# Overview

The rapid rise of large language models (LLMs) such as ChatGPT has enabled automated code generation. While these tools increase developer productivity, they raise new challenges in **software security, plagiarism detection, and academic integrity**.

This project explores **machine learning techniques** for distinguishing **human-composed** and **AI-generated Java programs** using **static code metrics**.

---

# Project Workflow

## 1. Data Collection

- **Human-written dataset**:
  - 100 Java/Android projects were collected from GitHub.
  - A Python script extracted only `.java` source files from each project.
- **AI-generated dataset**:
  - 100 Java/Android projects were generated using the OpenAI API.
  - Each program was created via a two-step prompt:
    1. Generate a unique Android project idea.
    2. Generate a complete Java implementation (all classes in a single file).
  - Files were cleaned to retain only **code and comments** (whitespace handled consistently).

---

## 2. Feature Extraction

A Python program processed all `.java` files and extracted quantitative features:

- **Code Percentage** – ratio of executable code lines.
- **Comment Percentage** – ratio of comments (single + multi-line).
- **Whitespace Percentage** – ratio of blank/whitespace-only lines.
- **Average Variable Name Length** – mean length of variable identifiers.
- **Average Method Name Length** – mean length of method identifiers.
- **Average Lines per Method** – structural complexity indicator.
- **Number of Imports** – count of `import` statements.
- **Cyclomatic Complexity** – computed with the **Lizard** library:
  - *Total complexity* (sum across all methods).
  - *Average complexity* (per method).

---

## 3. Machine Learning Models

The combined dataset was labeled:

- **0 → Human-written**
- **1 → AI-generated**

Train/test split: **80% training, 20% testing**

Models evaluated:

- Support Vector Machine (SVM)
- Naïve Bayes
- Decision Tree
- Random Forest

Feature importance was computed using **permutation importance** from `sklearn.inspection`.

---

## 4. Evaluation Metrics

Models were evaluated using:

- **Accuracy**
- **Precision**
- **Recall**
- **F1-score**

Confusion matrices were generated for each model.

---

## 5. Results

- **SVM** achieved the best performance (92.5% accuracy after feature engineering).
- Naïve Bayes and Decision Tree models showed stable performance regardless of feature engineering.
- Random Forest improved slightly but lagged behind SVM.

Key finding: **Feature engineering significantly boosted SVM performance**, particularly improving recall for AI-generated code and precision for human-written code.