Interaction Logs: Study Buddy

Data Science Internship Assignment

Monday 3rd November, 2025

Abstract

This document contains the complete interaction logs between the developer and AI during the development of the Study Buddy AI Agent. The logs span the entire project lifecycle from initial brainstorming through final implementation, covering technical discussions, problem-solving, architecture decisions, and implementation details. These interactions demonstrate the iterative development process and AI-assisted problem-solving methodology.

Contents

1 Phase 1: Project Brainstorming & Architecture **2**
1.1 Initial Project Discussion 2

2 Phase 2: Model Selection & Configuration **3**
2.1 Inference Strategy Discussion 3
2.2 Model Loading Issue 3
2.3 Instruction Model Success 4

3 Phase 3: Fine-tuning Pipeline **5**
3.1 Fine-tuning Data Strategy 5

4 Phase 4: Technical Implementation Issues **7**
4.1 Configuration Type Error 7

5 Summary of Key Technical Decisions **8**
5.1 Model Selection Justification 8
5.2 Architecture Decisions 8

6 Conclusion **8**

1 Phase 1: Project Brainstorming & Architecture

1.1 Initial Project Discussion

User Prompt

I am building an AI agent for my internship that can reason, plan, and execute to automate a manual task. The agent must use at least one fine-tuned model. I was thinking of a "Study Buddy" system with multiple specialized agents. What is your review of this approach?

AI Response

Your "Study Buddy" idea is excellent! The multi-agent approach is sophisticated and shows good understanding of AI system design. Let me help you refine this into an actionable plan:

Key Strengths of Your Approach:

- Clear task decomposition (Fetcher, Study Assistant, Formatter, Submitter)
- Realistic scope with human-in-the-loop verification
- Leveraging existing tools for hard problems

Critical Feasibility Considerations:

- Start with V1 (RAG Agent) as MVP - most immediate value
- The Formatter agent is perfect for fine-tuning requirement
- Use official APIs where possible (Canvas, Gradescope)
- Human verification is crucial for academic integrity

Recommended Implementation Order:

1. RAG Study Assistant (immediate value)
2. LaTeX Formatter (fine-tuning target)
3. Document Fetcher
4. Assignment Submitter

This transforms your idea from speculative to staged, actionable project. Would you like to proceed with the RAG agent implementation first?

2 Phase 2: Model Selection & Configuration

2.1 Inference Strategy Discussion

User Prompt

For inference i was thinking of using groq, using from groq import Groq. And for finetuning i will be using any 8b parameter model that is free. What do you think about it ?

AI Response

Excellent approach! Using Groq for fast inference with large models and fine-tuning a smaller 8B model for specialized formatting is very practical. Let me update the architecture:

Updated Strategy:

- RAG Agent: Groq with 70B models for best reasoning
- Formatter Agent: Free 8B model (Mistral 7B) for LaTeX conversion
- Fine-tuning: QLoRA on 8B model for efficiency

Key Benefits:

- Cost-effective (Groq for complex reasoning, local for formatting)
- Specialized models for different tasks
- Fine-tuning feasible on consumer hardware

I'll update the configuration and create:

1. Groq client wrapper for RAG agent
2. Local model client for Formatter agent
3. Proper model configuration switching

Updated Configuration: inference: groq: model: "llama-3.3-70b-versatile" temperature: 0.1 max_tokens: 1024
local: formatter_model: "mistralai/Mistral-7B-Instruct-v0.2" device: "cuda" temperature: 0.1 max_tokens: 2048

2.2 Model Loading Issue

User Prompt

I got an issue with the Mistral-7B model in the formatter agent - getting blank responses. What do you think?

AI Response

This is a common issue! The blank response suggests several possible causes. Let's debug systematically:
Root Cause Analysis:

1. **Prompt Formatting**: Mistral models need specific chat templates
2. **Model Type**: You might be using the base model instead of instruction-tuned
3. **Generation Parameters**: Temperature might be too low
4. **Model Loading**: Possible device or memory issues

Immediate Debug Steps:

1. Test with instruction-tuned model: "mistralai/Mistral-7B-Instruct-v0.2"
2. Use proper chat templates with `tokenizer.apply_chat_template()`
3. Increase temperature to 0.7 for more diverse outputs
4. Add fallback generation methods

Let me create a debug script and provide the fixes:

> **Debug Script: test_mistral.py** # Test basic generation def test_mistral_basic(): model_name = "mistralai/Mistral-7B-Instruct-v0.2" tokenizer = AutoTokenizer.from_pretrained(model_name) model = AutoModelForCausalLM.from_pretrained(model_name) test_prompt = "Convert this to LaTeX: Solve x^2 + 1 = 0"$inputs = tokenizer(test\_prompt, return\_tensors = "pt")outputs = model.generate(* * inputs)response = tokenizer.decode(outputs[0])print(f"Response : response")$

Run this script first to isolate the issue. I'll also provide updated formatter agent code with proper chat template handling.

2.3 Instruction Model Success

User Prompt

Handle this chat template error.

AI Response

Let’s fix the chat template error properly.
The issue is that the base tokenizer doesn’t have a chat template configured. Here’s the solution:
Key Fixes Needed:

- Use `tokenizer.apply_chat_template()` for models that support it
- Add fallback to manual formatting for base models
- Implement proper message formatting with system prompts
- Add error handling and template testing

Updated Local Client: def _format_chat_template(self, prompt: str, system_message: str = None) -¿ str: messages = [] if system_message: messages.append(”role”: ”system”, ”content”: system_message) messages.append(”role”: ”user”, ”content”: prompt)
try: formatted_prompt = self.tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True) return formatted_prompt except Exception as e: Fallback to manual formatting if system_message: return f”¡s¿[INST] system_messageprompt [/INST]” else: return f”¡s¿[INST] prompt [/INST]”

3 Phase 3: Fine-tuning Pipeline

3.1 Fine-tuning Data Strategy

User Prompt

Let’s look at fine-tuning data collection. I think we should use a mix of datasets available on the internet. What do you recommend?

AI Response

Great question! For fine-tuning our LaTeX formatting model, a mixed dataset approach is definitely the way to go. Here's my recommended strategy:
Recommended Data Sources Mix:

- **Academic Paper Datasets (60%)** - High quality, structured
  - ArXiv LaTeX sources
  - PeerRead academic papers
  - S2ORC semantic scholar corpus
- **Educational Content (25%)** - Closest to our use case
  - Math Stack Exchange Q&A pairs
  - University course materials
  - Jupyter notebooks with markdown/math
- **Synthetic Data (15%)** - Controlled, customizable
  - GPT-4 generated (text, LaTeX) pairs
  - Template-based generation
  - Math textbook problems → solutions

Specific Dataset Recommendations:

- Start with ArXiv via Hugging Face datasets
- Math Stack Exchange for real problems
- Create synthetic data with Groq for immediate testing

Data Format Structure: "input_text": "Solve the equation: x$^2 + 2x + 1 = 0$",*"output_latex" : "Solvetheequation* :x$^2 + 2x + 1 = 0$", "metadata": "source": "synthetic", "difficulty": "easy"

This gives us balanced, high-quality training data. Would you like me to implement the data collection pipeline?

4 Phase 4: Technical Implementation Issues

4.1 Configuration Type Error

User Prompt

```
/content/STUDY-BUDDY# python train_qlora.py
...
 QLoRA fine-tuning failed: '<=' not supported between instances of 'float' and 'str
Traceback (most recent call last):
  File "/content/STUDY-BUDDY/train_qlora.py", line 286, in <module>
    main()
  File "/content/STUDY-BUDDY/train_qlora.py", line 272, in main
    model_path = trainer.train()
  ...
TypeError: '<=' not supported between instances of 'float' and 'str'
```

AI Response

Ah! The classic YAML type conversion issue. The problem is that YAML is parsing `2e-4` as a string instead of a float. Let me fix this:
Root Cause: YAML parsing learning rate as string
Solution: Explicit type conversion in configuration
Key Fixes:

1. Update configuration to use explicit floats: `0.0002` instead of `2e-4`
2. Add type validation in ConfigLoader
3. Create a quick fix script with hardcoded parameters
4. Add configuration testing utility

Fixed Configuration: training: learning_rate: 0.0002 Explicit float instead of 2e-4 num_train_epochs: 3 per_device_train_batch_size: 1 ... other parameters as explicit numbers

ConfigLoader Type Fix: def _load_config(self, config_path: str) -¿ Dict[str, Any]: with open(config_path, 'r') as f: config = yaml.safe_load(f) Ensure numeric values are properly typed training_config = config['fine_tuning']['training'] training_config['learning_rate'] = float(training_config['learning_rate']) training_config['num_train_epochs'] = int(training_config['num_train_epochs']) ... convert all numeric parameters return config

This is a common issue with scientific notation in YAML - the explicit float format will resolve it. I'll provide the complete fixed implementation.

5 Summary of Key Technical Decisions

5.1 Model Selection Justification

- **Groq (Llama 3.3 70B) for RAG**: High-quality reasoning, fast inference, cost-effective for research
- **Mistral 7B Instruct for Formatter**: Instruction-tuned, efficient, open-source, fine-tunable
- **QLoRA for Fine-tuning**: 4-bit quantization, memory-efficient, maintains base capabilities

5.2 Architecture Decisions

- **Multi-agent Design**: Specialized agents for different tasks with clear interfaces
- **Mixed Inference Strategy**: Cloud API for complex reasoning, local models for specialized tasks
- **Modular Configuration**: YAML-based, easy experimentation and deployment

6 Conclusion

The interaction logs demonstrate a complete AI system development lifecycle:

- **Conception**: Initial idea refinement and feasibility analysis
- **Architecture**: Technical design and model selection
- **Implementation**: Code development and issue resolution
- **Optimization**: Fine-tuning and performance improvement
- **Evaluation**: Comprehensive testing and metrics collection