

Master prompt — “Build my FX quant system (beginner-friendly, config-driven, multi-machine)”

Use this prompt with an assistant, a contractor, or a code-generating agent to keep all work consistent. It bundles project context, goals, constraints, and the explicit step-by-step expectations I want followed. Copy/paste it every time you ask for a new task, code, or review so the responder has the same baseline.

Project summary (one-sentence)

I'm building a configurable FX trading algorithm that collects tick/candle data from multiple brokers, normalises and aggregates it into a shared cloud feature store, computes indicator features, backtests strategies using those features, evaluates/rejects hallucinated model outputs with confidence scoring, and (optionally, after safe testing) places live orders — everything driven by a plain-text configuration file and accessible from two developer computers (no reliance on local-only files).

Primary goals & constraints

1. **Config-driven:** every behaviour (timeframes, TA settings, economic event rules, broker list, model parameters, risk/size rules) must be in a human-readable config file (YAML/JSON). Minimal code edits to change strategy.
2. **Beginner-friendly:** instructions must assume no coding experience; show exact terminal commands, file locations, and copy/paste content.
3. **Multi-machine:** repository + cloud storage must let two computers work interchangeably. No sensitive data checked into Git.
4. **Secrets & safety:** API keys stored in `config/.env` (excluded by `.gitignore`), never committed.
5. **Robust data pipeline:** raw tick/candle import → cleaning → timezone-normalization → feature engineering (SMA/EMA/RSI/ATR/VWAP/returns/volatility) → persist to cloud DB (Supabase).

6. **Deterministic, auditable decisions:** trade decisions come from explicit rules + AI analysis with confidence scoring; any AI output that fails checks must be rejected, logged, and flagged for human review.
 7. **Safe execution:** begin with paper/trading-sim, then small live canary rollout. Include stops/position sizing engine and emergency kill switch.
 8. **No hallucination tolerance:** implement metrics to detect and reject hallucinated or low-confidence AI outputs. Use ensemble + retrieval + tests against historical outcomes.
 9. **Readable tracking:** Notion/Sheet mirrored view of tasks; every run reports which Notion row/task to update and to what status.
-

Repo layout (exact, copy into terminal)

Create this project structure inside your git repo `fx-quant`:

```
fx-quant/
├── .gitignore
├── README.md
├── requirements.txt
├── config/
│   └── .env                # secrets only, NOT in git
├── src/
│   ├── test_connection.py
│   ├── get_candles.py
│   ├── data_engine.py
│   ├── supabase_upload.py
│   ├── order_executor.py
│   ├── backtester.py
│   ├── config_loader.py
│   └── main.py
├── data/                  # local cache / parquet (optional)
├── models/                # trained model weights (optional)
├── logs/
└── docs/
```

Add `.gitignore` entries:

```
config/.env
*.pyc
__pycache__/.
.vscode/
*.sqlite
/data/*.parquet
```

Minimal config file (YAML) — `config/system.yaml`

Copy this exact template into `config/system.yaml`. You will edit values; code reads this file.

```
general:
  project_name: "fx-quant"
  timezone: "UTC"

brokers:
  - name: "oanda"
    type: "oanda_v20"
    enabled: true
    instruments:
      - "EUR_USD"
      - "USD_JPY"

data:
  candle_count: 200
  candle_granularities:
    - "M1"
    - "M5"
  tick_export: false    # true to capture raw ticks

features:
  sma_windows: [3, 20]
  ema_windows: [20]
  rsi_period: 14
  atr_period: 14
  vwap_window: 20
  volatility_window: 20

ai:
```

```
    model: "local-ensemble"      # example: 'openai', 'local-llm',
'ensemble'
    confidence_threshold: 0.85
    retriever_enabled: true
    retriever_source: "supabase"

strategy:
    rule: "sma_cross"      # human-readable name of the rule
    params:
        short: 3
        long: 20
    trade_size_pct_of_equity: 0.01
    max_drawdown_pct: 0.05

execution:
    paper_mode: true
    canary_size_pct: 0.01
    max_positions: 3

supabase:
    url: "https://<your>.supabase.co"
    key_env_name: "SUPABASE_KEY"      # key stored in .env
    table: "fx_candles"
```

.env example (DO NOT COMMIT) — config/.env

```
# OANDA
OANDA_API_KEY=your_oanda_practice_key_here
OANDA_ACCOUNT_ID=101-011-38544319-001
OANDA_ENV=practice

# Supabase
SUPABASE_URL=https://abcd1234.supabase.co
SUPABASE_KEY=your_service_role_key_here
```

Recommended Python libraries (add to requirements.txt)

```
pandas
numpy
requests
python-dotenv
oandapyV20
supabase
sqlalchemy
psycopg2-binary
pyyaml
scikit-learn
ta    # technical indicators helper (optional)
```

Install:

```
conda activate fxquant
pip install -r requirements.txt
```

Essential code & commands (beginner-friendly tasks)

1) Create conda env (if not done)

```
conda create -n fxquant python=3.11 -y
conda activate fxquant
pip install -r requirements.txt
```

2) Fill **.env** and ensure **config/.env** exists

```
mkdir -p config
notepad config\.env    # paste keys, save
```

3) Test broker connection (run on each computer)

Create `src/test_connection.py` (example earlier). Then:

```
conda activate fxquant
python src/test_connection.py
# Expect Status: 200 and account summary
```

If works → update Notion: **Connect OANDA API = Done** (or equivalent row).

4) Pull candles & convert to DataFrame

`src/get_candles.py` uses OANDA to pull candles, then `data_engine.candles_to_df` and `build_all_features`. Run:

```
python src/get_candles.py
# Expect printed DataFrame tail with columns: open high low close
volume sma_20 ema_20 rsi_14 ...
```

Update tracker: **Generate indicators = In Progress** (move to Done when multiple indicators are validated).

5) Persist features to cloud (Supabase)

Use `src/supabase_upload.py` (full robust version). Run:

```
python src/supabase_upload.py
# After success check Supabase Table Editor for rows in fx_candles
```

Update tracker: **Cloud Storage Setup = Done** when rows appear.

6) Backtest engine

`src/backtester.py` should:

- read features from Supabase (or Parquet),
- simulate simple rule (e.g., SMA cross),
- produce P&L, drawdown, trade log CSV.
Run and inspect outputs.

7) Risk & execution

`src/order_executor.py`:

- paper mode always first,
- limit per-instrument size,
- implement emergency `STOP_ALL_TRADES` file-based kill switch or REST endpoint,
- logs every order and response into `logs/`.

8) Model + AI decision wrapper

- Use the AI only to *interpret* enriched context and add suggestions; core trade decision must pass deterministic rule checks.
 - For AI outputs, require:
 1. numeric confidence score (0–1),
 2. rationale text,
 3. a numeric check: predicted return/edge must be > threshold and backtest-simulated on recent history, otherwise reject.
 - If AI confidence < `confidence_threshold`, **reject** and log for human review.
-

How to avoid AI hallucination (practical measures)

1. **Retriever + citation**: use the latest features and historical trades as the retrieval corpus; attach evidence items to AI answers.
 2. **Ensemble + calibration**: combine multiple models or run multiple prompts; accept only when outputs agree and high confidence.
 3. **Sanity checks**: numeric ranges, deterministic thresholds, replay on historical windows.
 4. **Reject & log**: whenever model output fails checks, do not trade; alert user.
 5. **Unit tests & backtest holdouts**: always validate model recommendations on unseen historical holdout windows and store the evaluation metrics.
-

Supabase quick checklist

1. Create project at supabase.com.
2. Run SQL (copy earlier) in SQL Editor to create `fx_candles` table.
3. Put `SUPABASE_URL` and `SUPABASE_KEY` in `config/.env`.

4. Run `python src/supabase_upload.py` (it will push rows).
 5. In Table Editor confirm rows. Mark task Done.
-

Notion tracking: exactly what to update (your screenshot mapping)

- `Connect GitHub remote` → Done (if repo on GitHub & push/pull OK)
- `Install base libraries, Create virtual environment, Install Python & Conda` → Done (after conda + pip)
- `Stream live tick data` → Done only when live streaming tick capture is running continuously (we did candles; leave until live tick is streaming)
- `Connect OANDA API` → Done (once `test_connection.py` returns 200)
- `Generate indicators (VWAP etc)` → In Progress now; set to Done when you have multi-window SMA/EMA/RSI/VWAP + tests
- `Cloud Storage Setup` → In Progress; set to Done after Supabase table populated
- `Build backtesting engine` → Planned/In Progress; update when `backtester.py` produces reliable outputs
- `Paper trading execution` → Planned; set to Done after `order_executor` runs in paper mode and logs trades
- `Live trade execution` → Planned; only mark Done after canary rollout + monitoring in place

When you ask me to move to the next task, explicitly tell me which Notion row to change and the new status — I will say the exact row name, column, and status value to set.

Example prompts you can paste to continue (pick one)

A. “Generate code for X”

I want a beginner-friendly, copy-paste-ready Python file to do [task]. Use conda env name fxquant, read secrets from config/.env, and ensure I can run it with `python src/<file>.py`. Explain what it does in one paragraph and give exact commands to run it. Output only code and the run commands.

B. “Help me debug”

I ran `python src/<file>.py` and got this terminal output:

<paste terminal output>

Please explain the error in simple terms, show the one-line or small snippet fix, and give the exact commands to re-run. Then say what I should update in Notion (row and status).

C. “Make a config change”

Update `config/system.yaml` to add the timeframe H1 for EUR_USD, add RSI period 7, and set canary_size_pct to 0.5%. Provide the exact YAML snippet to paste and the single command to restart the uploader/main script.

Final notes / safety checklist before live trading

- Always run **paper mode** for weeks and run the backtester benchmark on the last N years.
- Keep a kill-switch file `STOP_ALL_TRADING` read by the executor; if present, executor must cancel open orders and go idle.
- Rotate API keys regularly and never store them in Git.
- Monitor latency & failures and set alerting (email/slack) for exceptions and low-confidence AI outputs.