# CodeMentor

## AI-Powered Coding Assistant

Transformer-based LLMs meet backend systems to create an intelligent coding companion. Built with Ollama, Flask, and advanced prompt engineering.

# Problem Statement

### Learning Curve

Beginners struggle with syntax, patterns, and debugging unfamiliar codebases.

### Debugging Overhead

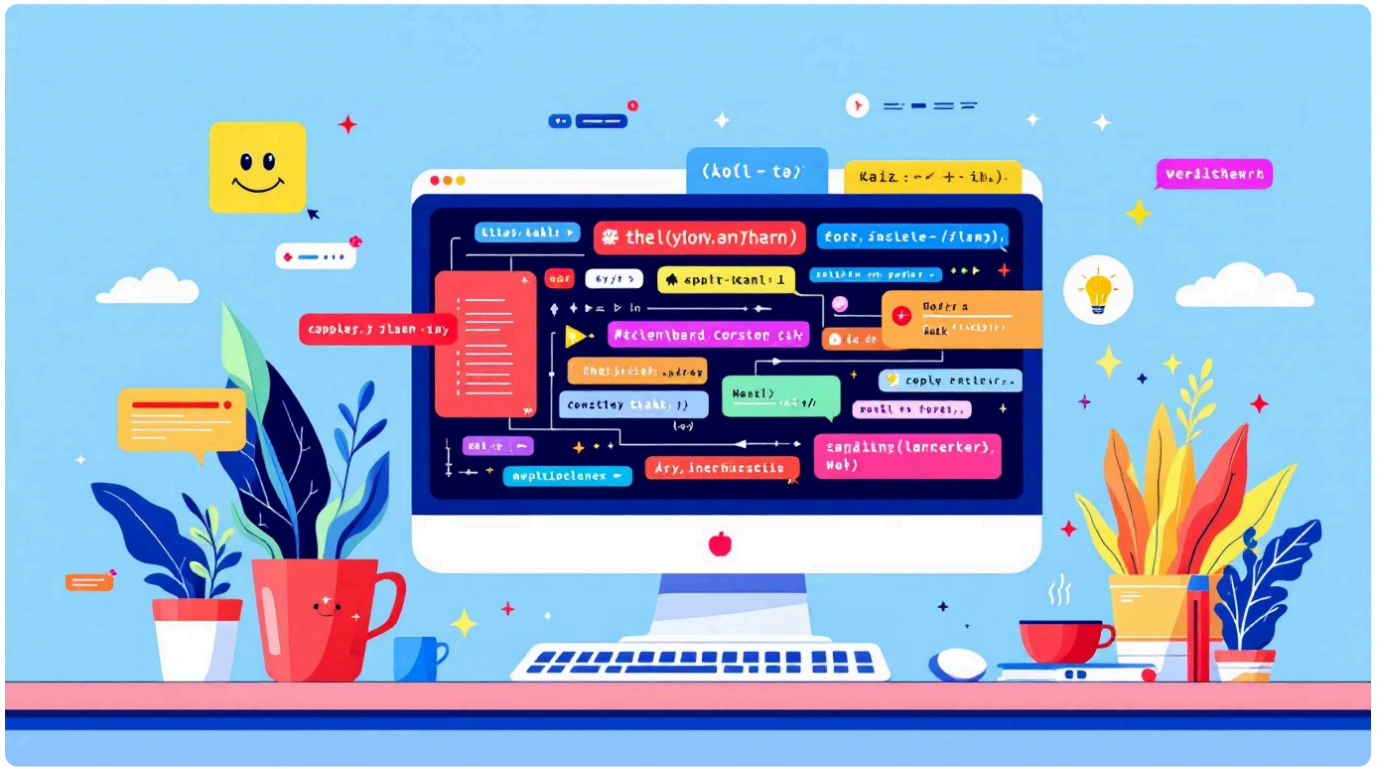
Hours spent identifying bugs instead of building features and learning concepts.

### Optimization Blindspots

Difficulty identifying performance bottlenecks and refactoring opportunities in existing code.

Developers need real-time guidance that understands context, explains trade-offs, and adapts to skill levels.

# Project Motivation



## Democratizing Development

Lower entry barriers for students and junior developers through instant expert guidance.

## Accelerated Learning

Real-time explanations transform passive coding into active learning experiences.

## Productivity Multiplier

Reduce debugging time from hours to minutes with AI-powered root cause analysis.

# Solution Overview

## User Interface

Web-based frontend accepts code input and queries

## Backend Engine

Flask API processes requests and formats prompts

## LLM Core

Quantized transformer models via Ollama

## Intelligent Response

Context-aware code analysis and explanations

Transformer-based architecture combines tokenization, self-attention mechanisms, and next-token prediction for contextual code understanding.

# System Architecture

01

## Client Request

Frontend sends code and context to Flask backend

02

## Prompt Engineering

Backend formats structured prompts with context and instructions

03

## LLM Processing

Ollama routes to quantized models with optimized inference

04

## Response Generation

Transformer decodes tokens using positional embeddings and attention

05

## Result Delivery

Structured JSON response parsed and displayed to user

# Technology Stack

## Backend & API

- Python 3.11 with type hints
- Flask for RESTful endpoints
- SQLAlchemy for data persistence
- WebSocket for streaming responses
- JWT for authentication

## LLM Layer

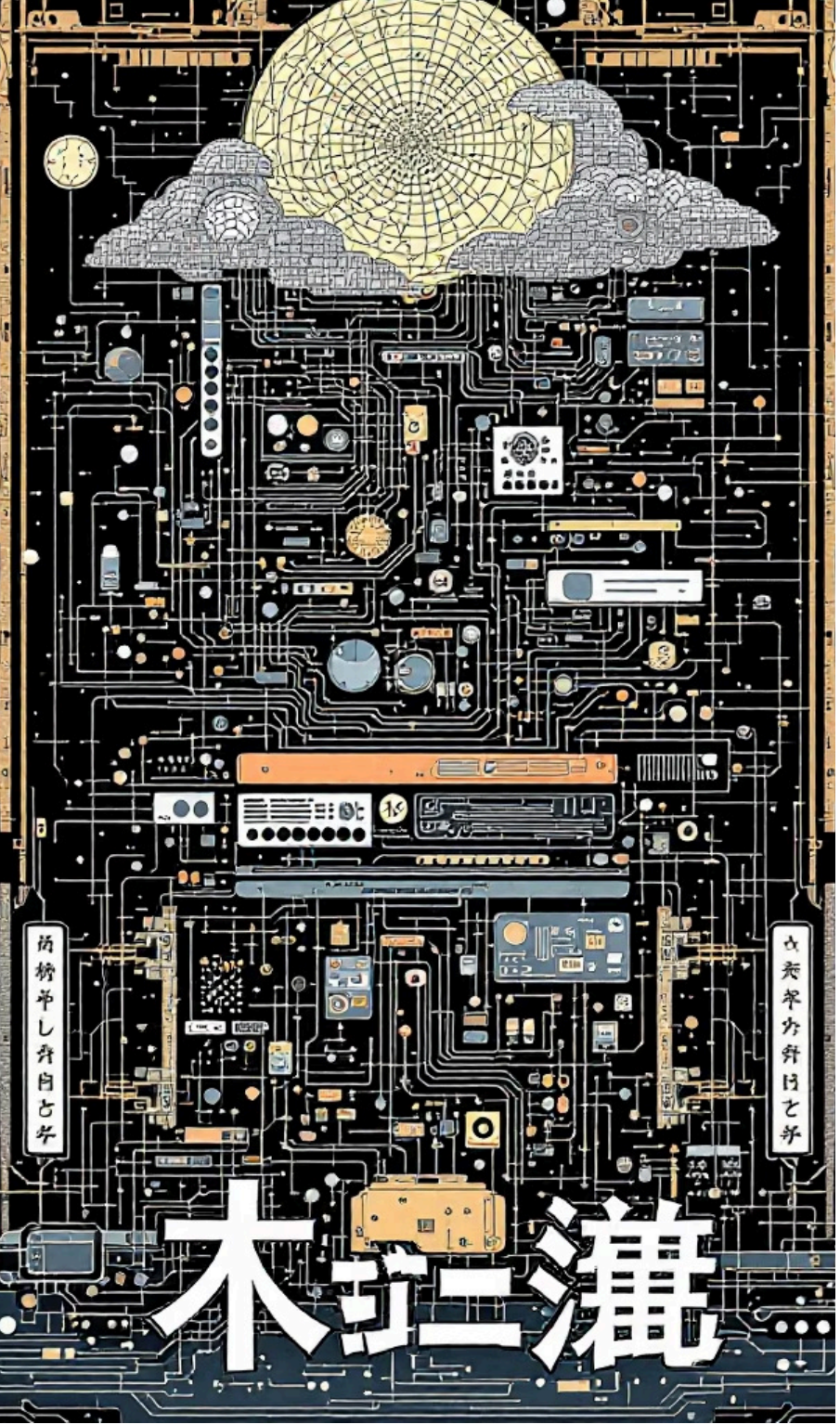
- Ollama for model management
- Llama3 8B/70B parameters
- Mistral 7B quantized variants
- GGUF format for efficiency

## Frontend

- React with TypeScript
- CodeMirror for syntax highlighting
- WebSocket integration
- Material UI components

## Infrastructure

- Docker containers
- NGINX reverse proxy
- Redis for caching
- PostgreSQL for metadata



# LLM Architecture Deep Dive

## Tokenization

Byte-pair encoding splits code into subword units, mapping to vector embeddings

## Transformer Layers

Multi-head self-attention computes token relationships using query, key, value projections

## Context Processing

Positional encodings inject sequence order, enabling attention to weight relevant tokens

## Token Prediction

Softmax output predicts next token probabilities, sampled during generation

# Prompt Engineering Strategy

1

### System Role Definition

Prepends "You are an expert software engineer" to establish persona and expertise level

2

### Context Injection

Embeds surrounding code, error messages, and stack traces for situational awareness

3

### Structured Output

Demands JSON format with specific fields: explanation, fix, complexity analysis

4

### Chain-of-Thought

Requests step-by-step reasoning before final answer, improving accuracy and explainability

Backend formats prompts with context windows limited to model's token capacity (4K-32K tokens).

# Model Selection & Optimization

## Transformer Models

- Llama3 (8B, 7B): balanced performance
- Mistral 7B: faster inference
- Gemini variants: multimodal

## Quantization Benefits

- GGUF format reduces size 60%
- 4-bit quantization enables CPU-only
- Lower VRAM requirements

## Performance Metrics

- 2-3x faster inference vs. full-precision
- ~80% accuracy retention
- Real-time streaming

## Latency Reduction

- Model caching in memory
- Prefill optimization
- Early stopping heuristics

# Key Features

## Code Explanation

Line-by-line breakdown of unfamiliar code with complexity analysis and pattern identification using AST parsing.

## Intelligent Debugging

Root cause analysis of errors with fix suggestions, stack trace correlation, and test case generation.

## Optimization Engine

Identifies bottlenecks, suggests refactoring patterns, and provides Big-O analysis with alternative approaches.

## Interview Mode

DSA problem solving with step-by-step guidance, time/space complexity verification, and coding best practices.