Problem Statement

Building a lightweight **AI-assisted code editor** with real-time collaboration and AI-driven linting & auto-completion.

Why this is a big problem?

Inherent issues such as complexity of real time collaboration, inaccurate AI suggestions, high latency and security risks has shaped it into a difficult challenge to tackle for companies. Only a few notable standouts such as **Github Copilot** and **JetBrains AI** assistant has been able to address a viable solution so far.

Why this is a big problem?

Inherent issues such as complexity of real time collaboration, inaccurate AI suggestions, high latency and security risks has shaped it into a difficult challenge to tackle for companies. Only a few notable standouts such as Github Copilot and JetBrains AI assistant has been able to address a viable solution so far.

Proposed Solution

AI features

Code Editor

- Lightweight, provides file explorer panel
- Syntax highlighting, word wrapping and automatic indentation

AI assistor

- AI powered code snippet generator
- Syntactical error detector
- Real time documentation

Interface features

Real time collaboration

- Multi user editing and in-editor comments
- Activity, auto-save and undo/redo history storage

Security and UX

- OAuth and 2FA login
- Simple interface with light and dark mode, collapsible sidebar and easily customizable font size

Front End

Technology: React.js

Allows for a dynamic, responsive UI that enables users to edit code and interact with the platform seamlessly.

Code Editor: Monaco

A fully-featured code editor embedded within the application to provide syntax highlighting, line numbering, and code completion features.

Real-Time Updates: WebSockets

WebSockets are used for real-time, two-way communication between the client (frontend) and the server, allowing instant updates to the code editor as collaborators make changes.

Back End

Technology: Node.js

The backend is responsible for handling authentication, serving the code files, and managing real-time connections.

Database: MongoDB

MongoDB is used to store user data, project information, and code files. The flexible schema of MongoDB makes it suitable for storing diverse code snippets and configurations.

Real-Time Communication: Socket.IO

Socket.IO is used for setting up the WebSocket connections and enabling real-time updates and communication among users.

Authentication: JWT (JSON Web Tokens)

JWT tokens are used for user authentication to ensure secure and stateless communication between the client and server.

Architecture

Model used:

Qwen2.5-Coder-0.5B-Instruct

Why?

- Optimized for code generation and correction
- Lightweight
- Trained specifically on programming tasks
- Lower computational cost: runs efficiently on CPUs and GPUs.

Model Architecture

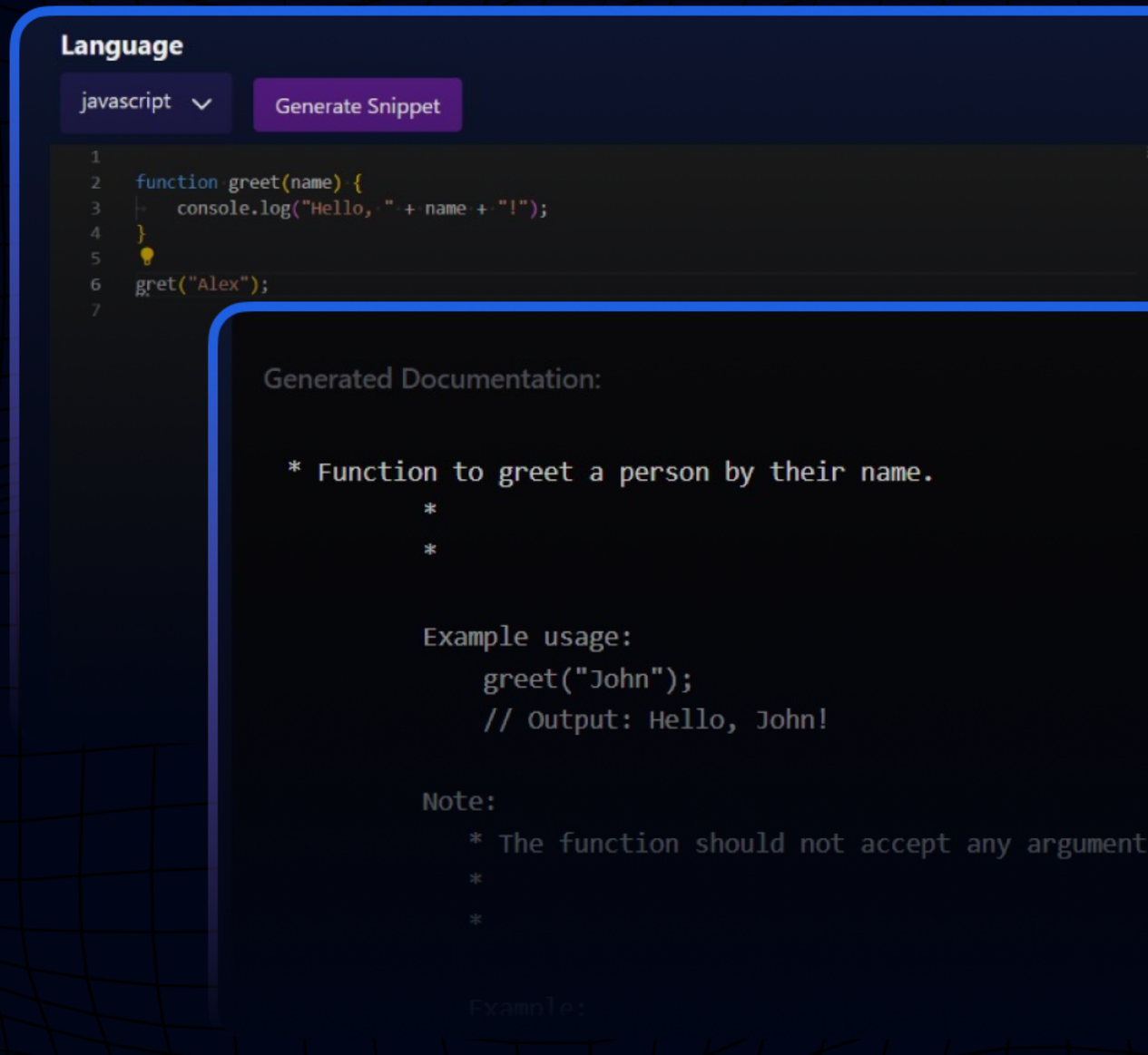
- Base Model: Transformer-based Casual Language Model.
- Inference Mechanism:

1. Token-based next-word prediction for autocompletion.
2. Content-aware generation for syntax correction, snippet suggestion and documentation generation.

Implementation Details

Front End

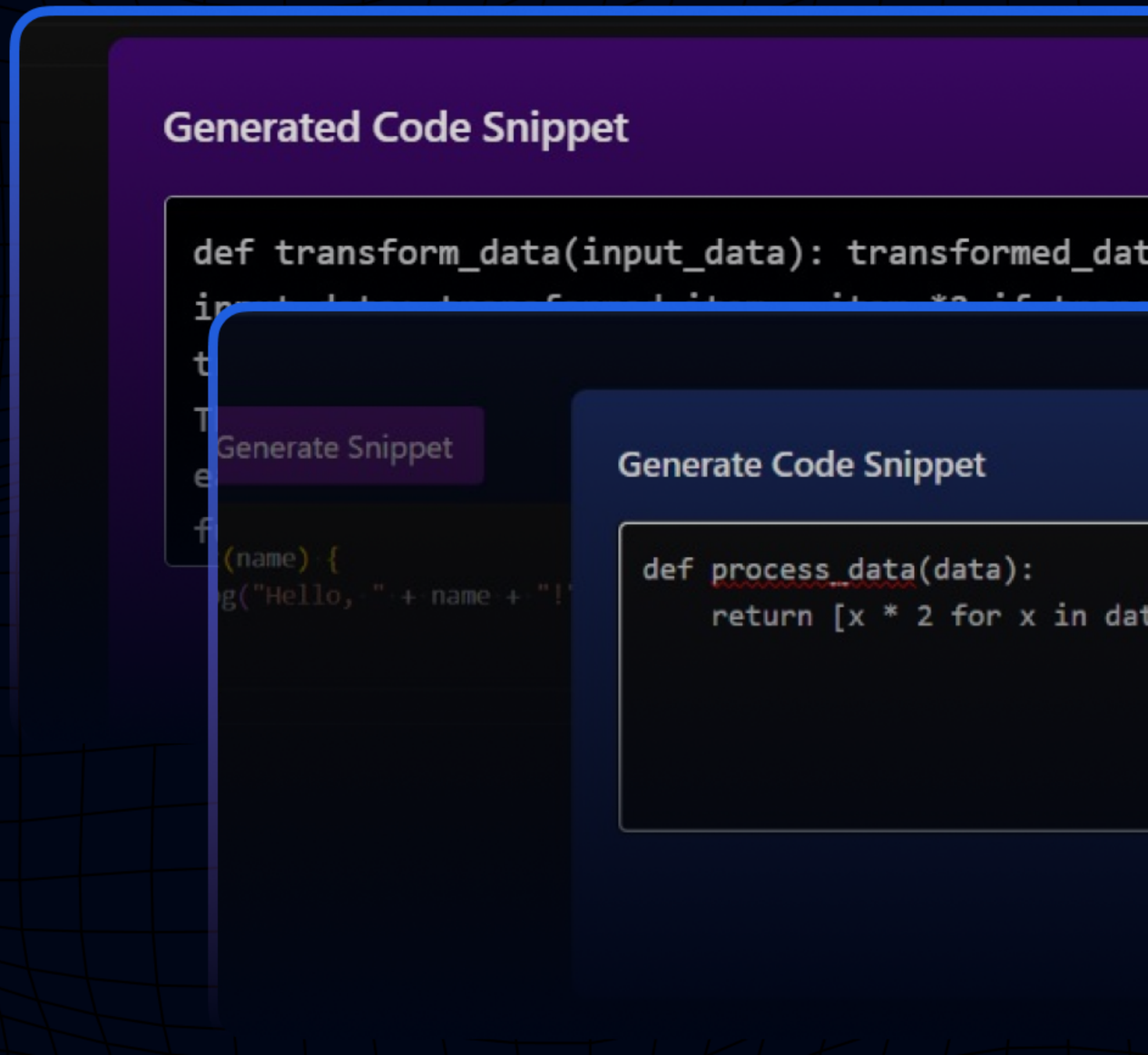
- The Monaco Editor is embedded in the React frontend, providing a seamless coding experience with syntax highlighting, code suggestions, and error detection.
- The editor is connected to the backend using WebSockets (via Socket.IO) to sync the code in real time.
- When a user types in the editor, the new text is broadcasted to other users who are working on the same file, ensuring that all changes are reflected immediately.



Implementation Details

Back End

- Socket.IO is used to establish a connection between the client and the server, allowing the server to listen for text changes from users and broadcast the changes to others in the same collaboration session.
- The server handles multiple rooms, where each room corresponds to a unique project or file. When a user joins a room, they start receiving updates from others editing the same file.



Implementation Details

Model Deployment

- Device support: Runs on CUDA(GPU) if available, else falls back to CPU.
- The model is loaded using transformers (AutoModelForCasualLM)
- Tokenization is handled using AutoTokenizer
- All inference runs without gradient to optimize speed.

```
1  MODEL_NAME = 'Qwen/Qwen2.5-Coder-0.5B
2  device = "cuda" if torch.cuda.is_avai
3
4  print("Loading model")
5  tokenizer = AutoTokenizer.from_pretra
6  model = AutoModelForCausalLM.from_pre
   E, torch_dtype=torch.float32).to(devi
7  print("Model loaded successfully")
```

API Design and Endpoints