CS Tutor AI - Complete Project Documentation

Project Overview

CS Tutor AI is an AI-powered coding assistant that enhances learning experiences by explaining code snippets and providing interactive educational features. Using Streamlit as our frontend, we create a web-based interface that allows users to generate, understand, and learn from AI-assisted code explanations.

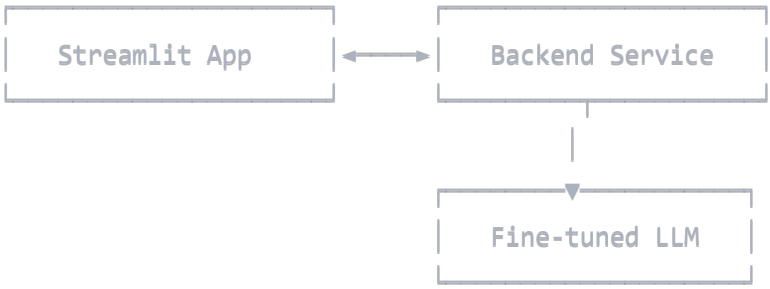
Project Deadline: May 12, 2025

Key Features

1. **Intelligent Code Generation**: Standard code completion and generation
2. **Explanation Mode**: Detailed breakdown of code functionality, time/space complexity, and design choices
3. **Learning Mode**: Interactive tutoring experience with adjustable difficulty levels
4. **Language-Specific Best Practices**: Customized guidance for different programming languages
5. **Contextual Awareness**: Adapts explanations based on the user's skill level and project context

Technical Architecture

System Components



Tech Stack

AI/ML Components

- Base Model: CodeLlama-34B or StarCoder2-15B
- Fine-tuning: QLoRA/LoRA using PEFT library
- Inference Engine: vLLM or Text Generation Inference (TGI)
- Quantization: GPTQ or AWQ for performance optimization

Backend

- API Framework: FastAPI

- Deployment: Docker containers
- Database: SQLite for session history (optional)

Frontend

- Streamlit for web interface
- Streamlit components for syntax highlighting and interactive elements
- Communication: HTTP requests to backend

Development Roadmap

Day 1-2: Foundation & Setup (May 9-10)

- Initialize GitHub repository
- Set up development environment
- Define project architecture and components
- Set up Streamlit app skeleton
- Select base model for backend
- Create initial documentation

Day 3: Backend Development (May 11)

- Implement FastAPI backend endpoints
- Set up model inference pipeline
- Create API for code generation and explanation
- Test API functionality
- Connect to pre-trained model (use existing model rather than fine-tuning to save time)

Day 4: Frontend & Integration (May 12)

- Build Streamlit UI components
- Implement code editor with syntax highlighting
- Create explanation display panels
- Set up user preferences (language, difficulty level)
- Connect frontend to backend API
- Perform end-to-end testing
- Deploy MVP
- Complete final documentation

Technical Implementation Details

Streamlit App Structure

```
cs-tutor-streamlit/
├── app.py              # Main Streamlit application
├── pages/
│   ├── home.py         # Home/landing page
│   ├── generate.py     # Code generation page
│   ├── explain.py      # Code explanation page
│   └── learn.py        # Interactive learning page
├── components/
│   ├── code_editor.py     # Custom code editor component
│   ├── explanation.py     # Explanation display component
│   └── settings.py        # User settings component
├── utils/
│   ├── api_client.py      # Backend API communication
│   └── code_formatter.py # Code formatting utilities
├── config.py           # Configuration settings
└── requirements.txt    # Dependencies
```

Backend API Endpoints

```
POST /api/generate
- Generates code based on prompt and context
- Returns generated code

POST /api/explain
- Takes code and generates explanations
- Returns detailed explanation at requested depth

POST /api/interactive
- Handles follow-up questions about code
- Returns answers to specific questions

GET /api/status
- Returns model and service status
```

User Interaction Flow

1. User visits the CS Tutor AI Streamlit app
2. User selects desired mode (generate, explain, learn)
3. For code generation:
   - User enters a description of the code they need
   - App sends request to backend

- Generated code is displayed with syntax highlighting

4. For code explanation:
   - User pastes/writes code in the editor
   - Selects explanation depth (basic, intermediate, advanced)
   - App sends code to backend
   - Detailed explanation is displayed in panels
5. For learning mode:
   - User selects a topic and difficulty level
   - App provides guided learning experience with code examples and explanations
   - User can ask follow-up questions

Model Prompt Template

```python
PROMPT_TEMPLATE = """
Mode: {mode}
Language: {language}
Difficulty: {difficulty}
User Level: {user_level}

Code:
{code}

Question/Task:
{question}

Instructions:
{instructions}
"""
```

Implementation Strategy

Backend Implementation (FastAPI)

python

```python
# Example FastAPI Backend Code
from fastapi import FastAPI, Body
from pydantic import BaseModel
import uvicorn
import os
from transformers import pipeline

app = FastAPI()

# Initialize model (use pre-trained to save time)
model = pipeline(
    "text-generation",
    model="codellama/CodeLlama-7b-Instruct-hf",
    device_map="auto"
)

class CodeRequest(BaseModel):
    code: str = ""
    language: str = "python"
    mode: str = "explain"
    difficulty: str = "intermediate"
    question: str = ""

@app.post("/api/explain")
def explain_code(request: CodeRequest):
    prompt = f"""
    Explain the following {request.language} code in {request.difficulty} level terms:

    ```{request.language}
    {request.code}
    ```

    Provide a detailed explanation covering:
    1. What the code does
    2. How it works step by step
    3. Time and space complexity
    4. Best practices and potential improvements
    """

    response = model(prompt, max_length=1024, temperature=0.7)
    return {"explanation": response[0]["generated_text"]}

Add other endpoints: generate, interactive, etc.
```

```
if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Streamlit App Implementation

python

```python
# Example main app.py
import streamlit as st
import requests
from components.code_editor import create_code_editor
from components.explanation import create_explanation_panel
from utils.api_client import APIClient

st.set_page_config(page_title="CS Tutor AI", layout="wide")
st.title("CS Tutor AI - Your Intelligent Coding Assistant")

# Initialize API client
api_client = APIClient(base_url="http://localhost:8000")

# Mode selection
mode = st.sidebar.selectbox(
    "Select Mode:",
    ["Code Generation", "Code Explanation", "Interactive Learning"]
)

# Language selection
language = st.sidebar.selectbox(
    "Programming Language:",
    ["Python", "JavaScript", "Java", "C++", "Go"]
)

# Difficulty/detail level
difficulty = st.sidebar.select_slider(
    "Explanation Detail:",
    options=["Basic", "Intermediate", "Advanced"]
)

# Main interface based on mode
if mode == "Code Explanation":
    st.header("Code Explanation")

    # Code input
    code = create_code_editor(language.lower())

    if st.button("Explain Code"):
        if code:
            with st.spinner("Generating explanation..."):
                response = api_client.explain_code(
                    code=code,
                    language=language.lower(),
                    difficulty=difficulty.lower()
                )
```

```
            # Display explanation
            create_explanation_panel(response["explanation"])
        else:
            st.warning("Please enter some code to explain.")

# Add other modes: generation, learning, etc.
```

Evaluation Metrics

1. **Code Correctness**: % of generated code that runs without errors
2. **Explanation Quality**: Rated by test users (1-5 scale)
3. **Response Time**: Latency for generating responses
4. **Learning Effectiveness**: User-reported understanding improvement

Advantages of Streamlit Approach

1. **Rapid Development**: Streamlit allows for quick iteration and prototyping
2. **Web Accessibility**: Users can access via any browser without installing extensions
3. **Simple Deployment**: Can be deployed to Streamlit Cloud or any cloud service
4. **Rich UI Components**: Streamlit provides many built-in components for interactive interfaces
5. **Easy Demo**: Ideal for showcasing the project's functionality

Deployment Options

1. **Local Development**: `streamlit run app.py`
2. **Streamlit Cloud**: Push to GitHub and deploy directly from Streamlit Cloud
3. **Docker Containers**: Package both frontend and backend in containers
4. **Cloud Services**: Deploy on AWS, GCP, or Azure

Future Enhancements

1. **Multimodal Support**: Add visualization for algorithms and data structures
2. **Progress Tracking**: Monitor user learning and adapt difficulty
3. **Custom Domain Support**: Allow training on company-specific codebases
4. **Offline Mode**: Enable fully local operation
5. **VS Code Extension**: Expand to IDE integration as a future enhancement

Required Skills

- Python development

- Streamlit UI design
- FastAPI backend development
- AI/ML knowledge (transformers)

Resource Requirements

- Development machine
- GPU for inference (or cloud-based GPU service)
- Storage for models
- Deployment server (or cloud service)

Testing Strategy

1. Unit tests for backend endpoints
2. Integration tests for API communication
3. User acceptance testing with CS students

Deployment Checklist

- All planned features implemented
- Documentation complete
- Performance metrics meet targets
- Security review completed
- Deployment configuration ready
- README and user guides finalized

This documentation is a living document and will be updated throughout the development process.