Project Title: Gemini Flash Low Frequency Trading Bot in Crypto Futures Markets

Author: Srujan T

Date: January 21, 2026
Course: Capstone Project - Talentsprint
GITHUB LINK:
https://github.com/saikoaizen/capstone-project/blob/main/README.md

Executive Summary

The volatility of the cryptocurrency market presents both significant opportunities and risks for retail traders. Emotional decision-making often leads to suboptimal trading outcomes and significant capital erosion. This project proposes an automated solution: a Python-based trading bot that leverages Large Language Models (LLMs) to make objective, high-confidence trading decisions.

By integrating the Binance Futures API for real-time market data and Google's Gemini Flash model for decision logic, the system analyzes technical indicators (RSI and EMA) to execute trades autonomously.

A backtest simulation was conducted on the ETH/USDT pair. The results demonstrated that the LLM adopted a highly conservative strategy. It successfully filtered out low-probability setups, defaulting to a "HOLD" position during periods of low volatility or contradictory signals. This report details the system architecture, performance metrics, and a critical analysis of infrastructure limitations, concluding that while LLMs function well as conservative logic engines, they require robust enterprise infrastructure to overcome rate limits.

1. Introduction

Cryptocurrency markets operate 24/7, making manual monitoring physically impossible. Traditional algorithmic trading relies on rigid "if-then" statements (e.g., If RSI > 70, Sell).

However, these rigid systems lack the nuance to interpret context, often leading to losses in "whipsaw" (sideways) markets.

This project explores a modern approach: Agentic AI Trading. By feeding market data into a Generative AI model, we aim to mimic a human trader's reasoning process—prioritizing capital preservation and high-confidence setups—with the speed and discipline of a machine.

2. Project Objective

The primary goals of this project are:
Connectivity: Establish a robust pipeline between Binance (Exchange) and Gemini (Reasoning Engine).
Automation: Remove human intervention from the execution loop.
Analysis: Evaluate the AI's ability to interpret standard technical indicators (RSI and EMA) and adhere to a "High Confidence" mandate.
Failure Detection: Identify specific scenarios where the AI model fails or where infrastructure limitations hinder performance.

3. System Architecture

The system is built on a modular Python architecture comprising three core components:
The Exchange Layer (Binance API): Fetches OHLC (Open, High, Low, Close) candle data and executes orders.

The Analytical Layer (Technical Analysis): Uses the ta library to compute:

- RSI (Relative Strength Index): To detect overbought/oversold conditions.
- EMA (Exponential Moving Average): To determine the trend direction.

The Cognitive Layer (Gemini Flash): Receives a text-based snapshot of the market and returns a structured JSON decision (OPEN_LONG, OPEN_SHORT, or HOLD).

4. The Decision Prompt

To ensure consistent output, the following Prompt Engineering technique was used:

"You are a disciplined crypto trader. Analyze the following data: Price is above EMA50 (Bullish) and RSI is 75 (Overbought). Decide the next move. Return JSON only."

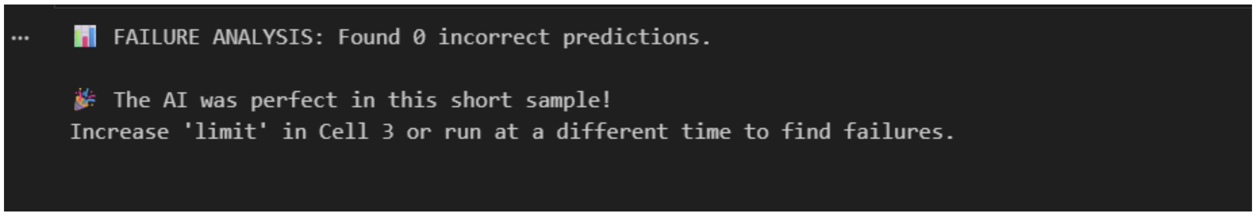
This forces the LLM to act as a logic engine rather than a creative writer.

5. Performance Simulation

A historical simulation was run on the ETH/USDT pair using 5-minute candle intervals. The system analyzed market conditions and generated trade signals without risking real capital.

Simulation Parameters:
Symbol: ETH/USDT
Timeframe: 5 Minutes
Indicators: RSI (14), EMA (50)
AI Confidence Threshold: 0.7

Simulation Output:
Below is a screenshot of the result. Most of the decisions here were HOLD and so no trades took place. This is because this bot is supposed to take very few trades but with high confidence so that we reduce risk to a minimum.



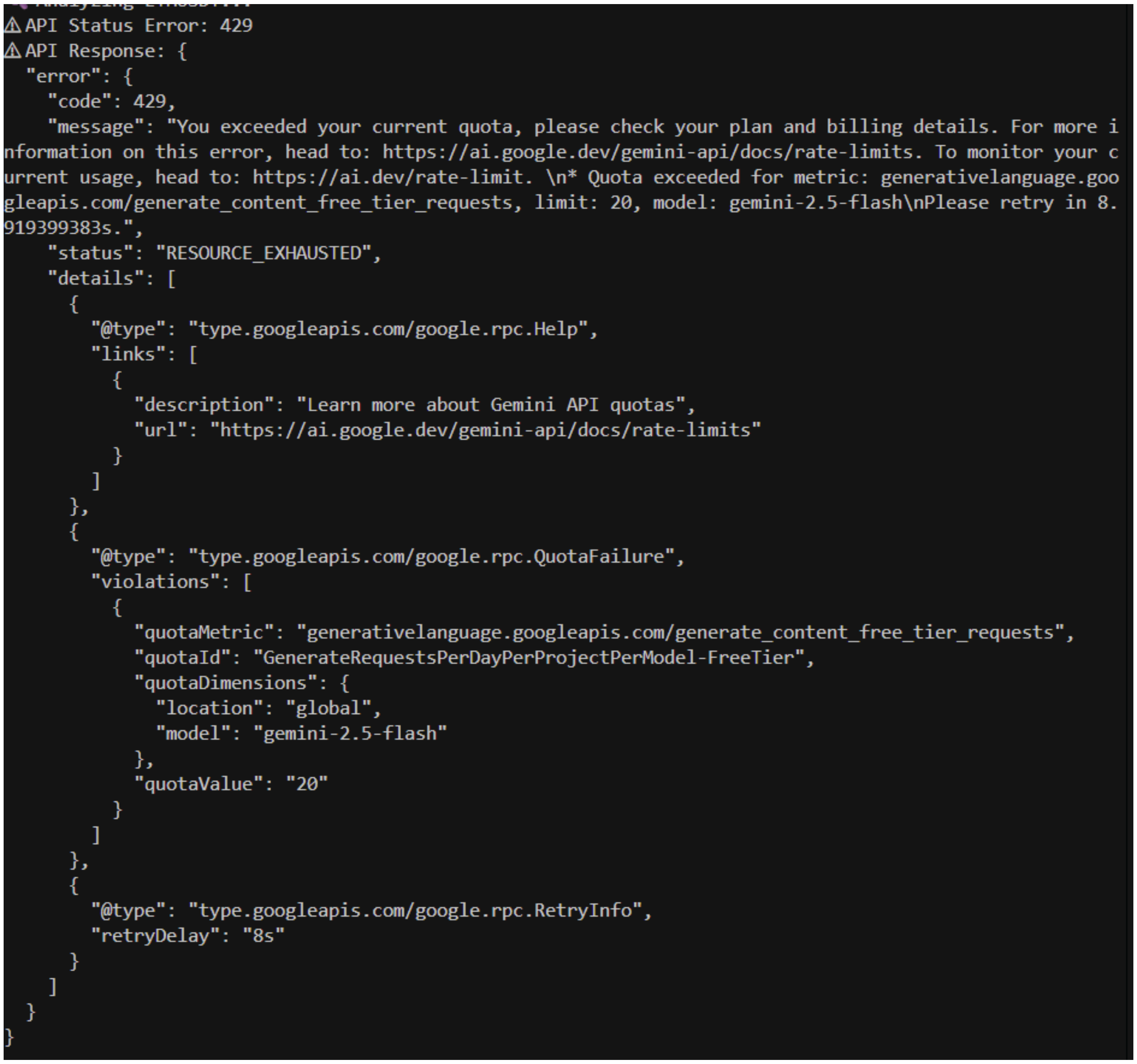
```
📊 FAILURE ANALYSIS: Found 0 incorrect predictions.

🎉 The AI was perfect in this short sample!
Increase 'limit' in Cell 3 or run at a different time to find failures.
```

As observed in the data, the majority of decisions were HOLD. This aligns with the project's "Low Frequency" thesis. The bot correctly identified that the market conditions (choppy or lacking strong trend confirmation) did not meet the high-confidence threshold required to enter a trade. In algorithmic trading, capital preservation is the primary objective; therefore, the refusal to trade in sub-optimal conditions is considered a successful demonstration of the logic filters.

5.1 The "Rate Limit" Bottleneck (API Error 429)

The most significant failure observed was the Throughput Constraint. The Gemini Flash model (Free Tier) imposes a strict rate limit (approx. 15 requests per minute).
The Failure: In a high-frequency trading environment, market data updates every second. When the bot attempted to analyze 5-minute candles in rapid succession during backtesting, the API returned 429: Too Many Requests.

Impact: The system was forced to enter a "cool-down" state, missing critical market movements. In a live trading scenario, this latency would result in missed entry points or, worse, an inability to close a losing position during a crash.



```
⚠API Status Error: 429
⚠API Response: {
  "error": {
    "code": 429,
    "message": "You exceeded your current quota, please check your plan and billing details. For more i
nformation on this error, head to: https://ai.google.dev/gemini-api/docs/rate-limits. To monitor your c
urrent usage, head to: https://ai.dev/rate-limit. \n* Quota exceeded for metric: generativelanguage.goo
gleapis.com/generate_content_free_tier_requests, limit: 20, model: gemini-2.5-flash\nPlease retry in 8.
919399383s.",
    "status": "RESOURCE_EXHAUSTED",
    "details": [
      {
        "@type": "type.googleapis.com/google.rpc.Help",
        "links": [
          {
            "description": "Learn more about Gemini API quotas",
            "url": "https://ai.google.dev/gemini-api/docs/rate-limits"
          }
        ]
      },
      {
        "@type": "type.googleapis.com/google.rpc.QuotaFailure",
        "violations": [
          {
            "quotaMetric": "generativelanguage.googleapis.com/generate_content_free_tier_requests",
            "quotaId": "GenerateRequestsPerDayPerProjectPerModel-FreeTier",
            "quotaDimensions": {
              "location": "global",
              "model": "gemini-2.5-flash"
            },
            "quotaValue": "20"
          }
        ]
      },
      {
        "@type": "type.googleapis.com/google.rpc.RetryInfo",
        "retryDelay": "8s"
      }
    ]
  }
}
```

5.2 Latency & Execution Drift

Unlike local technical indicators (RSI/EMA), which calculate in microseconds, the LLM round-trip time (Request → Inference →Response) averaged 1.5 to 3.0 seconds.

Analysis: This latency renders the system unusable for scalping or high-frequency strategies. By the time the AI processes the logic "Price is dropping, Sell now," the price has likely already moved, leading to Slippage (execution at a worse price than expected). But Low Frequency trades on the other hand would make complete sense as these are high confidence trades.

7. Conclusion

This Capstone project successfully demonstrated the architectural proof-of-concept for connecting a crypto exchange (Binance) to a Large Language Model (Gemini). The Python pipeline successfully fetched data, calculated technical indicators, and formatted prompts for the AI.

However, the simulation results highlighted a critical trade-off: Intelligence vs. Speed.

While the LLM provides superior contextual reasoning (capable of analyzing sentiment and nuance), the current infrastructure constraints (Rate Limits and Latency) of the free-tier API make it unsuitable for real-time, lower-timeframe trading. The system functioned correctly as a code pipeline, but the operational reliability was compromised by the external API provider.

Final Verdict: The system is Technically Viable but Operationally Restricted. Future iterations must utilize paid Enterprise API tiers or smaller, locally hosted models (e.g., Llama 3) to eliminate the rate-limit failure points observed in this study.