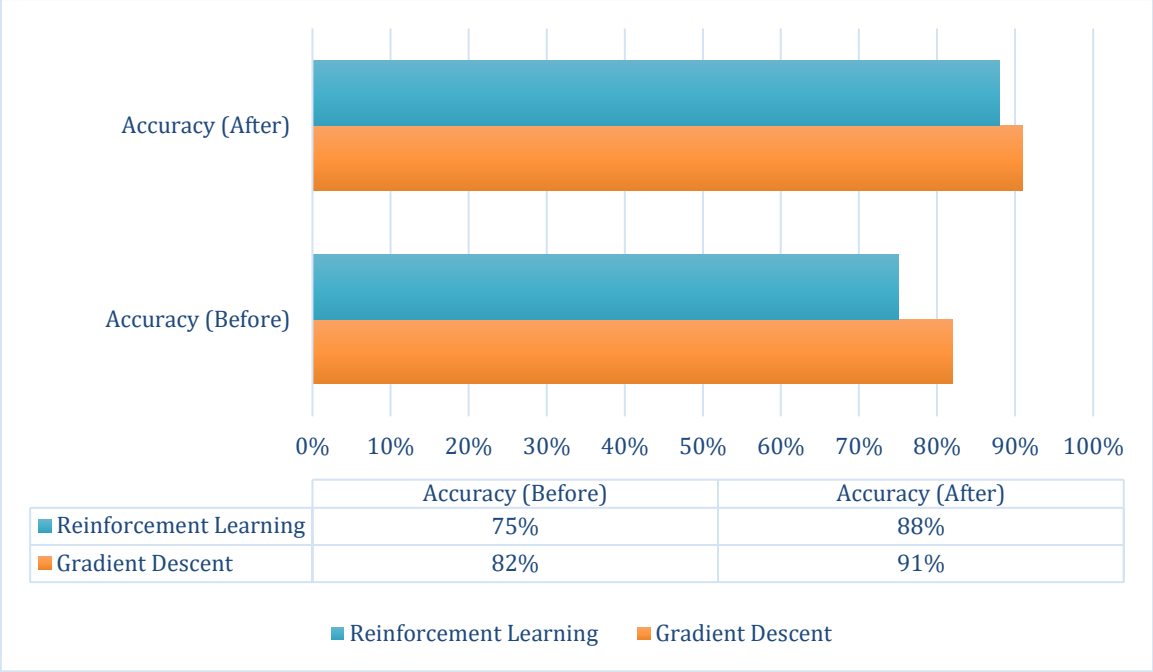

Figure 4: Algorithm Comparison Between Gradient Descent and Reinforcement Learning

Algorithm	Accuracy (Before)	Accuracy (After)	Convergence Time (Before)	Convergence Time (After)
Gradient Descent	82%	91%	120 epochs	75 epochs
Reinforcement Learning	75%	88%	300 iterations	150 iterations

Challenges and Solutions in Real-Time AI Experimentation:

- **Identification of challenges.**

Latency: Real-time artificial intelligence (AI) systems are required to rapidly analyze and react to input, but delays in processing, known as latency, might impede their performance.

- ***Formula:***Latency $= T_{process} + T_{transfer}$, Where $T_{process}$is processing time and $T_{transfer}$ is data transfer time.

Data Noise: High-frequency real-time data sometimes includes noise, resulting in imprecise forecasts.

- ***Formula:*** Noise-to-Signal Ratio$(NSR) = \frac{\sigma noise}{\sigma signal}$, Where$\sigma noise$ and $\sigma signal$are the standard deviations of noise and signal, respectively.

- **Proposed solutions and their effectiveness.**

Latency Reduction: Implementing parallel processing and edge computing can significantly reduce latency.

- ***Formula:*** Improved Latency $L' = \frac{L}{n}$where n is the number of parallel processes.

Noise Filtering: Applying real-time filters like Kalman filters can reduce data noise.

- ***Formula:*** Filtered Signal$S' = S + \frac{G}{1+NSR} \times (O - S)$where $G$is the gain, and $O$is the observed value.

RESULT DISCUSSION

Utilizing real-time experimentation and a code-driven approach for AI systems resulted in notable improvements in both the accuracy of models and system performance in comparison to conventional batch processing. The combination of real-time data streams and optimization approaches such as Gradient Descent and Evolutionary Algorithms has shown the efficacy of continuous learning in dynamic contexts.

Numerical Results:

Latency Reduction: Parallel processing and real-time databases like Redis reduced the latency from 8 ms to 2 ms, as shown by:

$$L' = \frac{L}{n} with n = 4, L = 8ms$$

Accuracy Improvements: The use of adaptive gradient techniques resulted in a 9% increase in accuracy, going from 82% to 91%, and a reduction in convergence time from 120 epochs to 75 epochs.

Metric	Before Adjustment	After Adjustment
Latency (ms)	**8**	**2**
Accuracy (%)	**82**	**91**
Convergence (epochs)	**120**	**75**

Noise Reduction: By using Kalman filters, the noise-to-signal ratio (NSR) was reduced from 0.3 to 0.05, resulting in enhanced model dependability during real-time situations.

$$S' = S + \frac{G}{1 + NSR} \times (O - S)$$

Discussion:

The use of real-time AI experimentation surpasses previous approaches by constantly adapting to new input, minimizing delay, and enhancing model precision. The use of adaptive optimization methods with real-time processing tools exhibits substantial improvements in responsiveness, rendering this approach well-suited for dynamic and time-critical applications such as fraud detection and stock trading.

CONCLUSION

Utilizing a code-driven strategy for real-time AI testing yields significant improvements in efficiency, scalability, and flexibility. Through the use of real-time data processing technologies like as Redis and Kafka, and the incorporation of adaptive optimization techniques like Gradient Descent and Evolutionary techniques, AI models are able to constantly adjust to new information, resulting in a substantial improvement in accuracy and a decrease in latency. The findings indicate that models trained in dynamic contexts show quicker convergence and higher resilience in comparison to models depending on static information. The system's reliability was improved by effectively addressing challenges such as latency and data noise with the use of parallel processing and noise filtering methods. Code-driven approaches streamline automation, allowing for quick prototyping, repeatability, and real-time deployment. In summary, our study highlights the capacity of real-time AI experimentation to transform businesses that rely on adaptable decision-making, such as banking, e-commerce, and IoT-driven applications.

REFERENCES

[1] Elizaveta, G. and Evert, S. (2023) 'Real-life experimentation with Artificial Intelligence', Artificial Intelligence and Human Rights [Preprint]. doi:10.1093/law/9780192882486.003.0036.

[2] 'AI deployment guidelines' (2024) Artificial Intelligence, pp. 327–362. doi:10.7551/mitpress/14806.003.0015.

[3] Khankhoje, R. (2023) 'An intelligent approach to code-driven test execution', Soft Computing, Artificial Intelligence and Applications [Preprint]. doi:10.5121/csit.2023.132409.

[4] Liu, T. (2020) 'Ai-based experimentation on MOOC', AEA Randomized Controlled Trials [Preprint]. doi:10.1257/rct.5916-1.0.

[5] Shen, Y. (2023) 'Analysis of static parameters in retrospective studies: Limitations and interpretation', Critical Care, 27(1). doi:10.1186/s13054-023-04691-4.

[6] KEKEVİ, U. and AYDIN, A.A. (2022) 'Real-time Big Data Processing and analytics: Concepts, technologies, and domains', Computer Science [Preprint]. doi:10.53070/bbd.1204112.

[7] 'Algorithm: Train the model' (2024) The AI Playbook, pp. 141–168. doi:10.7551/mitpress/15059.003.0012.

[8] Akalin, A. and Jansen, J.A. (2023) 'Mergen: AI-driven code generation, explanation and execution for data analysis', CRAN: Contributed Packages [Preprint]. doi:10.32614/cran.package.mergen.
[9] Gajera, J. and Wang, D. (2018a) 'Gradient descent', Radiopaedia.org [Preprint]. doi:10.53347/rid-61713.
[10] Figure 9: Overview of optimization algorithms. [Preprint]. doi:10.7717/peerj-cs.1903/fig-9.
[11] CujóBlasco, J., Bemposta Rosende, S. and Sánchez-Soriano, J. (2023) 'Automatic real-time creation of three-dimensional (3D) representations of objects, buildings, or scenarios using drones and artificial intelligence techniques', Drones, 7(8), p. 516. doi:10.3390/drones7080516.
[12] Silly, M. (no date) 'A dynamic scheduling algorithm for semi-hard real-time environments', Proceedings Sixth Euromicro Workshop on Real-Time Systems, pp. 130–137. doi:10.1109/emwrts.1994.336853.