DAY 10

Building Smart AI Code Agents Using Gemini

Today was an exciting milestone in my training journey — I explored how to build **real-world AI agents** using Gemini that can **generate, evaluate, improve, and debug code.**Rather than building just one monolithic tool, I built **four specialized micro-agents**, each handling a different phase of the software development cycle. These agents not only demonstrate Gemini's flexibility but also reflect a mini end-to-end **AI-assisted development workflow**-just like a supervised ML pipeline!

Dataset Creation + Gemini Evaluation (Supervised Code Generation)

We started by simulating a typical machine learning setup — using **prompt–completion pairs** to create a dataset. These pairs are like mini conversations between a user and an AI assistant: the prompt asks a question, and the completion provides the expected Python solution.

Steps I followed:

- Created a list of **5 prompt Python code** examples (like "check palindrome", "find max", etc.)
- Convert them into a **CSV file** (code_generation_dataset.csv) to keep a readable format.
- Split the data into:
  - train_data.json – for training simulation
  - test_data.json – for evaluation
- Used **Gemini Flash model** to test the prompts in the test set.

- ❖ Compared Gemini's output with my reference solution line-by-line.

Why it matters:

This approach mirrors real ML fine-tuning workflows. Even though Gemini isn't being fine-tuned here, I'm testing it as if it were — **treating Gemini like a trained model**, validating its ability to generalize to unseen prompts.

Example Prompt:

Prompt: Write a function to count vowels in a string.

Expected Output:

```
def count_vowels(s):
    return sum(1 for c in s.lower() if c in 'aeiou')
```

This dataset-driven flow gave me an understanding of how **AI-powered coding assistants** can be evaluated like real ML models.

Interactive Prompt-to-Code Generator (Live Code Assistant)

In this agent, I became the test set.

I manually typed in my own code prompts, like "write a function to reverse a list," and also provided the expected code. Gemini responded in real-time — just like a co-pilot sitting beside me.

Flow:

- User types a prompt
- Gemini instantly writes the code
- I compare Gemini's answer with what I expected

Why it feels powerful:

It's **live**, **flexible**, and totally human-in-the-loop. This would be perfect in:

- A VS Code plugin
- Coding interview practice
- Auto-evaluation system in ed-tech apps

Code Optimizer (Gemini as a Smart Refactor Bot)

In this project, I asked Gemini to **improve messy or inefficient Python code**.

Instead of writing new code, the user pastes existing logic — and Gemini rewrites it for:

- Clean formatting
- Pythonic style
- Simplicity and readability

Example:

Messy code:

def add(a, b): return a + b if a != None and b != None else 0

Gemini output:

```
def add(a, b):
    return a + b if a and b else 0
```

Clean, efficient, and easy to read. Perfect use case for developers who want automated refactoring or beginner coders who want to learn best practices.

Python Debugging Agent (Explain + Fix Code)

This was my favorite. You paste buggy code, and Gemini:

1. Detects and explains the issue
2. Provides a corrected version
3. (Optionally) gives a working example

Debug prompt sample:

```
def greet(name): print("Hello " + name if name != None else "Hi")
```

Gemini Fix:

```
def greet(name):
    if name:
        print("Hello " + name)
    else:
        print("Hi")
```

This project clearly shows Gemini's ability to reason about syntax, runtime errors, and even logical flaws.

Summary of All Four Agents

Agent	Purpose
Dataset Evaluation	Simulate a machine learning workflow with train/test split
Live Code Generator	Real-time Gemini prompt-to-code testing
Code Optimizer	Improve bad or unclean Python code
Code Debugger	Diagnose and fix errors with explanation

What I Learned

- How to build **modular agents** for different dev tasks using Gemini
- How prompt design impacts output clarity, logic, and creativity
- How to simulate supervised ML pipelines using JSON data formats
- How to evaluate LLMs both **programmatically** and **interactively**
- That Gemini can truly act as a teammate — not just a tool

Real-World Use Cases This Taught Me

- AI mentors in coding platforms (like LeetCode, HackerRank)
- VS Code agents that suggest optimizations or fix bugs

- Data-backed evaluation of AI models in development
- Continuous integration bots that test prompts as unit tests

Final Takeaways

Today wasn't just about writing code — it was about **engineering with AI**.

I explored how to **design, evaluate, and refine** multiple specialized AI agents that can handle real-world coding workflows. Gemini wasn't just answering prompts — it was acting like a **team member**, thinking through tasks with me.

Key Learnings:

- **Prompt quality = output quality**: Small tweaks in prompts led to big improvements in results.
- **LLMs can be structured like ML models**: With training-style data (JSON), I tested Gemini just like a supervised learning setup.
- **Code tasks are modular**: Writing, reviewing, optimizing, and debugging code can each be handled by a **dedicated micro-agent**.
- **Gemini is reliable across coding contexts**: Whether it's generating fresh code, catching bugs, or improving logic, Gemini proved adaptable and accurate.
- **AI is more powerful when human-in-the-loop**: Especially in the live code generator, real-time feedback makes the experience collaborative and educational.

Practical Takeaways:

- I now understand how to evaluate LLMs using structured datasets
- I can turn prompt-response pairs into train/test workflows
- I know how to build AI tools that go beyond chat — actual **agents with roles**
- I see how to build smart assistants for real dev tasks (and not just toy problems)

This project showed me that AI isn't replacing developers — it's becoming their **most helpful teammate**.Whether it's for students, startups, or seasoned engineers — building Gemini-powered agents can:

- Speed up dev time
- Improve code quality
- Make learning interactive
- Turn manual effort into smart automation

"I built a real AI workflow — not just code, but intelligence."