AYUSH RAWAT / PAPER TRADING SYSTEMS

Trading Bot Brotherhood Prompt Pack

The six copy-paste prompts, setup sequence, and critique guardrails for building a paper-trading bot with TypeScript, Node.js, real market data, and no live orders.

Prepared for careful experimentation. Not financial advice.

How to use this pack

Run the prompts in order. Prompts 01 through 03 ask questions and pause for confirmation. Prompts 04 through 06 create files, run replays, and report blockers. Keep the entire workflow in paper or test mode.

Before you start

- Use Claude Desktop, Claude Code, Claude in the browser, Cursor, or another MCP-capable client.
- Never paste API keys into chat or source code. Keep credentials in the MCP client configuration, an approved secrets store, or a server-side environment file.
- Start with market-data-only or restricted paper permissions.
- Do not skip the connection, backtest, or instructions-file gates.
- A blocked command must be reported as blocked. Never invent a result.

Expected project flow

01 connection report -> 02 TradingView backtest -> 03 trading_bot_instructions.md -> 04 raw baseline -> 05 raw versus memory comparison -> 06 documented experiment-ready build

Default example

BTCUSDT on a 5-minute interval, using a 9-period and 21-period moving-average crossover and Binance public klines. This is a test hypothesis, not a promise of an edge.

PROMPT 01

Connect MCP To Claude Before Build

Verify a broker or exchange MCP/API safely before any bot code exists.

PASTE INTO CLAUDE

```
Help me connect my broker or exchange MCP/API to Claude before we build the trading bot.

Do not write bot code yet. First guide me through the connection safely.

Start by asking me these questions if the answers are not obvious:
- Which venue: Alpaca, Pionex, Binance, Bybit, another crypto exchange, or another
brokerage?
- Trading stocks, crypto, or both?
- Which Claude environment: Claude Desktop, Claude Code, Claude in the browser, Cursor with
Claude, or another client?
- Am I using paper/test mode? If not, stop and tell me to create a paper/test setup first.
- Market-data-only permissions first, or paper-trading permissions?

Connection rules: paper/test mode only, no live trading, no order placed, no API keys pasted
into chat or source code, no secrets logged, no credentials exposed to frontend code. If an
exchange supports sub-accounts or restricted API keys, recommend that restricted paper/test
setup first.

Step 1: Identify the correct setup path - official MCP server if available, safest API
fallback if not, stop and give setup checklist if MCP tools are not visible.

Step 2: Verify the connection - account status, balances, open positions, open orders,
market data. Do not submit, preview-submit, or cancel an order during this smoke test unless
I explicitly ask later.

Step 3: Give me a connection report - venue, client, MCP status, account mode, tools
verified, credentials kept out of codebase, next step before building.

If anything is live, unknown, or unsafe, stop and tell me exactly what to fix.
```

PROMPT 02

Backtest Strategy With TradingView

Validate the strategy before it becomes the bot source of truth.

PASTE INTO CLAUDE

```
Help me backtest a trading strategy in TradingView before we build the bot.

Start by asking me for the missing inputs: market/ticker, timeframe, test window, strategy
idea and rules, risk rules, fees/slippage, long-only or long/short.

Default strategy if I do not provide one: BTCUSDT, 5m, 9-period fast MA, 21-period slow MA,
buy on bullish crossover, sell/exit on bearish crossover, paper/backtest only.

Use TradingView if the TradingView MCP or browser workflow is available.
If a TradingView MCP is available: open or prepare the chart, add the strategy as a
backtestable strategy (not just an indicator), run it over the date range, and report
results clearly.
If a TradingView MCP is not available: do not pretend you ran it. Generate a Pine Script v5
strategy instead and give exact steps to paste it into Pine Editor, add it to the chart, and
read the Strategy Tester results.

Backtest quality rules: no future data, no repainting signals, include commission and
slippage assumptions, make entry/exit rules explicit, separate in-sample from out-of-sample
where possible, and do not claim profitability the backtest does not support.

Return: the strategy hypothesis, exact rules tested, Pine Script if needed, full backtest
metrics (net profit, win rate, max drawdown, profit factor, total trades, average trade,
best trade, worst trade), weaknesses or market conditions where it fails, whether the rule
set is good enough for the instructions file, and the final concise rule set for the bot
instructions file.
```

PROMPT 03

Create The Instructions File

Interview first, then write the one-page rules file.

PASTE INTO CLAUDE

```
I want to create the instructions file for a paper-trading bot, but before writing it,
interview me so the bot matches my goals and trading style.

Do not write code yet. Do not build the bot yet. Do not connect to a broker yet.

Your job is to ask me questions, go back and forth with me, and then create a one-page file
called trading_bot_instructions.md.

Interview rules: ask 1-3 questions at a time, follow up on vague answers, do not give
financial advice, translate my preferences into clear bot rules, suggest a safe
paper-trading default if I do not know, and keep everything paper/test mode first.

Ask about: asset class, venue, default symbols, strategy style, timeframes, backtest
result/source, position sizing, max position, stop loss/take profit, max drawdown, whether
memory should start as two local files, what the bot should never do, and desired terminal
outputs.

After the interview, create trading_bot_instructions.md with:
1. Project Goal
2. Safety Rules
3. Strategy Rules
4. Risk Rules
5. Broker/MCP Rules
6. Memory Rules
7. Definition Of Done

When done: show me the full file contents, tell me to add or drag
trading_bot_instructions.md into my Claude Project/project knowledge, or keep it at the root
of my local coding project, and wait for my confirmation before any build prompt.
```

PROMPT 04

First Build Without Memory

Build an honest baseline with real market data and no memory yet.

PASTE INTO CLAUDE

```
Build the first working version of the paper-trading bot described in
trading_bot_instructions.md.

Use TypeScript and Node.js. If the project is empty, create the project from scratch.

Important: this first version should not have memory yet. Do not create ledger.csv. Do not
create learnings.md. Do not add adaptive filtering yet. Do not intentionally make the bot
fail. Do not fake losing trades. Do not use generated candle data or fixture candles. Use
real market candles where possible and report the actual results.

Create: package.json, tsconfig.json, src/types.ts, src/market.ts, src/strategy.ts,
src/risk.ts, src/execution.ts, src/replay.ts, src/bot.ts, src/index.ts.

Default: BTCUSDT, 5m interval, Binance public klines endpoint, 9/21 MA crossover. BUY when
fast MA crosses above slow MA, SELL when it crosses below, HOLD when there is no fresh
crossover.

Safety: paper/local execution only, no real trades, no live broker order endpoints, no API
keys requested. Keep broker/MCP integration as a future adapter unless already verified in
paper/test mode. Strategy returns a signal; risk approves or rejects; execution only
simulates a paper order.

Risk: configurable quantity and max position. SKIP if quantity exceeds max position. Every
decision needs a plain-English reason.

CLI: npm run scan and npm run replay:raw. replay:raw must print a trade-by-trade table and
summary metrics, flag repeated weak setups only if they appear, and say clearly if there are
not enough setups or no repeated losses in the lookback window.

After implementing, show me: final file structure, exact commands to run, sample scan
output, sample replay:raw output with summary metrics, and a note confirming this version
has no memory system yet.
```

PROMPT 05

Add Memory System And Run Comparison

Add the two-file journal and compare raw versus memory side by side.

PASTE INTO CLAUDE

```
Now add the two-file memory system to the existing TypeScript paper-trading bot.

Important: keep the original raw strategy path working, add memory as a separate improved
path, run the improved memory path after implementation, and compare the raw strategy output
against the memory-enabled output. Do not fake losses, invent candles, or force the bot to
fail.

Keep replay:raw as an honest baseline that still ignores memory.
Add: npm run replay:memory and npm run memory:reset.
Create or update: data/ledger.csv, data/learnings.md, src/memory.ts, src/adaptiveFilter.ts,
src/replay.ts, src/bot.ts, src/index.ts, package.json.

ledger.csv header:
timestamp,symbol,action,price,quantity,reason,mode,outcome,pnl

Before any future BUY or SELL, read both files and ask: has this symbol lost on a similar
setup before, does learnings.md warn about this setup, and is this signal repeating a known
bad trade? If matched: SKIP, no paper order, append a SKIP row when the memory path runs,
and print the reason clearly.

Memory quality rules: no seeded fake losses, no invented candles, no forced failure, learn
only from real outcomes. If no prior loss exists, replay:memory should say to run replay:raw
first.

Replay learning behavior: replay:raw still calculates the raw baseline without memory. After
the memory system exists, replay:raw may append real replay outcomes to ledger.csv. If
replay:raw finds a real losing crossover setup, write a plain-English lesson to
learnings.md. If it finds none, do not seed one. replay:memory should only skip with a real
prior warning; otherwise HOLD or tell me to keep paper testing.

After implementing, run: npm run replay:raw and npm run replay:memory.
Then show me: files changed, exact commands run, sample replay:raw output, sample
replay:memory output, the latest ledger.csv row, the relevant learnings.md note, and a short
explanation of what changed between the raw bot and the memory-enabled bot.
```

PROMPT 06

Finalize Bot For Experimentation

Package the build with documentation, environment config, and guardrails.

PASTE INTO CLAUDE

```
Finalize this paper-trading bot so I can safely experiment with it after the build.

Use the existing project, trading_bot_instructions.md, first bot implementation, broker/MCP
guardrails, and two-file memory system.

Do not add live trading. Do not expose secrets to frontend code. Do not commit API keys. Do
not fake market data or fake performance.

Finalize the project so it has: clear setup instructions, safe paper/local defaults,
optional paper MCP/API mode only if the connection is already verified, working commands for
scan, raw replay, memory replay, and memory reset, a clear place to change
symbols/intervals/strategy settings/quantity/max position, clear terminal logs, and
guardrails that prevent accidental live orders.

Create or update: README.md, .env.example, .gitignore, package.json scripts, any config file
needed, and any small src cleanup needed to make commands reliable.

README.md should explain: what the bot does, how to install, how to run scan/raw
replay/memory replay/memory reset, how paper/local execution works, how to configure
optional paper MCP/API mode safely, where ledger.csv and learnings.md live, how to
experiment with a new strategy or symbol, and the safety rules and limitations.

Package scripts should include: scan, replay:raw, replay:memory, memory:reset, and
broker:check/broker:preview only if a broker/MCP adapter exists.

Run final verification: npm install if needed, npm run scan, npm run replay:raw, npm run
replay:memory, npm run memory:reset, then npm run replay:memory again to confirm the
empty-memory behavior is clear.

If any command cannot run due to missing credentials, missing MCP tools, or unavailable
public market data, do not fake the result. Print the blocker clearly and make the fallback
behavior safe.

At the end, show me: final file structure, commands that passed, commands that were blocked
and why, any environment variables required, a short safety checklist, and the next three
experiments I can try in paper mode.
```

The critique additions

The companion critique strengthens this pack in three ways. Keep these rules beside the prompts while you work.

1. Do not overclaim the result

A headline such as Rs 10 lakh per month is not evidence. Any performance claim needs capital, position sizing, win rate, drawdown, fees, slippage, test window, and out-of-sample results. If those are missing, label the project an unverified experiment.

2. Call memory what it is

The two-file memory layer is an auditable journal plus a conservative skip filter. It does not discover an edge or become intelligent by itself. It can over-block and miss future winners, so review it on a fresh date range.

3. Put risk in the test

Make quantity, maximum position, stop or invalidation rules, maximum daily loss, drawdown, commission, spread, slippage, latency, outages, and applicable tax treatment explicit before discussing live money.

The honest default

If the evidence is thin, the correct decision is HOLD. A system that can say there is not enough evidence to justify a new risk is safer than a system that invents confidence.

Final verification checklist

Run these checks after Prompt 06. Keep the terminal output with the project so another person can reproduce the experiment.

Commands

```
npm install
npm run scan
npm run replay:raw
npm run replay:memory
npm run memory:reset
npm run replay:memory
```

Confirm

- The first replay uses real public market data or fails clearly.
- The raw path still ignores memory.
- The memory path reads ledger.csv and learnings.md before any BUY or SELL.
- No fake losses, invented candles, live orders, or exposed secrets exist.
- The final README explains setup, commands, memory reset, configuration, limitations, and the next three paper experiments.

Brand note

This prompt pack is part of Ayush Rawat's Trading Bot Brotherhood Intelligence field notes. It is an educational software workflow, not a trading recommendation.