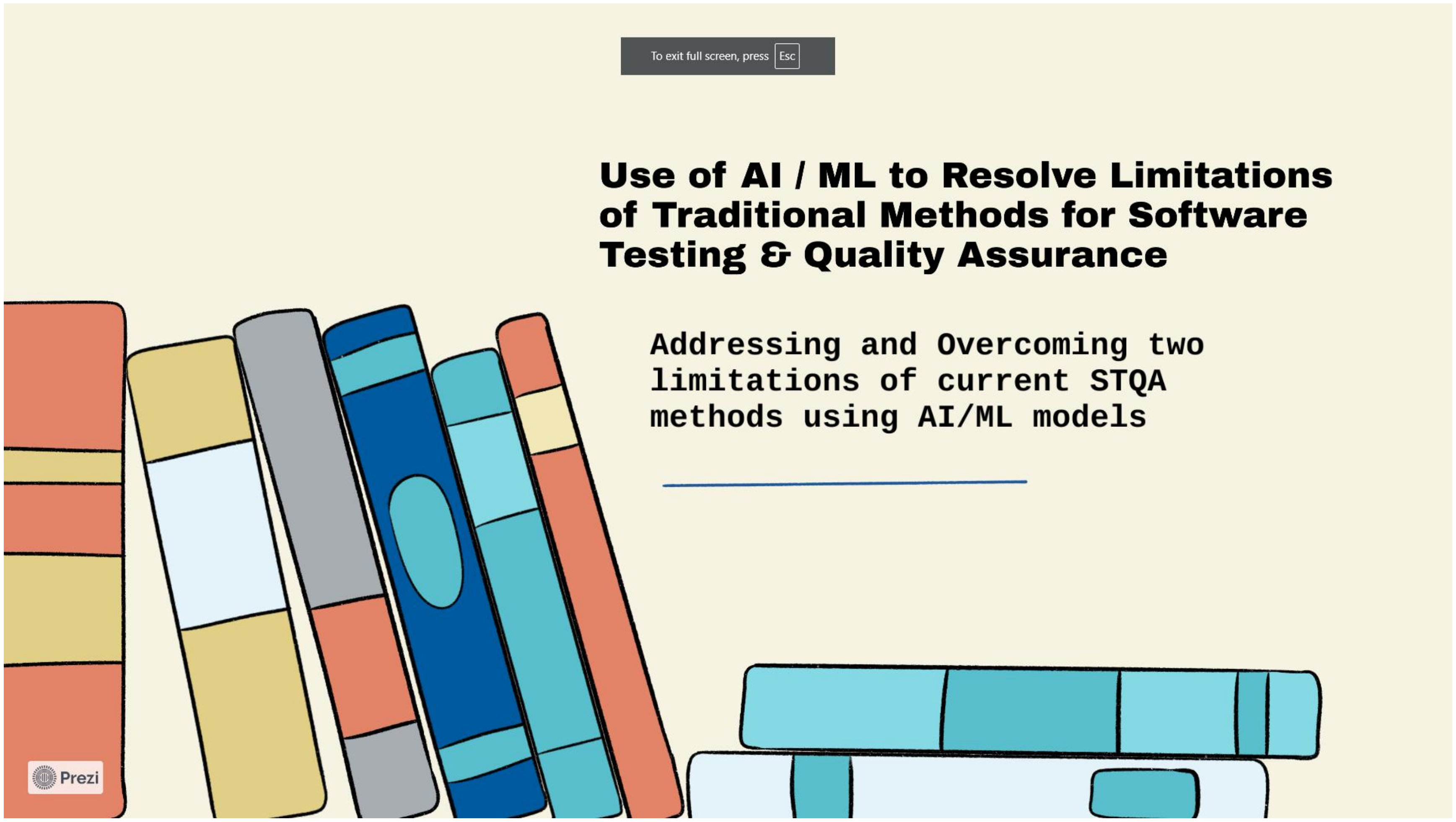


Use of AI / ML to Resolve Limitations of Traditional Methods for Software Testing & Quality Assurance

Addressing and Overcoming two limitations of current STQA methods using AI/ML models

Introduction to AI/ML in Software Testing

1- Ensuring reliable and robust software

2- Traditional testing methods: manual, automated

3- Scalability and maintenance challenges faced

4- AI/ML integration to overcome limitations

5- Enhanced testing efficiency and effectiveness

Motivation

1- Limitations of traditional testing

2- Increasing software complexity

3- Need for scalable solutions

Aims and Objectives

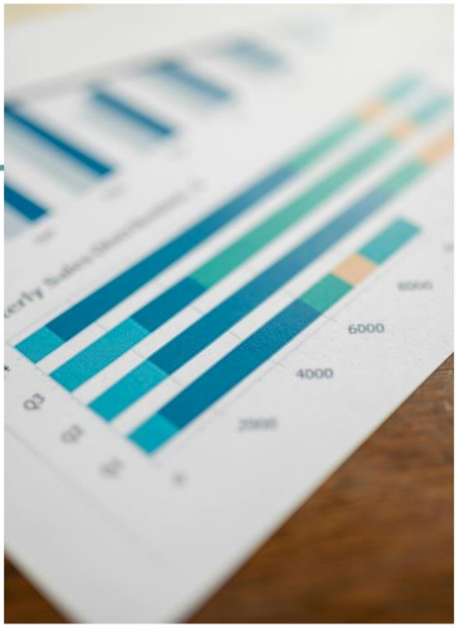
1- Analyze traditional testing limitations

2- Explore AI/ML techniques for testing

3- Implement scalable, maintainable solutions

Background

- Traditional testing methods face scalability issues
- Increasing complexity of modern software systems
- AI/ML offers advanced testing techniques
- Improved defect prediction and test prioritization



Prezi

Related work

1- ML techniques in ST&QA

2- AI in ST&QA

3- Test Case Prioritization

4- Random forest, GBM, BERT & KNN

Limitations

Both traditional and AI methods for software testing and quality assurance face several inherent limitations:

Scalability Issues

Data Requirements

High Maintenance Overhead

Adaptability

Traditional Methods

These are the traditional methods we use in the current software

- Manual
- Automated
- Regression
- Acceptance
- Exploratory

AI Methods

These are the AI approaches for STQA

Automated test case generation

Defect Prediction

Automated Regression Testing

Test Case Prioritization

Combinatorial Testing

Prezi

Scalability Issue

Traditional STQA methods face scalability issues due to **increased execution times** and **resource demands** when handling **large test** volumes without efficient parallelization

Increased Maintenance overhead

Maintenance overhead in traditional STQA methods is the effort and complexity involved in managing and updating a **growing number of test cases** and configurations

JUnit

a popular Java testing framework for creating and running unit tests.

Types of analysis

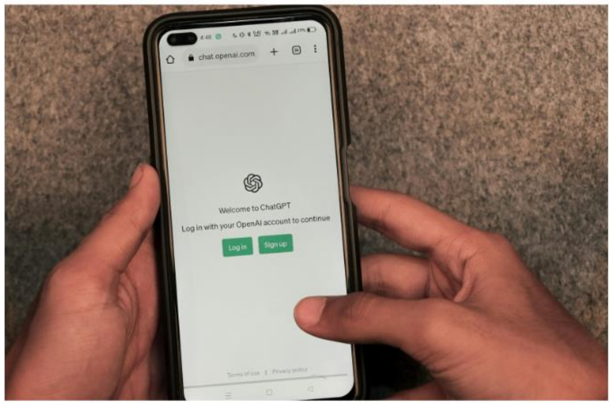
Theoretical Analysis   Empirical Evidence   Practical Examples

Solution

To address scalability issues and maintenance overhead in JUnit, **parallel test execution** and **test optimization or Prioritization.** can improve efficiency. **Enhanced test organization** and **automation** can reduce manual effort and simplify maintenance.

Implementation and Evaluation

This section details the methodology behind the integration of AI and ML in software testing, focused on the phases of data preparation, model training, results analysis, and performance evaluation.



RANDOM Forest

Random Forest is an **ensemble learning method** that aggregates **multiple decision trees** to improve **prediction accuracy** and **reduce overfitting**.

Random Forest For test case Prioritization

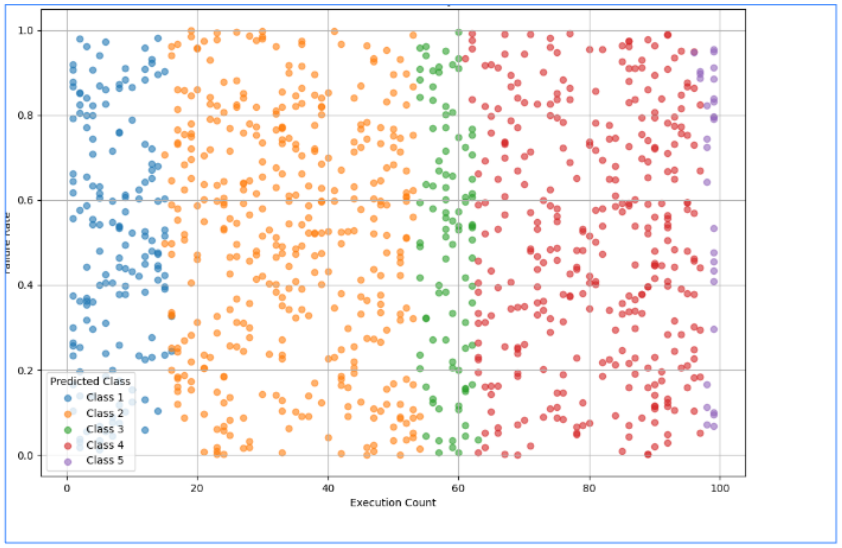
Used for Test Case Prioritization

Synthetic Dataset

Feature Selection

Model Tuning

Prioritization Mechanism



GBM, NN and XGBoost

Gradient Boosting Machines (GBM): An **ensemble learning** method that **builds models sequentially**, where each model **corrects the errors of its predecessor** to enhance **prediction accuracy**.

Neural Networks (NN): A **computational model** inspired by the human brain, consisting of **interconnected nodes** (neurons) that learn to make **predictions** by **adjusting weights** based on input data.

XGBoost (**Extreme** Gradient Boosting): An **optimized** version of gradient boosting that uses **advanced techniques** to improve performance, including **regularization** and **parallel processing**, to enhance **prediction accuracy** and efficiency.

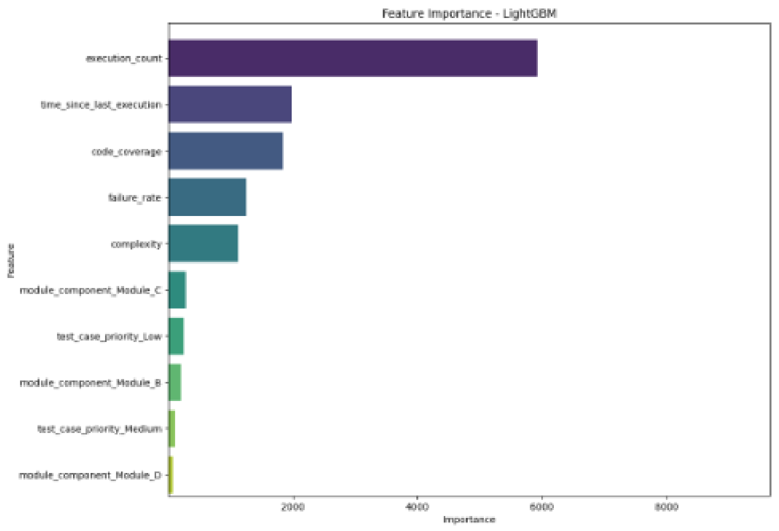
GBM for test case Prioritization and feature Importance

GBM, NN, and XGBoost enhance test case prioritization with their advanced predictive capabilities.

GBM: Iteratively refines predictions, uses features like defect severity, and updates rankings dynamically.

NN: Uses deep learning for complex patterns, applies SMOTE for class balance, and tracks training performance.

XGBoost: Optimizes gradient boosting with efficient training and hyperparameter tuning, and is compared with other models.



BERT

BERT (Bidirectional Encoder Representations from Transformers) is a **deep learning model for natural language understanding** that **pre-trains on vast text data** to capture context and meaning from both directions in a sentence.

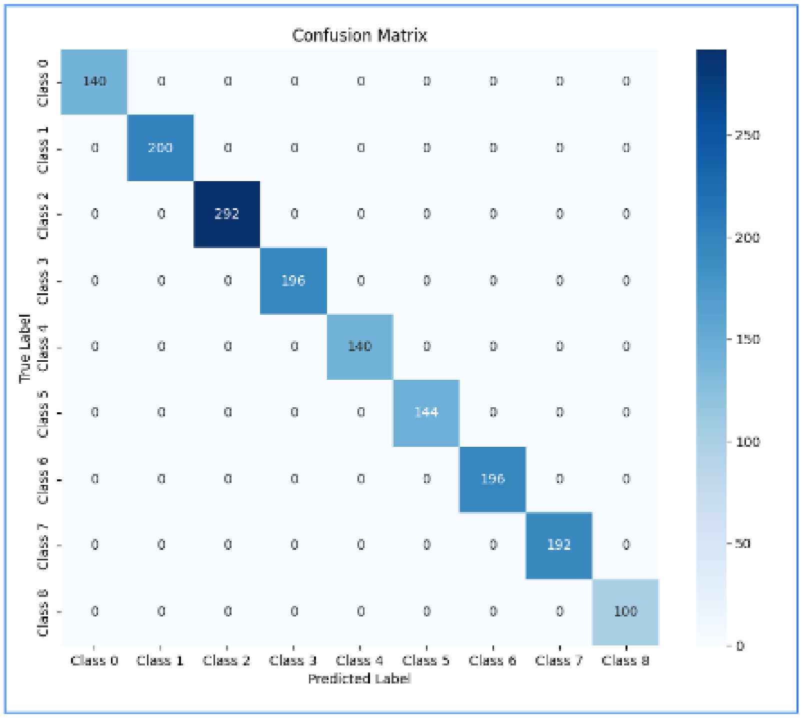
BERT based model for Defect prediction

Applied BERT for Defect Prediction

Automated Test Case Prioritization

Proved Improved Accuracy

Addressed Class Imbalance



KNN

K-Nearest Neighbors (KNN) is a machine learning algorithm used to **classify software** test cases and **predict defects** by analyzing the **similarity** of new data to existing labeled examples.

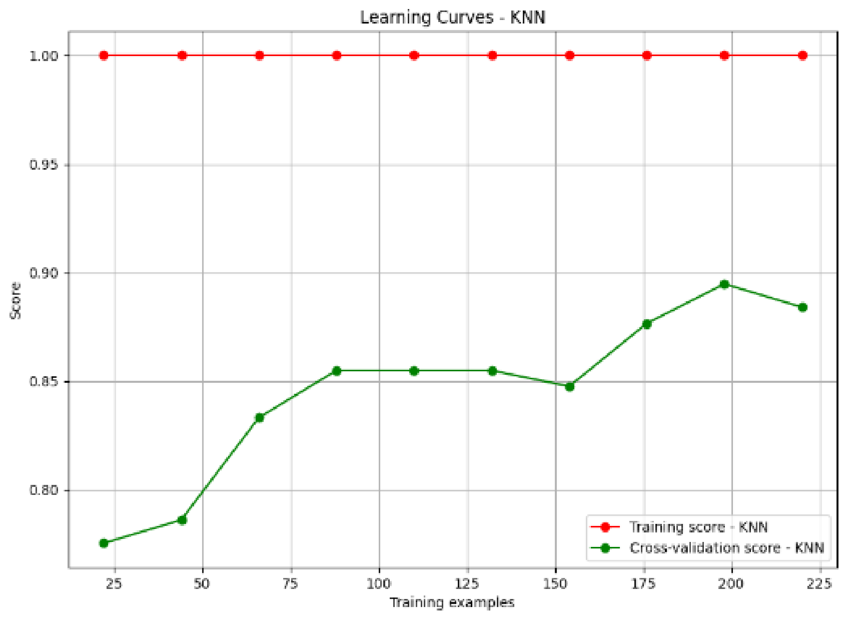
KNN for Classification and Defect Prediction

Generated and Preprocessed Data

Trained and Tuned KNN Model

Evaluated Model Performance

Enhanced Testing Efficiency



Conclusion

1- Identified Limitations in Traditional Software Testing:

Scalability Issues

Maintenance Overhead

2- Applied AI/ML Models to Overcome Limitations

Random Forest and GBM

BERT for Defect Prediction

KNN for Classification

3- Enhanced Scalability using Random Forest and GBM

4- Reduced Maintenance Overhead using BERT and KNN

Comparison with related work

1- Traditional vs. AI-Based Testing:

- Traditional methods (e.g., SVM Rank) lack scalability.
- Our AI models (Random Forest, GBM) address scalability more effectively.

2- Test Case Prioritization with ML:

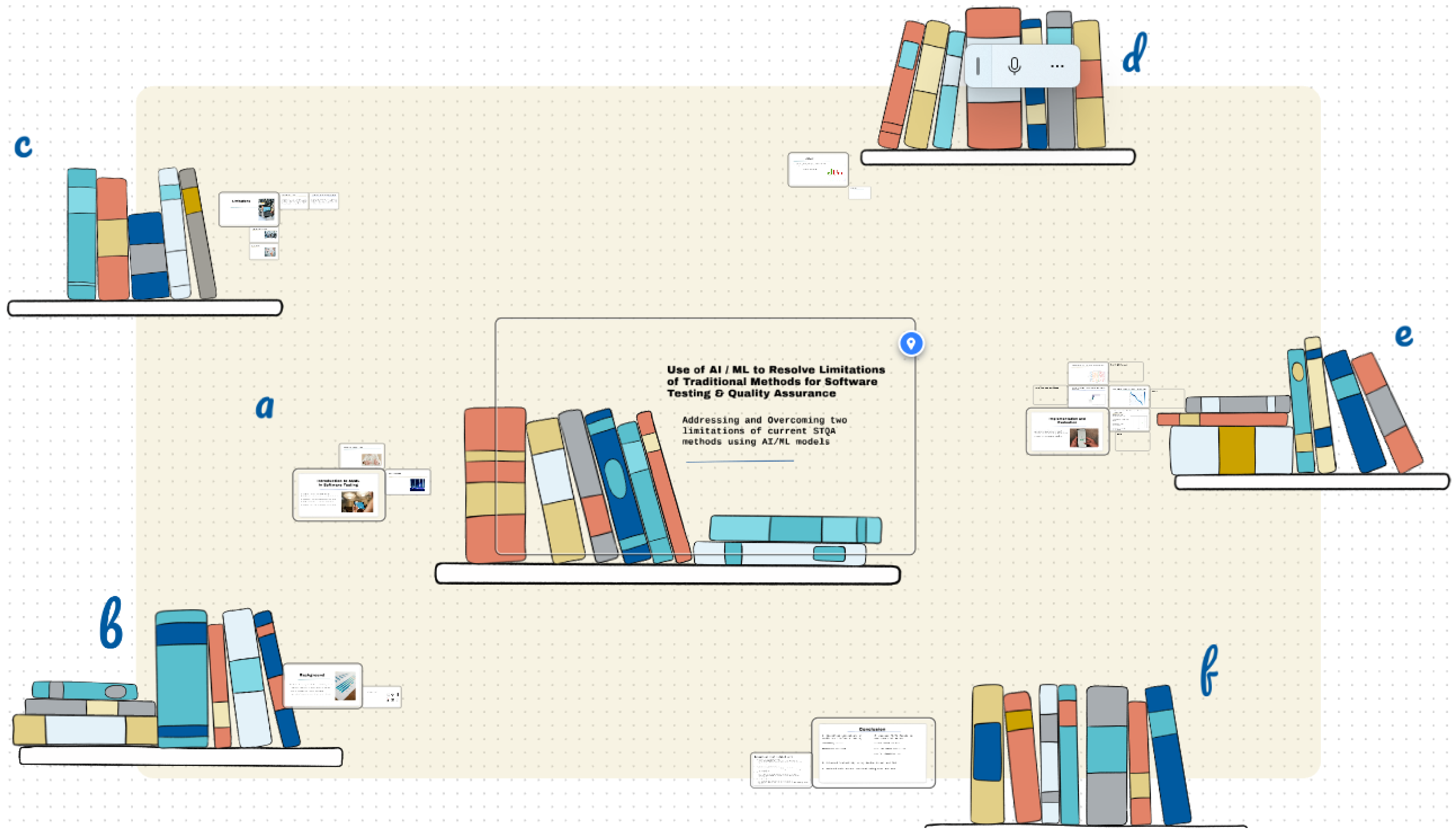
- Previous work faced issues with data complexity.
- Our Random Forest and GBM handle larger datasets and complex features better.

3- Defect Prediction:

- BERT's use in defect prediction is novel in testing.
- Superior to classical ML models in understanding defect descriptions.

4- KNN for Classification:

- Previous KNN use was limited to small datasets.
- Our approach scales KNN, improving adaptability and reducing manual updates.



Use of AI / ML to Resolve Limitations of Traditional Methods for Software Testing & Quality Assurance

```
Addressing and Overcoming two
limitations of current STQA
methods using AI/ML models
```