Technical Report: AI-Powered GitHub Self-Analysis

Data Science Intern Assignment | Khushi Patel (Khushipatel27)

1. Local LLM Experience and Challenges

This project used Ollama as the local LLM runtime on a Windows machine. Two models were used: Llama 3.1 (8B parameters) as the primary model and Mistral (7B parameters) as the comparison model. Both ran on consumer hardware without dedicated GPU acceleration. The Ollama API runs locally on port 11434 and was accessed through a custom Python module (llm_analysis.py) using the requests library.

The setup process was smooth. Ollama was installed using the official Windows installer, and both models were pulled with simple commands (ollama pull llama3.1 and ollama pull mistral). Each model download took roughly 5-10 minutes. The REST API format closely mirrors OpenAI's chat completions endpoint, so building the Python wrapper was straightforward.

Several challenges emerged during the project. First, context window limits required careful data truncation. With 38 commits and 7 READMEs, the raw data fit within limits, but for larger profiles this would require sampling strategies. Second, response formatting was inconsistent. Despite explicit instructions to avoid markdown, Llama 3.1 still occasionally produced bold text and headers. Adding the no-markdown instruction to both the system prompt and user prompt reduced but did not fully eliminate this behavior. Third, inference speed was a bottleneck. Llama 3.1 averaged 6.8 tokens/sec while Mistral averaged 9.4 tokens/sec. Individual tasks took between 27 and 71 seconds, making the full 7-task analysis take over 5 minutes total. Finally, there was a clear quality-speed tradeoff between the two models. Llama 3.1 consistently produced more detailed and nuanced analysis (averaging 378 tokens per response) while Mistral was faster but more concise (averaging 362 tokens per response).

2. Cost Analysis

Provider	Model	Cost/1K Input	Cost/1K Output	Total Project Cost
Ollama (local)	Llama 3.1 (8B)	$0.00	$0.00	$0.00
Ollama (local)	Mistral (7B)	$0.00	$0.00	$0.00

Since the project used only local LLMs through Ollama, the total monetary cost was zero. Across 7 analysis tasks (with 3 tasks run on both models for comparison), the project processed approximately 10 prompts generating roughly 3,700 total output tokens. The only real cost was compute time on the local machine, with the full pipeline taking around 8 minutes of inference time. The hardware requirement is modest: Ollama needs at least 8GB of RAM for 7-8B parameter models. No cloud API keys were needed.

3. Top Insights About Myself

Insight 1 - Positive Commit Sentiment: Sentiment analysis classified 60% of my 38 commits as positive, 32% as neutral, and 28% as frustrated, with zero negative commits. The frustrated commits clustered around commits 8-10, likely corresponding to deadline pressure. The overall tone shifted from neutral to more positive over time, suggesting growing comfort with the development workflow.

Insight 2 - Strong Python Focus with Narrow Breadth: Python and Jupyter Notebook dominate my 9 repositories. The skill extraction rated my Python proficiency as advanced and identified domain expertise in data management, machine learning (LSTM, SARSA, YOLOv8), and data visualization (Tableau). However, there is no representation of web development, JavaScript, Java, or cloud technologies, limiting portfolio breadth.

Insight 3 - Documentation Quality Varies Widely: LLM-based README evaluation scored my repos between 6 and 10 on clarity, but most repos received N/A for setup instructions and usage examples. The Movie-Recommendation repo scored highest (clarity: 9, setup: 10, examples: 9, completeness: 10) while course repos like DSCI551 scored lowest on completeness (6/10). Three out of five evaluated repos had no setup instructions at all.

Insight 4 - Portfolio Clusters Reveal Gaps: Topic clustering identified five thematic groups: data management, machine learning/computer vision, data visualization, business applications, and advanced analytics. K-Means clustering confirmed this by placing 6 of 9 repos in a single dominant cluster, meaning most projects are similar in scope. The LLM specifically flagged gaps in full-stack development, cloud services, DevOps, and containerization.

Insight 5 - Accelerating but Sparse Activity: Time series analysis showed a slightly increasing trend of +0.3 commits per month, though the R-squared value of 0.065 indicates high variance. My GitHub account was created in October 2021 but the first project did not appear until April 2025, suggesting a long inactive period followed by a burst of activity with 9 repos created in about 6 months.

4. What I Would Do With More Time

Build a RAG System: I would create a Retrieval-Augmented Generation pipeline that indexes all my code files and documentation into a vector database like ChromaDB or FAISS. This would let me query my own codebase in natural language, for example asking 'which of my projects uses LSTM' or 'show me how I handled CSV loading across repos'. The embeddings would be generated locally using Ollama's embedding models.

Improve Forecasting with ARIMA/Prophet: The linear regression model used for commit forecasting had an R-squared of only 0.065, meaning it explains very little of the variance. With more time, I would implement Facebook Prophet or ARIMA models that can capture seasonality patterns (like semester-based coding bursts) and provide confidence intervals on predictions.

Collect More Data: With only 9 public repos and 38 commits, several analyses were limited. I would expand the dataset by including private repository data (with a token), pull request history, issue comments, and code review activity. This would give a much richer picture of collaboration patterns and development habits.

Fine-Tune a Model: I would fine-tune a small language model on my commit messages and README text to generate documentation in my personal writing style. This would require more training data than my current 38 commits, so I would combine commit data with README text and code comments. I would use LoRA (Low-Rank Adaptation) for parameter-efficient fine-tuning on the Llama 3.1 base.

Add Code Quality Analysis: I would pull actual source code files from my repos and have the LLM perform code quality reviews, complexity analysis, and identify common anti-patterns. This was listed as an optional analysis task but was not completed due to time constraints. It would require fetching file contents through the GitHub API and managing larger context windows.