# Designing an Autonomous AI Crypto & Polymarket Trading Bot

The goal is to transform the existing CCXT-based arbitrage bot into a *fully autonomous AI trading system*. This involves integrating a large language model (LLM) “agent” that uses our bot’s infrastructure as *tools* under its control. The agent will use skill definitions (e.g. **skill.md** files) to understand its capabilities and best practices[[1]](https://www.mintlify.com/blog/skill-md#:~:text=What%20belongs%20in%20skill), and will operate on human funds with safety checks. Core features include persistent memory (short-term context in Redis, long-term state in SQLite), multi-agent coordination (analyst, trader, risk management roles), and automated *quests* (hourly/daily tasks and funding milestones) to guide its activity. The Telegram bot remains the user interface, letting users link exchange & Polymarket keys, view portfolios, and watch a live stream of the agent’s trades.

## Core AI Model and Agent Framework

We would use a state-of-the-art LLM as the agent’s “brain.” As of early 2026, OpenAI’s **GPT-5.3 (Codex)** and Anthropic’s **Claude Opus 4.6** have been announced for coding and reasoning[[2]](https://www.interconnects.ai/p/opus-46-vs-codex-53#:~:text=Last%20Thursday%2C%20February%205th%2C%20both,rather%20how%20to%20assess%2C%20live). These models excel at complex planning and code generation, which is ideal for trading logic. If a sufficiently capable open-source model is available (e.g. a 4–20B parameter LLaMA3 or Mistral variant), it can be run locally on the M1 Pro using tools like Apple’s MLX-LM framework[[3]](https://medium.com/@lukekerbs/goodbye-api-keys-hello-local-llms-how-i-cut-costs-by-running-llm-models-on-my-m3-macbook-a3074e24fee5#:~:text=MLX,without%20needing%20a%20dedicated%20GPU). MLX-LM leverages the M1’s Neural Engine and supports quantized LLMs (e.g. Mistral, Llama) so one can “cut the cord” from cloud APIs while preserving performance[[3]](https://medium.com/@lukekerbs/goodbye-api-keys-hello-local-llms-how-i-cut-costs-by-running-llm-models-on-my-m3-macbook-a3074e24fee5#:~:text=MLX,without%20needing%20a%20dedicated%20GPU).

The agent will be *multi-agent* or at least multi-role. Research on **LLM trading agents** shows they use layered memory and specialized sub-agents to handle complex markets[[4]](https://www.emergentmind.com/topics/llmtradingagent#:~:text=,returns%2C%20and%20transparent%20decision%20rationales). For example, such systems may include **Analyst** agents (market research), **Trader** agents (order execution), and **Risk** agents (stop-loss and capital allocation) that debate and approve trades[[5]](https://www.emergentmind.com/topics/llmtradingagent#:~:text=,and%20approve%20final%20trade%20actions). By integrating textual data (news, tweets), numerical data (price feeds, order books), and even visual charts, these agents adapt strategies in real time and provide explainable rationales[[6]](https://www.emergentmind.com/topics/llmtradingagent#:~:text=,returns%2C%20and%20transparent%20decision%20rationales)[[7]](https://www.emergentmind.com/topics/llmtradingagent#:~:text=3,with%20External%20Tools).

All existing bot functions (e.g. CCXT trading calls, portfolio queries, Polymarket API) become *tools* the AI can invoke. We’ll follow the “tool-calling” paradigm: the LLM is given descriptions of tools (via **skill.md** and prompt instructions) and can call them as needed. A *skill.md* file is a concise markdown spec that tells the agent what each tool does, plus any usage rules[[1]](https://www.mintlify.com/blog/skill-md#:~:text=What%20belongs%20in%20skill). For example, one could define a get\_price(symbol) tool or a place\_order tool and describe them in skill.md. This ensures the agent always has up-to-date context on our system’s capabilities[[1]](https://www.mintlify.com/blog/skill-md#:~:text=What%20belongs%20in%20skill). Frameworks like LangChain (and its LangGraph extension) or Microsoft’s AutoGen can orchestrate such agents, managing tool calls and memory[[8]](https://www.leanware.co/insights/llm-agent-architecture-guide#:~:text=These%20frameworks%20provide%20tools%20to,build%20and%20manage%20LLM%20agents).

  
*Figure: The agent operates in a developer-like environment, invoking code and tools. It has access to all system functions (e.g. trading APIs) much like a programmer would use code editors and scripts.* The AI’s “workspace” could even include a code execution context (as above) for advanced actions (e.g. backtesting, calling shell scripts, or querying a local ML model). This design lets a *generalist agent* solve many problems with few explicit tools, instead using the filesystem and skill modules to expand capabilities[[9]](https://blog.langchain.com/using-skills-with-deep-agents/#:~:text=Anthropic%20skills%20just%20follows%20this,md%20file)[[10]](https://blog.langchain.com/using-skills-with-deep-agents/#:~:text=The%20second%20benefit%20is%20reduced,of%20skills%20for%20encoding%20actions).

## Data Sources & Trading Strategy

The agent will ingest live market data from crypto exchanges and Polymarket. For cryptocurrencies, we continue using CCXT (in Go or Python) to fetch prices, order books, and execute trades. For prediction markets, we integrate Polymarket’s API. Polymarket is an Ethereum/Polygon-based market requiring a USDC-funded wallet; it uses off-chain order matching with on-chain settlement[[11]](https://www.quantvps.com/blog/automated-trading-polymarket?srsltid=AfmBOopV4hIva8hzXyqmRe0t6cqMWsmDHgKRAVecXfIvujWItE5rwPZ1#:~:text=Automated%20trading%20on%20Polymarket%20,competitive%20nature%20of%20the%20platform)[[12]](https://www.quantvps.com/blog/automated-trading-polymarket?srsltid=AfmBOopV4hIva8hzXyqmRe0t6cqMWsmDHgKRAVecXfIvujWItE5rwPZ1#:~:text=To%20get%20started%2C%20you%E2%80%99ll%20need,ownership%20verification%20from%20trading%20execution). In practice, the user must supply a Polygon-compatible wallet and approve USDC (a “funder” address) for trading. Polymarket API keys (HMAC-signed) plus the private-key-signed user address together authorize trades[[12]](https://www.quantvps.com/blog/automated-trading-polymarket?srsltid=AfmBOopV4hIva8hzXyqmRe0t6cqMWsmDHgKRAVecXfIvujWItE5rwPZ1#:~:text=To%20get%20started%2C%20you%E2%80%99ll%20need,ownership%20verification%20from%20trading%20execution).

Data from these sources flows into the agent’s memory. The LLM uses short-term cache (e.g. Redis) to hold recent events and conversation context, and long-term memory (SQLite tables) to store persistent state (user profiles, risk settings, historical performance)[[13]](https://levelup.gitconnected.com/build-an-ai-agent-with-long-term-memory-266fe5b7bcc1?gi=a863085d52fd#:~:text=Step%201%3A%20Building%20The%20Long,Memory)[[14]](https://redis.io/blog/build-smarter-ai-agents-manage-short-term-and-long-term-memory-with-redis/#:~:text=AI%20agent%20memory%20is%20crucial,more%20coherent%20and%20personalized%20responses). For example, SQLite can store a table of *user sessions* (portfolio targets, risk tolerance) as a structured “source of truth”[[13]](https://levelup.gitconnected.com/build-an-ai-agent-with-long-term-memory-266fe5b7bcc1?gi=a863085d52fd#:~:text=Step%201%3A%20Building%20The%20Long,Memory), while Redis caches the latest market ticks and dialogue. This hybrid memory ensures the agent “remembers” trends or its own past actions, avoiding repeat mistakes[[15]](https://redis.io/blog/build-smarter-ai-agents-manage-short-term-and-long-term-memory-with-redis/#:~:text=Why%20does%20memory%20matter%3F)[[13]](https://levelup.gitconnected.com/build-an-ai-agent-with-long-term-memory-266fe5b7bcc1?gi=a863085d52fd#:~:text=Step%201%3A%20Building%20The%20Long,Memory).

The trading strategies themselves are codified in prompts and training. We can implement or teach strategies like scalping, arbitrage, or sentiment-driven bets. For instance, in Polymarket one can arbitrage YES/NO shares whose prices don’t sum to 1[[16]](https://www.quantvps.com/blog/automated-trading-polymarket?srsltid=AfmBOopV4hIva8hzXyqmRe0t6cqMWsmDHgKRAVecXfIvujWItE5rwPZ1#:~:text=,time). Risk controls (stop-loss orders, daily loss caps) are enforced by the agent’s safety rules and by dedicating a *Risk Manager* sub-agent to veto dangerous trades. All trades and rationales are logged to the portfolio (and streamed to Telegram) so users can audit decisions.

  
*Figure: Real-time market data (like this candlestick chart) feeds into the agent’s analysis. The AI processes both numerical prices and textual news to adapt its trading decisions.* In practice, the agent uses both **numerical time-series** (candlestick/volume) and **unstructured data** (social sentiment, news headlines) as inputs[[4]](https://www.emergentmind.com/topics/llmtradingagent#:~:text=,returns%2C%20and%20transparent%20decision%20rationales)[[7]](https://www.emergentmind.com/topics/llmtradingagent#:~:text=3,with%20External%20Tools). It may even analyze chart images with a vision-capable LLM extension to spot patterns (e.g. support/resistance). By continuously scanning these inputs, the agent generates trade proposals, then polls other sub-agents (bullish vs bearish) and risk checks before executing any order.

## System Architecture & Persistence

The backend combines **Golang** services (existing CCXT modules, Telegram bot server) with an LLM-agent loop (likely in Python for leveraging agent frameworks). We use a SQLite database to store persistent state (user accounts, API keys, portfolio history, quest progress)[[13]](https://levelup.gitconnected.com/build-an-ai-agent-with-long-term-memory-266fe5b7bcc1?gi=a863085d52fd#:~:text=Step%201%3A%20Building%20The%20Long,Memory). Redis serves as a fast in-memory store for short-term context: conversation history, real-time indicators, rate limiting, and job coordination[[14]](https://redis.io/blog/build-smarter-ai-agents-manage-short-term-and-long-term-memory-with-redis/#:~:text=AI%20agent%20memory%20is%20crucial,more%20coherent%20and%20personalized%20responses). For example, each user session could have a Redis key holding the current prompt context, and periodic persistence checkpoints flush context to SQLite.

To handle *autonomous workflows*, we need a jobs system. The agent runs continuously or on a schedule, launching sub-tasks (quests) at specified intervals. For instance, an hourly quest might be “scan all markets for arbitrage opportunities,” while a daily quest could be “rebalance portfolio according to target allocations.” We can use a task queue (backed by Redis) or Go routines with scheduled ticks to manage concurrency. Each quest’s outcome (e.g. trades made, profit/loss, new memory entries) is appended to the database. This ensures parallel sub-jobs don’t conflict and that state (like cumulative earnings or milestones) persists across runs.

Security and isolation are critical: all exchange and Polymarket API keys (and any private keys) must be encrypted at rest. The agent only uses these credentials to sign real transactions; sensitive actions (like withdrawing funds) remain manual or require extra safeguards.

## Telegram User Interface

The **Telegram bot** continues as the human-facing interface. Users can issue commands to **link accounts** (e.g. /connect\_binance API\_KEY SECRET, or /connect\_polymarket WALLET\_ADDRESS API\_SECRET). The bot UI provides authentication flows where possible and stores tokens in the database. Once connected, users can use Telegram commands or buttons to view their portfolio, current balances, and profit/loss. Importantly, every trade the AI executes (entry/exit of positions) is immediately reported in the chat, creating a **live transaction stream**. This transparency lets the user monitor the agent’s actions and even override or pause it if needed.

  
*Figure: The Telegram bot serves as the user console. Users link wallets/API keys and see the AI’s transactions (e.g. in a chat with BotFather or a custom bot). This screenshot illustrates a Telegram chat on a phone.* The bot can also display rich info like charts or portfolio summaries (e.g. using Telegram’s charting keyboard or image previews). Authenticated model API keys (e.g. for OpenAI) can be stored per-user so each person can choose which AI model or API to use for their bot. In short, the UI remains a chat, but with custom commands for **/status**, **/balance**, **/tradehistory**, etc., pulling data from our SQLite-backed state.

## Implementation Notes

Most components already exist: CCXT trading logic (in Go), and the Telegram bot (TypeScript or Go). We’d extend these by launching an **LLM agent service**. This could be a Python service using LangChain/AutoGen or an in-house loop. It would load skill.md files and have access to our trading libraries as callable tools. On macOS M1 Pro, one could run open-source LLMs locally using MLX-LM (as shown above[[3]](https://medium.com/@lukekerbs/goodbye-api-keys-hello-local-llms-how-i-cut-costs-by-running-llm-models-on-my-m3-macbook-a3074e24fee5#:~:text=MLX,without%20needing%20a%20dedicated%20GPU)). If using cloud, we’d call GPT-5.3/Claude Opus via API, but all environment data (market feeds, DB) stays private.

We must ensure **parallelism**: each user or strategy could have its own agent session (with isolated memory). Redis can coordinate locks so two sessions don’t trade the same asset simultaneously. Agents communicate via the DB or message queue (e.g. one agent’s output log may seed another’s input on the next step).

Popular open-source frameworks will help. LangChain’s agent tools let an LLM call Python functions directly (e.g. to fetch prices). CrewAI or SuperAGI could manage multi-agent workflows, and LlamaIndex or Pinecone could store embeddings for fast retrieval. However, given our budget (run on a Mac), we may start lightweight and expand.

**Summary:** We will build a multi-agent LLM-based trader: it uses skill definitions (skill.md) to know its tools, Redis for working memory and SQLite for long-term memory[[13]](https://levelup.gitconnected.com/build-an-ai-agent-with-long-term-memory-266fe5b7bcc1?gi=a863085d52fd#:~:text=Step%201%3A%20Building%20The%20Long,Memory)[[14]](https://redis.io/blog/build-smarter-ai-agents-manage-short-term-and-long-term-memory-with-redis/#:~:text=AI%20agent%20memory%20is%20crucial,more%20coherent%20and%20personalized%20responses). It integrates CCXT and Polymarket APIs as callable tools, and follows a collaborative workflow (analyst, trader, risk) for each trade[[4]](https://www.emergentmind.com/topics/llmtradingagent#:~:text=,returns%2C%20and%20transparent%20decision%20rationales)[[5]](https://www.emergentmind.com/topics/llmtradingagent#:~:text=,and%20approve%20final%20trade%20actions). The Telegram bot is revamped to manage keys and display portfolio/trades. By using cutting-edge models (GPT-5.3/Codex or open-source alternatives), the agent can reason about market patterns and code its own execution logic. All trade activity is logged for transparency, and automated “quests” keep the agent engaged on an hourly/daily schedule to pursue profit targets and account milestones.

**References:** State-of-the-art LLMs like GPT-5.3 and Claude Opus 4.6 have specialized code and reasoning improvements[[2]](https://www.interconnects.ai/p/opus-46-vs-codex-53#:~:text=Last%20Thursday%2C%20February%205th%2C%20both,rather%20how%20to%20assess%2C%20live). LLM trading agents typically employ layered memory and distinct analytic/trading roles[[4]](https://www.emergentmind.com/topics/llmtradingagent#:~:text=,returns%2C%20and%20transparent%20decision%20rationales)[[5]](https://www.emergentmind.com/topics/llmtradingagent#:~:text=,and%20approve%20final%20trade%20actions). Structured memory (SQLite) and fast cache (Redis) are recommended for persistent AI agent state[[13]](https://levelup.gitconnected.com/build-an-ai-agent-with-long-term-memory-266fe5b7bcc1?gi=a863085d52fd#:~:text=Step%201%3A%20Building%20The%20Long,Memory)[[14]](https://redis.io/blog/build-smarter-ai-agents-manage-short-term-and-long-term-memory-with-redis/#:~:text=AI%20agent%20memory%20is%20crucial,more%20coherent%20and%20personalized%20responses). Polymarket integration requires Polygon wallets with USDC and HMAC API keys[[12]](https://www.quantvps.com/blog/automated-trading-polymarket?srsltid=AfmBOopV4hIva8hzXyqmRe0t6cqMWsmDHgKRAVecXfIvujWItE5rwPZ1#:~:text=To%20get%20started%2C%20you%E2%80%99ll%20need,ownership%20verification%20from%20trading%20execution). And the **skill.md** format (used by Anthropic/LangChain) helps document agent capabilities and best practices[[1]](https://www.mintlify.com/blog/skill-md#:~:text=What%20belongs%20in%20skill). These references guided our design.

[[1]](https://www.mintlify.com/blog/skill-md#:~:text=What%20belongs%20in%20skill) skill.md: An open standard for agent skills

<https://www.mintlify.com/blog/skill-md>

[[2]](https://www.interconnects.ai/p/opus-46-vs-codex-53#:~:text=Last%20Thursday%2C%20February%205th%2C%20both,rather%20how%20to%20assess%2C%20live) Opus 4.6, Codex 5.3, and the post-benchmark era

<https://www.interconnects.ai/p/opus-46-vs-codex-53>

[[3]](https://medium.com/@lukekerbs/goodbye-api-keys-hello-local-llms-how-i-cut-costs-by-running-llm-models-on-my-m3-macbook-a3074e24fee5#:~:text=MLX,without%20needing%20a%20dedicated%20GPU) Goodbye API Keys, Hello Local LLMs: How I Cut Costs by Running LLM Models on my M3 MacBook | by Luke Kerbs | Medium

<https://medium.com/@lukekerbs/goodbye-api-keys-hello-local-llms-how-i-cut-costs-by-running-llm-models-on-my-m3-macbook-a3074e24fee5>

[[4]](https://www.emergentmind.com/topics/llmtradingagent#:~:text=,returns%2C%20and%20transparent%20decision%20rationales) [[5]](https://www.emergentmind.com/topics/llmtradingagent#:~:text=,and%20approve%20final%20trade%20actions) [[6]](https://www.emergentmind.com/topics/llmtradingagent#:~:text=,returns%2C%20and%20transparent%20decision%20rationales) [[7]](https://www.emergentmind.com/topics/llmtradingagent#:~:text=3,with%20External%20Tools) LLM Trading Agent Overview

<https://www.emergentmind.com/topics/llmtradingagent>

[[8]](https://www.leanware.co/insights/llm-agent-architecture-guide#:~:text=These%20frameworks%20provide%20tools%20to,build%20and%20manage%20LLM%20agents) LLM Agent Architecture Explained: Components and Applications

<https://www.leanware.co/insights/llm-agent-architecture-guide>

[[9]](https://blog.langchain.com/using-skills-with-deep-agents/#:~:text=Anthropic%20skills%20just%20follows%20this,md%20file) [[10]](https://blog.langchain.com/using-skills-with-deep-agents/#:~:text=The%20second%20benefit%20is%20reduced,of%20skills%20for%20encoding%20actions) Using skills with Deep Agents

<https://blog.langchain.com/using-skills-with-deep-agents/>

[[11]](https://www.quantvps.com/blog/automated-trading-polymarket?srsltid=AfmBOopV4hIva8hzXyqmRe0t6cqMWsmDHgKRAVecXfIvujWItE5rwPZ1#:~:text=Automated%20trading%20on%20Polymarket%20,competitive%20nature%20of%20the%20platform) [[12]](https://www.quantvps.com/blog/automated-trading-polymarket?srsltid=AfmBOopV4hIva8hzXyqmRe0t6cqMWsmDHgKRAVecXfIvujWItE5rwPZ1#:~:text=To%20get%20started%2C%20you%E2%80%99ll%20need,ownership%20verification%20from%20trading%20execution) [[16]](https://www.quantvps.com/blog/automated-trading-polymarket?srsltid=AfmBOopV4hIva8hzXyqmRe0t6cqMWsmDHgKRAVecXfIvujWItE5rwPZ1#:~:text=,time) Automated Trading on Polymarket: Bots, Arbitrage & Execution Strategies | QuantVPS

<https://www.quantvps.com/blog/automated-trading-polymarket?srsltid=AfmBOopV4hIva8hzXyqmRe0t6cqMWsmDHgKRAVecXfIvujWItE5rwPZ1>

[[13]](https://levelup.gitconnected.com/build-an-ai-agent-with-long-term-memory-266fe5b7bcc1?gi=a863085d52fd#:~:text=Step%201%3A%20Building%20The%20Long,Memory) Build an AI Agent with Long-Term Memory | by Fabio Hiroki | Feb, 2026 | Level Up Coding

<https://levelup.gitconnected.com/build-an-ai-agent-with-long-term-memory-266fe5b7bcc1?gi=a863085d52fd>

[[14]](https://redis.io/blog/build-smarter-ai-agents-manage-short-term-and-long-term-memory-with-redis/#:~:text=AI%20agent%20memory%20is%20crucial,more%20coherent%20and%20personalized%20responses) [[15]](https://redis.io/blog/build-smarter-ai-agents-manage-short-term-and-long-term-memory-with-redis/#:~:text=Why%20does%20memory%20matter%3F) Build smarter AI agents: Manage short-term and long-term memory with Redis | Redis

<https://redis.io/blog/build-smarter-ai-agents-manage-short-term-and-long-term-memory-with-redis/>