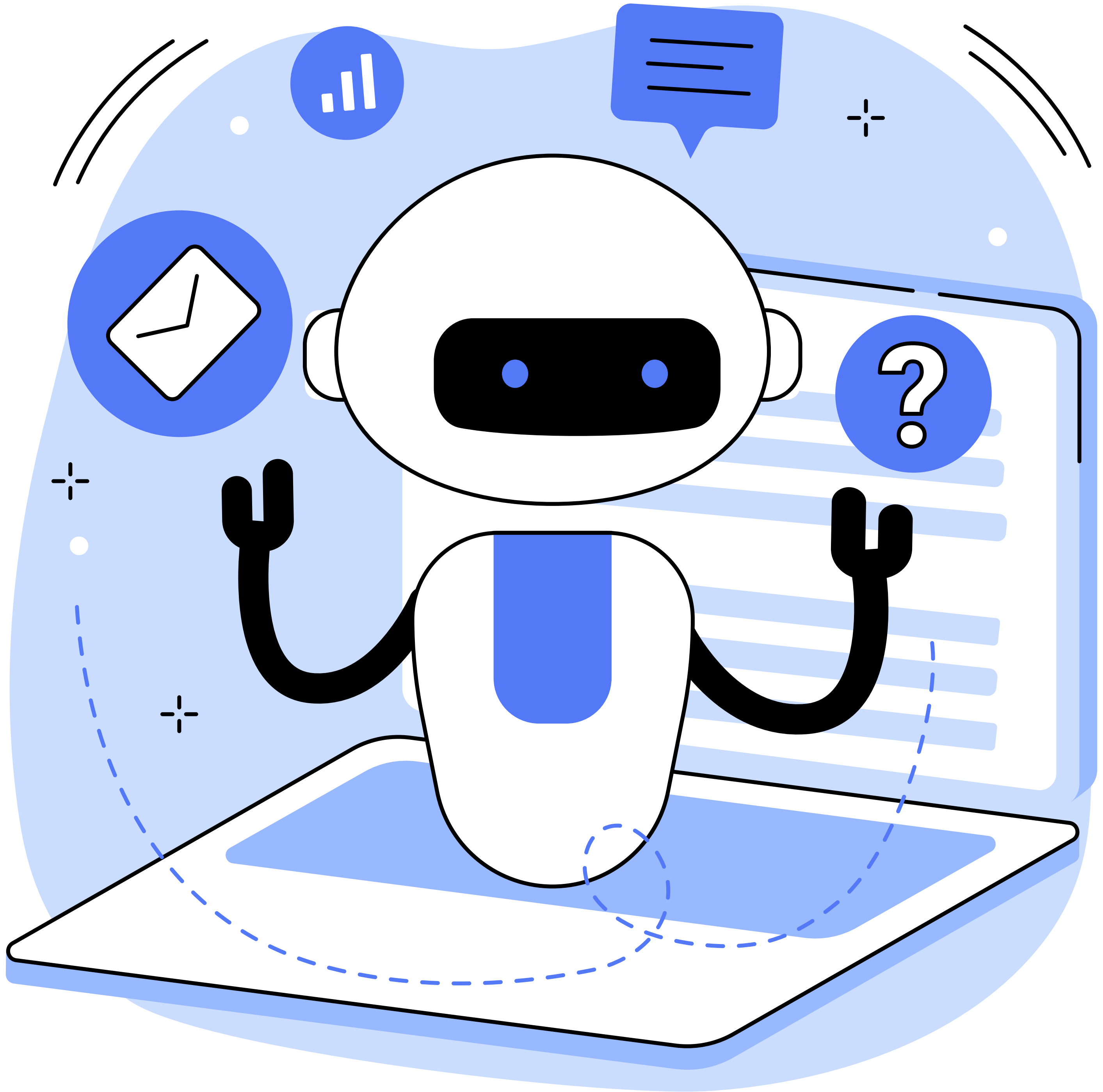


BUILDING A GENERATIVE AI-POWERED CODE MIGRATION PIPELINE FOR APPLICATION MODERNISATION

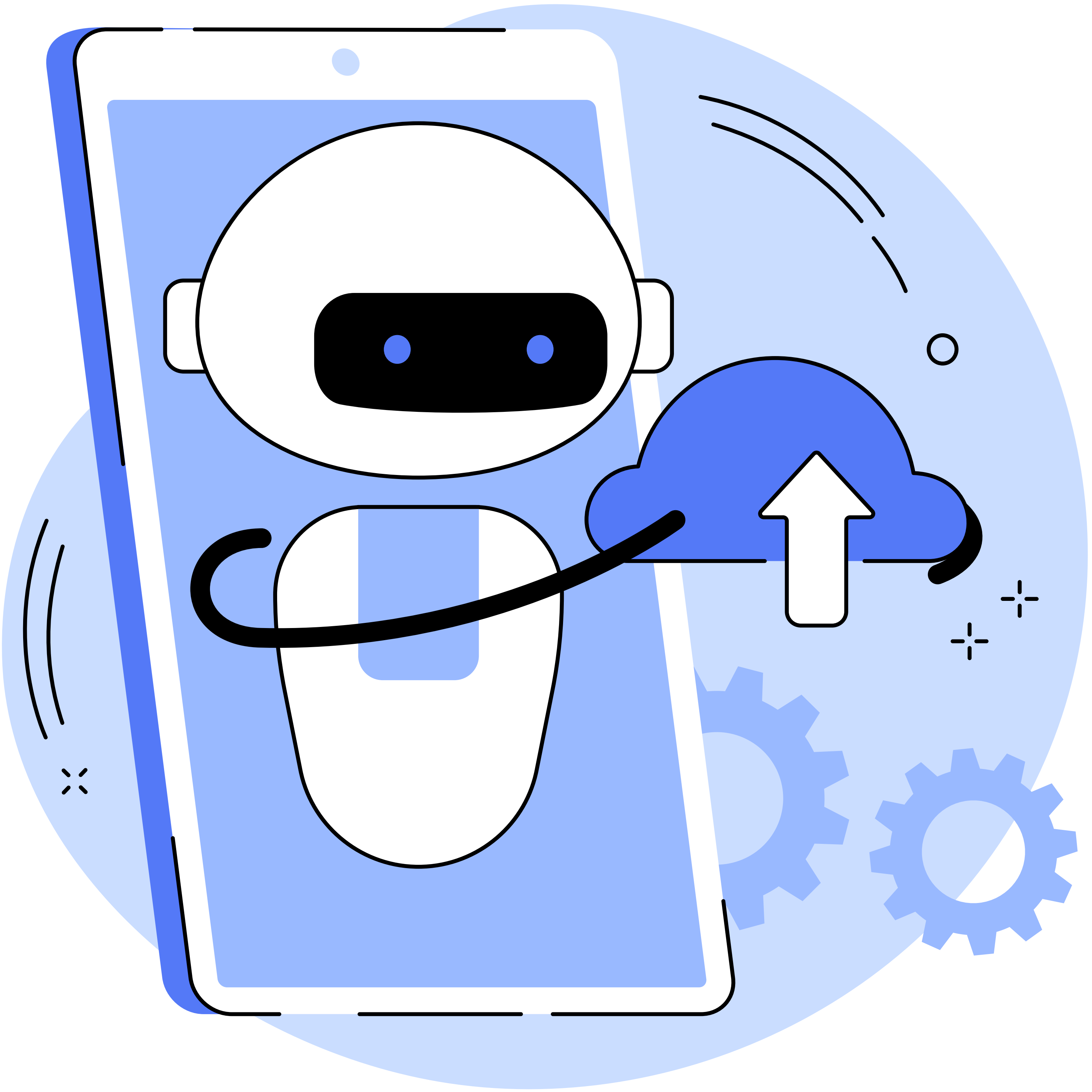
Emma Roche - 20088680

INTRODUCTION

- Explored the use of **generative artificial intelligence (AI)** in facilitating and improving the **code migration process**

- Developed a **generative AI-powered code migration pipeline**

- Explored different **prompt engineering techniques** to compare their impact on the migration process



RESEARCH QUESTIONS

1. How does the contemporary landscape of generative AI contribute to the facilitation of code migration?

2. How can the quality and correctness of code migration using generative AI be assessed?

3. How can the dissertation's insights provide practical recommendations for code migration using generative AI?

SCOPE

- Use of Pre-Trained Models
- Migration of Class and Application Level Code
- Migration from Java to Kotlin and JavaScript to TypeScript
- Comparison of Two Different Prompt Styles
- Testing (with pre-developed tests) and Analysis of Migrated Code

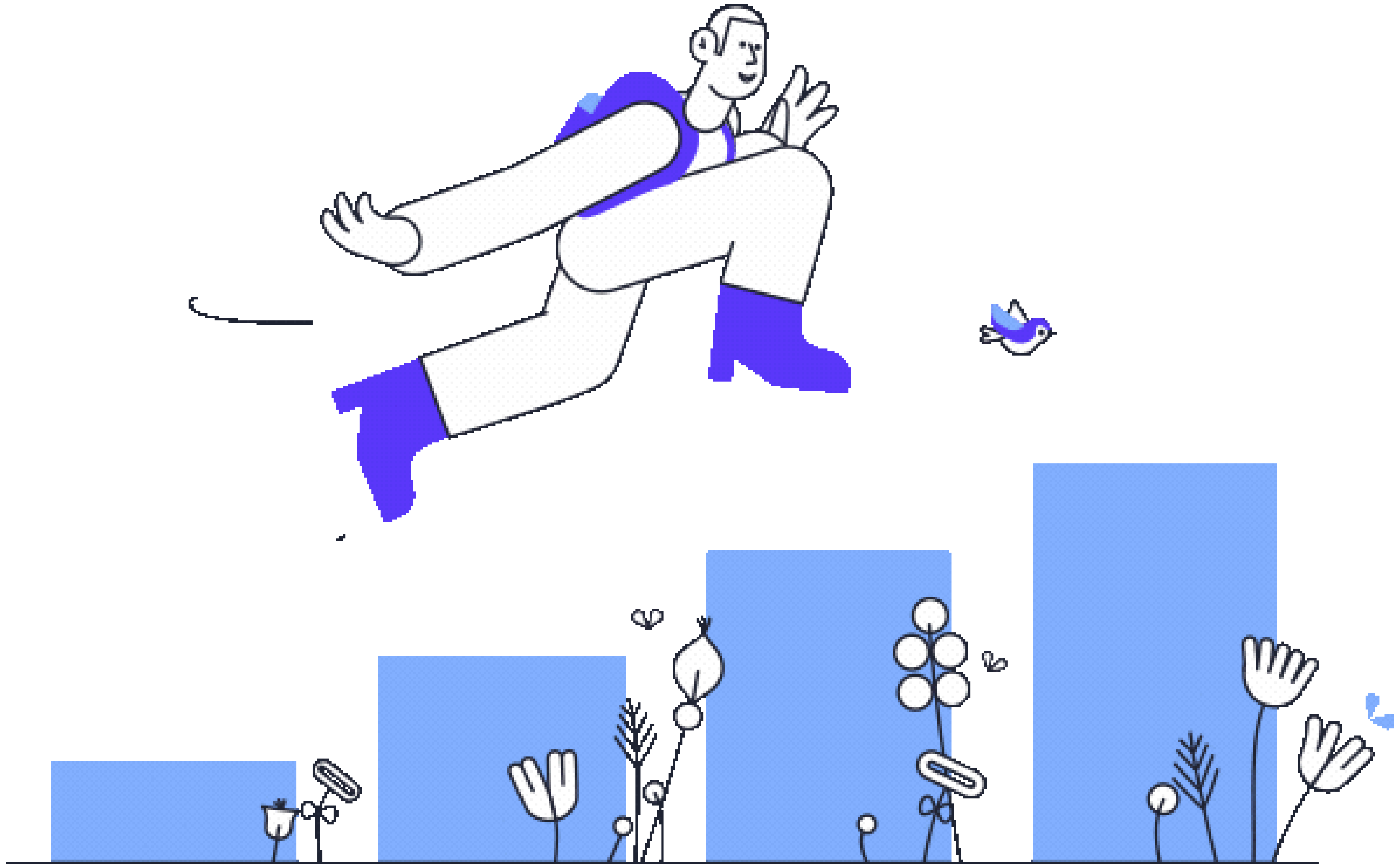
LIMITATIONS

- Time Constraints
- Computational Limitations
- AI Model Limitations
- Programming Language Pairs
- Code Artefacts Complexity

METHODOLOGY OVERVIEW

- **Exploratory:** explored the use of **different AI models** & tested the effects of the two **different prompt styles** on migration outcomes

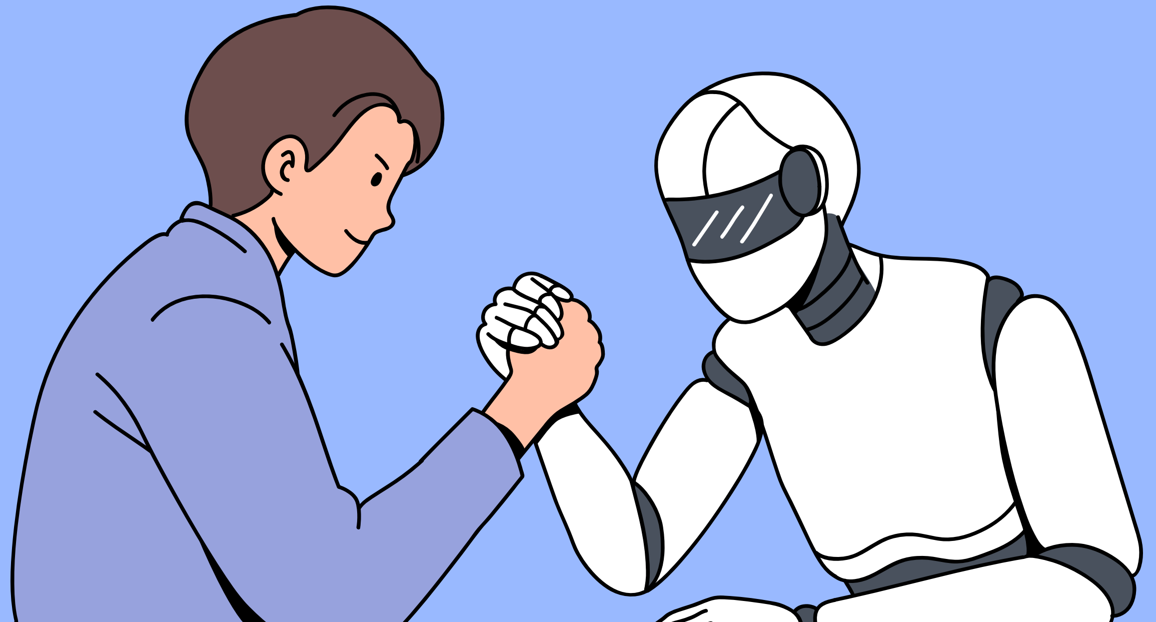
- **Experimental:** key **quantitative** metrics were measured, such as **migration speeds, test pass/fail rates**, and **static analysis** results



GENERATIVE AI MODEL SELECTION

The list below outlines the criteria followed for selecting models:

1. Model compatibility with programming languages
2. Model compatibility with computer used for study
3. Model costs



SELECTED AI MODELS

The models were incorporated into the pipeline through LangChain modules from the following providers:

OpenAI	Vertex AI	ollama
GPT-3.5 Turbo by OpenAI	Gemini Pro by Google	Llama 3 by Meta
GPT-4 Turbo by OpenAI	PaLM2 by Google	CodeLlama by Meta
GPT-4o by OpenAI	Codey by Google	CodeGemma by Google

PROGRAMMING LANGUAGES SELECTED

Java to Kotlin

- Java and Kotlin are most **commonly used** for **Android** app **development**
- **Kotlin** was developed with a more **concise** and **readable syntax** than Java
- **Kotlin** has become the **preferred language** for **Android** development

JavaScript to TypeScript

- JavaScript and TypeScript are most **commonly used** for **web** app **development**
- **TypeScript** is a **superset** of JavaScript that includes **additional features**
- **Migrating** from JavaScript **to TypeScript** is becoming more of a **common task**

PROMPT ENGINEERING TECHNIQUES SELECTED

Zero-shot Prompting

- **Zero-shot prompting** directly instructs the model without examples or illustrations
- **Leverages** the model's **broad training** to handle various code tasks with **minimal input**
- **Ideal** for producing **faster** outputs in situations where **time** is a **constraint**

Instruction Prompting

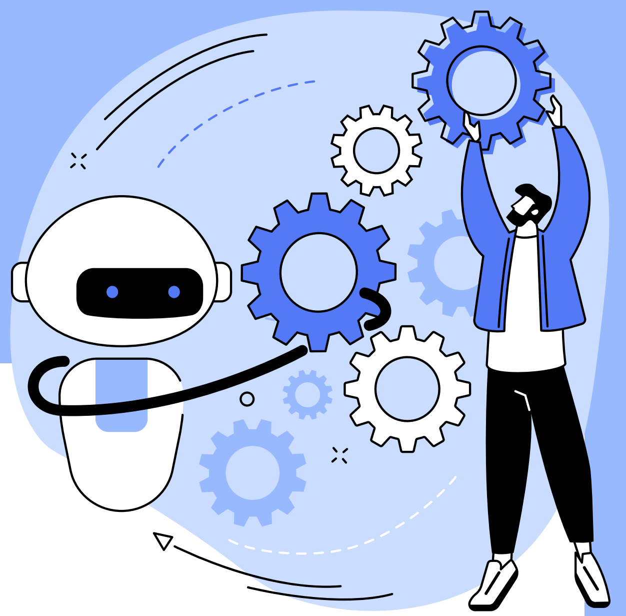
- **Instruction prompting** provides detailed instruction and guidance to an AI model
- Helps the models **better understand** the intended **task**
- **Detailed** prompts can **improve** the models output **quality** and **accuracy**

PIPELINE DESIGN OVERVIEW

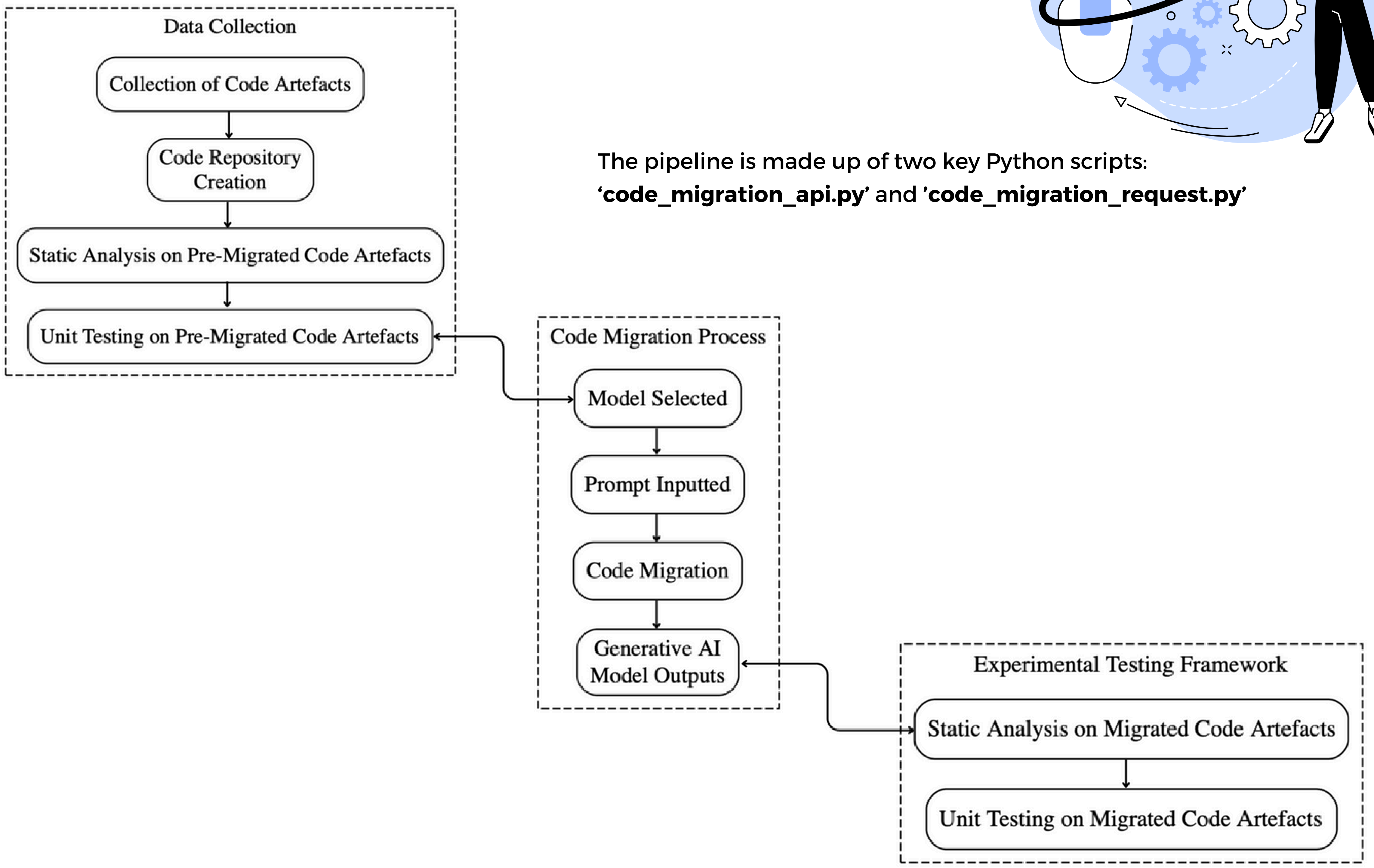
The purpose of this code migration pipeline was to create a **repeatable** and **automated** process that aims to **reduce** as much **time** and **manual intervention** as possible.

The pipeline includes three main steps:

1. Data Collection
2. Code Migration
3. Experimental Testing

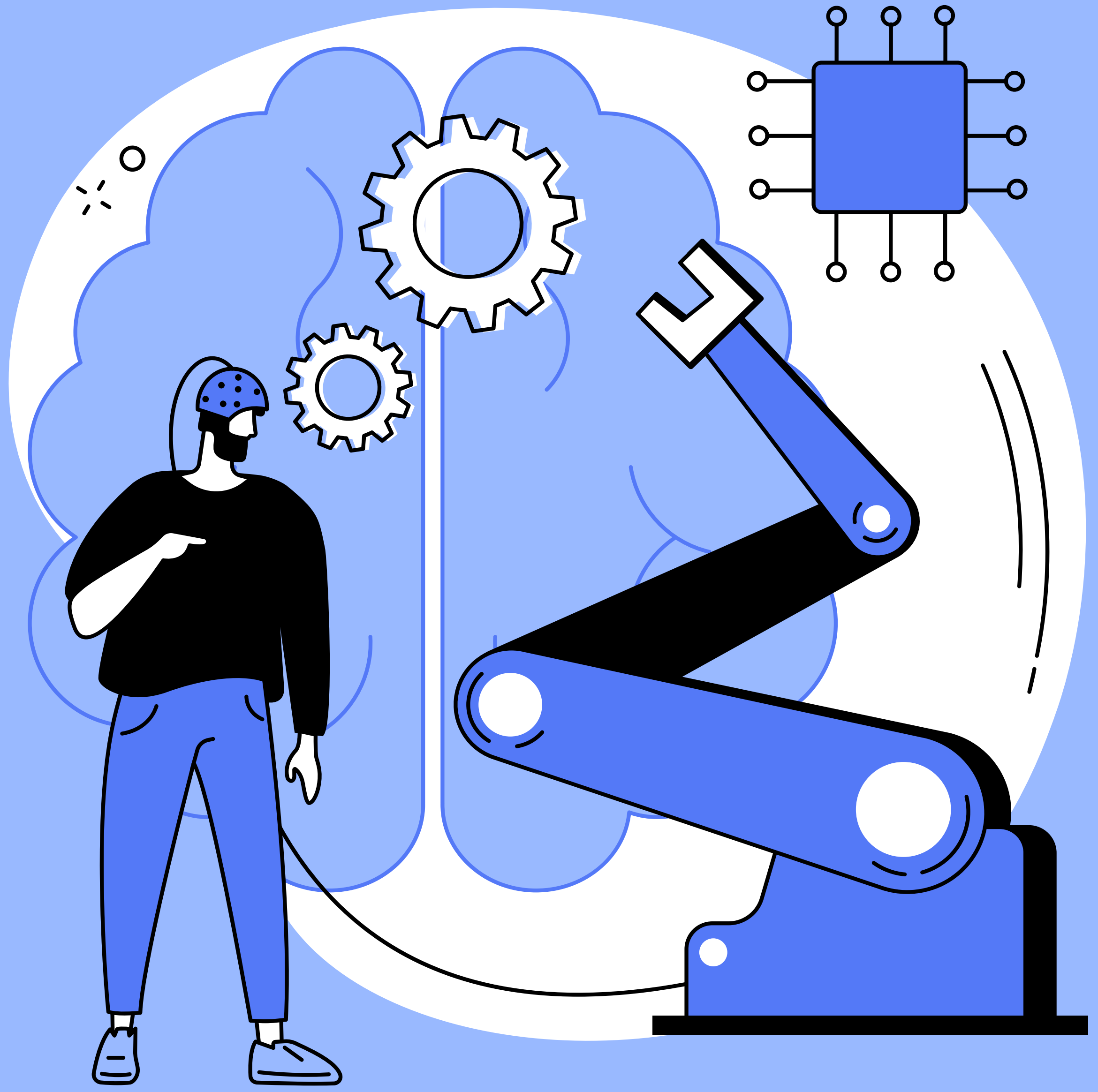


The pipeline is made up of two key Python scripts: ‘**code_migration_api.py**’ and ’**code_migration_request.py**’


Data Collection
Collection of Code Artefacts
Code Repository Creation
Static Analysis on Pre-Migrated Code Artefacts
Unit Testing on Pre-Migrated Code Artefacts
Code Migration Process
Model Selected
Prompt Inputted
Code Migration
Generative AI Model Outputs
Experimental Testing Framework
Static Analysis on Migrated Code Artefacts
Unit Testing on Migrated Code Artefacts

TOOLS & TECHNOLOGIES INCORPORATED

1. LangChain
2. Flask
3. SonarQube
4. Gradle & JUnit 5
5. Node Package Manager & Jest



SUMMARY OF KEY RESULTS

Migration Process Efficiency

VertexAI models completed migrations **fastest**, while **Ollama** models were the **slowest. Zero-shot prompts** were slightly **more efficient** than instructional prompts

Static Analysis Results

Zero-shot prompts led to **fewer** code **quality issues**, with **GPT-4 Turbo** producing the **least** issues, while **Llama 3** and **Gemini Pro** had the **most**

Unit Testing Results

Instructional prompts were more **successful** in **producing code** that **passed** test suites. **VertexAI models** and **GPT-4o** achieved the **highest** overall **success rates**

Manual Intervention Results

Due to **low pass** rates on the **application**-level **migrations**, manual intervention assessments were conducted. **Manual corrections** significantly **improved** the test **success rates**

TO CONCLUDE

Generative AI significantly **improves the efficiency** of code migration between programming languages, but **human oversight remains essential** to ensure accuracy and quality

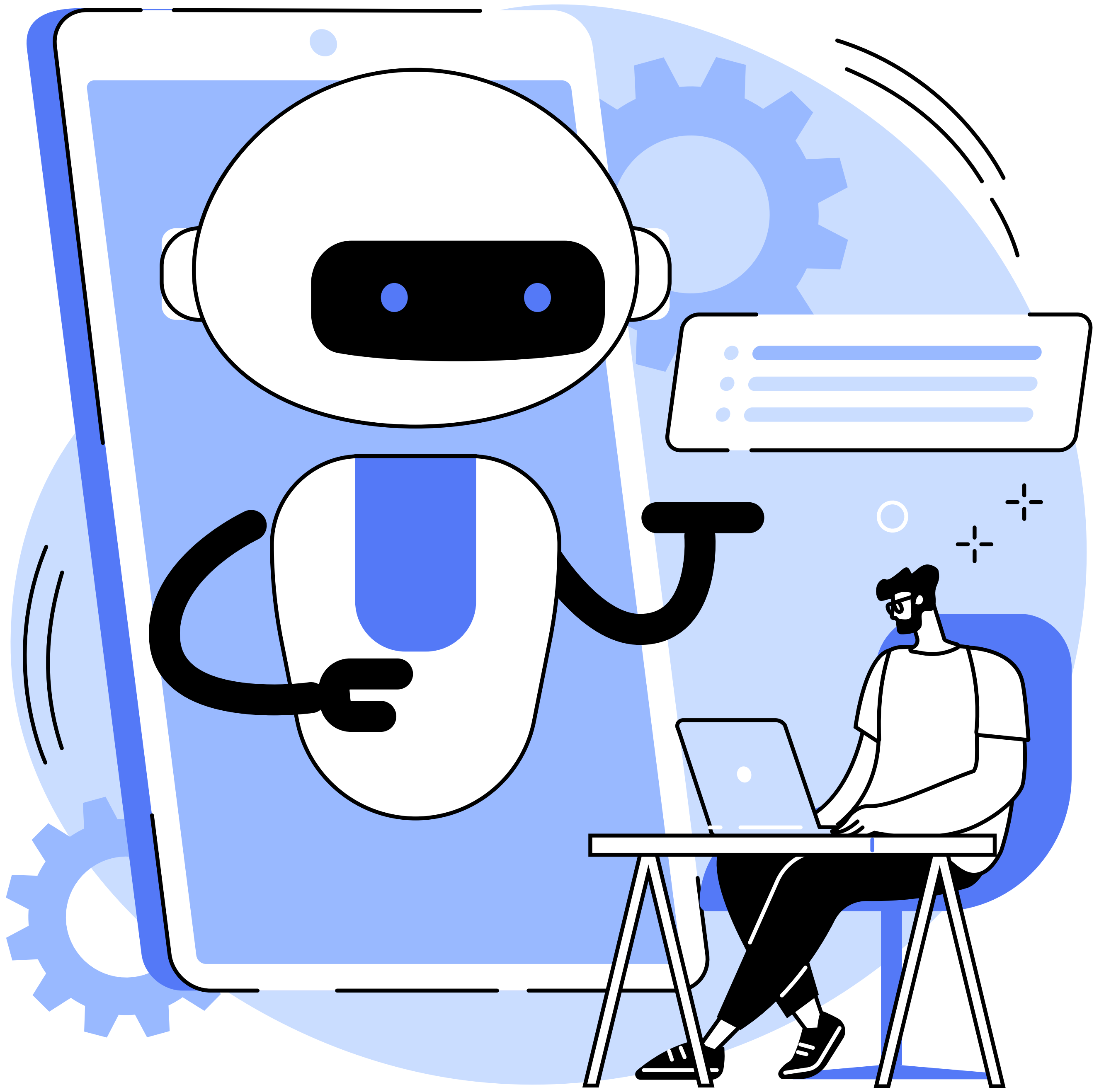
PRACTICAL RECOMMENDATIONS

1. Have experienced developers on your team that can review and correct any errors that the AI may produce

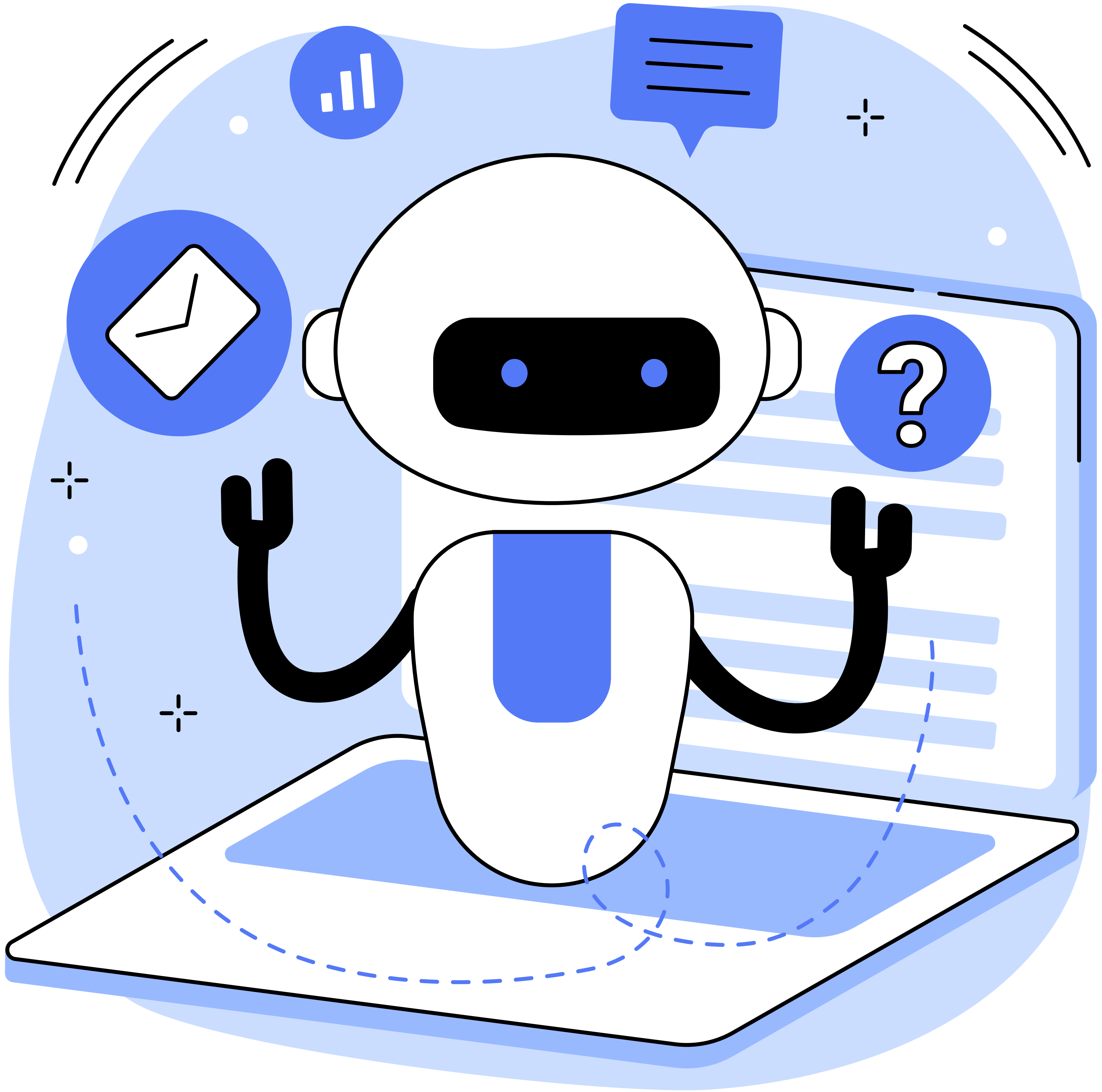
2. Run pre-migrated code through static analysis tools to ensure high code quality, as AI alone may not address all existing issues, OR craft prompt to instruct it to do so

3. Use zero-shot prompts for faster migration times and instructional prompts for higher accuracy

4. Opt for AI models with extensive training data and high parameter counts for optimal migration performance



DEMO TIME!



THANK YOU FOR LISTENING!