Initial Project Submission - Phase 1

Learning a Code Language Model using Self-Supervised Learning (SSL) for Code

Team Number - 22

Team Members with Roll Numbers

1. **Member 1**: Nikhil Singh, 2024201067
2. **Member 2**: Rajath Gadagkar, 2024201035
3. **Member 3**: Abhradeep Das, 2024202018
4. **Member 4**: Sanket Adlak, 2024204005

Scope

This project aims to develop an intelligent code completion system using Self-Supervised Learning (SSL). A language model, trained on a vast code corpus, will predict the next word or segment of code to provide accurate, real-time code suggestions. The system will support multiple programming languages and seamlessly integrate into an IDE, enhancing the coding experience with efficient and context-aware code completions.

Input(s):

- Code snippets are provided by the user in the code editor.
- Preprocessed code corpus for model training.

Step-by-Step Control Flow / Data Flow:

- A user enters code in the frontend code editor.
- The front end captures the code and sends it to the backend via **WebSocket/REST API.**
- The backend's inference service processes the code snippet and passes it to the machine learning model.
- The trained model predicts the next token or set based on the input code.
- Predictions are sent back to the front end, where they are displayed as suggestions in the code editor.
- Feedback from the user is captured to improve model performance.
- Periodic retraining of the model using feedback and new data.

Output(s):

- Autocomplete suggestions for the user's code.
- Feedback on the model’s performance for continuous improvement.

System Design

Component Details:

1. **Frontend (User Interaction Layer)**:
    - Code is written by users in the code editor, which sends the code snippets to the backend for predictions.
    - **WebSocket** or **REST API** calls are made from the editor to the **API Gateway**.
2. **API Gateway**:
    - The API Gateway manages and routes all incoming requests.
    - It also handles **security aspects** like **authentication** and **throttling** to manage the request flow.
3. **Inference Service**:
    - Once the code snippet reaches the **Inference Service**, it is processed by the trained model to generate predictions.
    - The results are sent back to the user via the gateway, ensuring **low-latency** predictions.
4. **Training Service**:
    - The **Training Service** is responsible for training the model using **self-supervised learning (SSL)**.
    - It manages the model's training and periodic updates based on the feedback.
5. **Feedback Service**:
    - This service collects and processes user feedback for improving the model.
    - Feedback is stored in the database and used for retraining the model.
6. **Model Storage**:
    - **Version control** ensures that models can be easily rolled back if needed.
    - This component handles the secure storage of trained models and ensures **continuous deployment** during model updates.
7. **Database**:
    - Stores **feedback**, **logs**, and the **code corpus**. It supports fast retrieval for model training and debugging purposes.
8. **Code Corpus Storage**:
    - Stores the large-scale code data for training and fine-tuning the model.
    - This repository is updated with new data regularly to ensure that the model remains up to date with the latest trends.

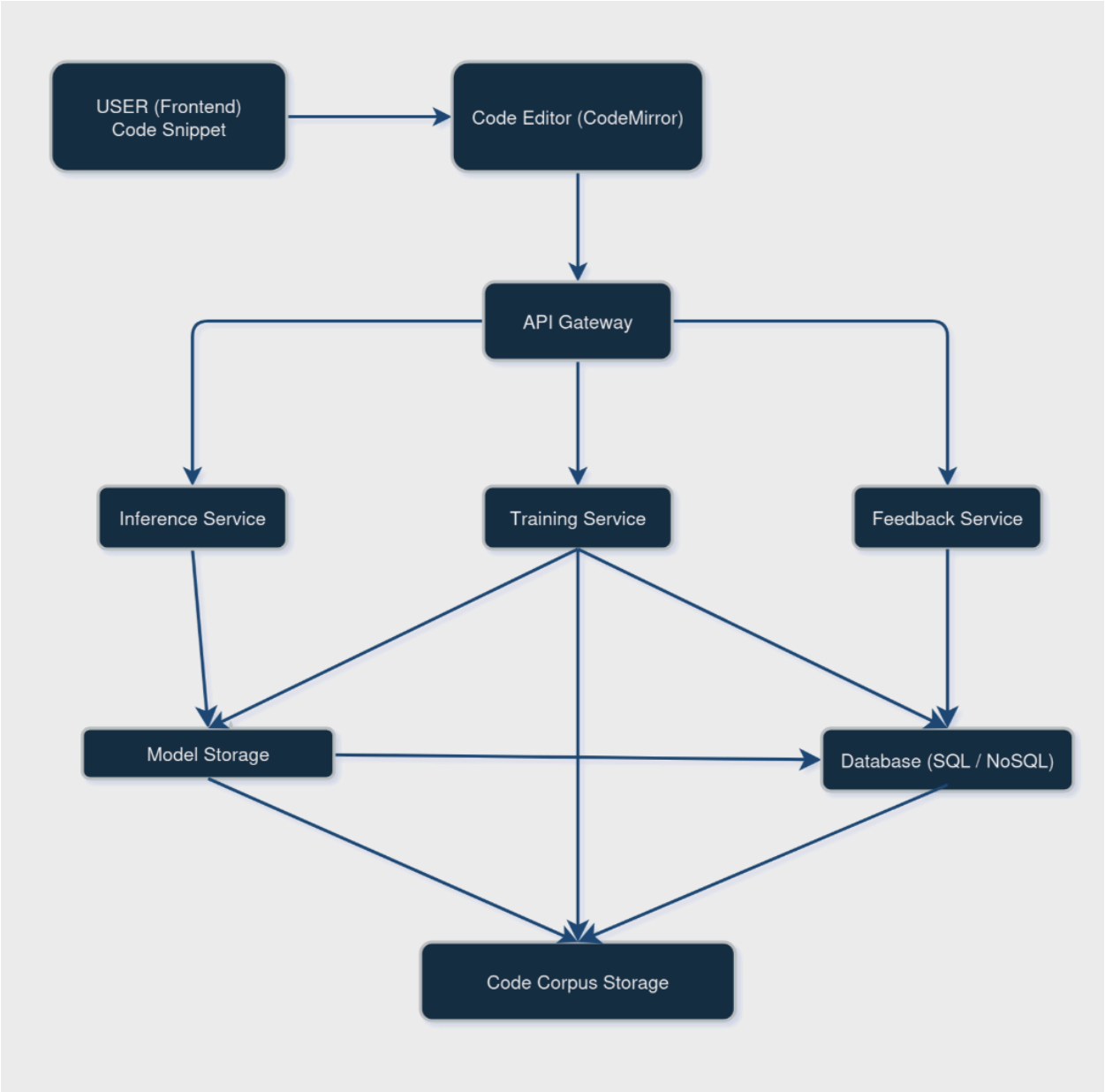

Fig. A code completion platform with a user interface connecting to backend services for inference, training, and feedback, all managed via an API Gateway and supported by model storage, a database, and code corpus storage.

Stakeholders

1. **Project Team Members:**
    - Responsible for designing, developing, and implementing the system.
2. **End Users (Developers):**
    - Developers who will use the code completion tool to enhance productivity.
3. **Faculty Supervisor:**
    - To guide the project and ensure it meets academic and technical expectations.
4. **POC (Point of Contact):**
    - Acts as the communication link between the project team and stakeholders.

User-Cases

1. Real-time code completion for Python.
2. Code completion in Java for large-scale enterprise projects.
3. Predictive coding for JavaScript to increase development speed.
4. Error correction suggestions while typing.
5. Providing multiple autocomplete suggestions ranked by relevance.
6. Continuous learning based on user feedback to improve predictions.
7. Training the model on a new programming language.
8. Detecting and completing frequently used code patterns.
9. Cross-referencing code snippets to suggest best practices.
10. Supports different code styles for different IDEs.

UI/UX

- **UI Sample Screen 1:** A text editor where developers can input incomplete code and see autocomplete suggestions.
- **UI Sample Screen 2:** A dashboard showing the model's predictions and accuracy on test datasets.

Delivery Timeline

Week	Milestone
Week 1	**Finalize System Architecture and Design:** Review overall architecture, finalize microservices, and prepare initial design diagrams. Identify code corpus for training.
Week 2	**Dataset Preparation and Preprocessing:** Collect, clean, and preprocess the code dataset. Since we are provided with dataset, we can directly use [https://github.com/microsoft/CodeXGLUE/blob/main/Code-Code/Clone-detection-POJ-104/README.md] it. Define the pipeline for data tokenization and splitting.
Week 3	**Basic Model Training Pipeline Setup:** Implement the training pipeline using Transformer models (e.g., GPT-3 or CodeBERT). Run initial training with a small dataset.
Week 4	**Frontend UI Design and Development:** Design the code editor UI (Monaco / CodeMirror) and feedback modal. Establish communication with the backend through REST API / WebSocket.
Week 5	**Backend Development and API Gateway Setup:** Implement core microservices (Inference Service, Training Service, Feedback Service) and set up the API Gateway for routing requests.
Week 6	**Model Integration with Frontend:** Integrate the trained model with the frontend. Enable real-time code suggestions in the editor.
Week 7	**Model Optimization for Low-Latency Predictions:** Optimize model inference using frameworks like TensorFlow Serving or TorchServe to reduce prediction latency.
Week 8	**Testing and Feedback Gathering:** Conduct extensive testing with multiple users, focusing on usability and accuracy. Collect feedback for further model improvements.
Week 9	**Retraining Model with Feedback Data:** Retrain the model based on the feedback gathered. Fine-tune parameters and improve suggestion accuracy.
Week 10	**Final Deployment, Performance Tuning and Documentation:** Deploy the system with optimized performance and ensure security compliance. Monitor real-time performance and make final adjustments. Ensure all documentation is complete and accurate.

Work-Breakdown Structure (WBS) with Work Distribution

Week 1: Finalize System Architecture and Design

- **Tasks**:
  - Finalize system architecture and design (microservices-based).
  - Create a system design diagram (Client / Server / Middleware).
  - Identify the code corpus for training.
  - Design initial UI wireframes for the code editor and feedback modal.
  - Define data flow and control flow for the system.
- **Work Distribution**:
  - **Member 1**: Research and finalize microservices architecture.
  - **Member 2**: Create the system design diagram.
  - **Member 3**: Identify the code corpus for training.
  - **Member 4**: Design initial UI wireframes for code editor and feedback modal.

Week 2: Dataset Preparation and Preprocessing

- **Tasks**:
  - Collect relevant code datasets.
  - Clean and preprocess the dataset.
  - Create a tokenization pipeline.
  - Implement code normalization techniques (e.g. removing comments, whitespace).
- **Work Distribution**:
  - **Member 1**: Identify and collect diverse code corpora.
  - **Member 2**: Clean and preprocess code data.
  - **Member 3**: Create a tokenization pipeline for training.
  - **Member 4**: Implement code normalization techniques.

Week 3: Basic Model Training Pipeline Setup

- **Tasks**:
  - Implement the initial model training pipeline using Transformer-based models (GPT-3 or CodeBERT).
  - Set up the hardware environment (GPUs / TPUs).
  - Configure Transformer models.
  - Prepare the dataset for training.

- **Work Distribution**:
    - **Member 1**: Set up the hardware environment for model training.
    - **Member 2**: Configure Transformer models (GPT-3 or CodeBERT).
    - **Member 3**: Prepare the dataset for training.
    - **Member 4**: Implement the training pipeline.

Week 4: Frontend UI Design and Development

- **Tasks**:
    - Finalize frontend design for the code editor and feedback modal.
    - Implement the code editor interface (Monaco / CodeMirror).
    - Establish communication between frontend and backend (REST API/WebSocket).
- **Work Distribution**:
    - **Member 1**: Finalize frontend design for the code editor and feedback modal.
    - **Member 2**: Implement the code editor interface using Monaco / CodeMirror.
    - **Member 3**: Set up communication between frontend and backend.
    - **Member 4**: Integrate REST API or WebSocket into the design.

Week 5: Backend Development and API Gateway Setup

- **Tasks**:
    - Implement core microservices (Inference Service, Training Service, Feedback Service).
    - Set up the API Gateway for routing and load balancing.
    - Implement security protocols for API authentication.
- **Work Distribution**:
    - **Member 1**: Develop Inference Service for code predictions.
    - **Member 2**: Work on the Feedback Service for collecting user feedback.
    - **Member 3**: Implement the Training Service for retraining.
    - **Member 4**: Set up the API Gateway and implement security protocols.

Week 6: Model Integration with Frontend

- **Tasks**:
  - Integrate a trained model with the frontend for real-time code suggestions.
  - Implement feedback buttons in the code editor.
  - Test and troubleshoot frontend-backend integration.
- **Work Distribution**:
  - **Member 1**: Integrate a trained model for real-time code suggestions.
  - **Member 2**: Implement feedback buttons and communication with the Feedback Service.
  - **Member 3**: Test frontend-backend integration.
  - **Member 4**: Troubleshoot integration issues.

Week 7: Model Optimization for Low-Latency Predictions

- **Tasks**:
  - Optimize inference model for low-latency predictions.
  - Set up serving infrastructure (TensorFlow Serving or TorchServe).
  - Profile the system and identify performance bottlenecks.
- **Work Distribution**:
  - **Member 1**: Optimize the inference model.
  - **Member 2**: Set up the serving infrastructure (e.g., TensorFlow Serving).
  - **Member 3**: Profile the system for performance issues.
  - **Member 4**: Identify bottlenecks and optimize performance.

Week 8: Testing and Feedback Gathering

- **Tasks**:
  - Conduct user testing and gather feedback.
  - Monitor system performance during user testing.
  - Analyze user feedback and identify areas of improvement.
- **Work Distribution**:
  - **Member 1**: Coordinate user testing and feedback gathering.
  - **Member 2**: Monitor system performance.
  - **Member 3**: Collect and analyze user feedback.
  - **Member 4**: Categorize issues and prioritize improvements.

Week 9: Retraining Model with Feedback Data

- **Tasks**:
    - Retrain the model based on feedback.
    - Fine-tune model parameters for improved accuracy.
    - Validate improvements in code suggestions.
- **Work Distribution**:
    - **Member 1**: Retrain the model based on feedback data.
    - **Member 2**: Fine-tune model parameters.
    - **Member 3**: Validate the accuracy and performance of the updated model.
    - **Member 4**: Test improvements in code suggestions.

Week 10: Final Deployment, Performance Tuning and Documentation

- **Tasks**:
    - Finalize deployment with performance optimization.
    - Conduct security audits and apply necessary encryption.
    - Set up real-time performance monitoring (latency, accuracy, feedback).
- **Work Distribution**:
    - **Member 1**: Handle the final deployment and system performance tuning.
    - **Member 2**: Conduct security audits and apply encryption for sensitive data.
    - **Member 3**: Set up real-time performance monitoring tools.
    - **Member 4**: Monitor system health and ensure ongoing system stability.
    - **All Member:** Ensure all documentation is complete and accurate.