Mini HFT Stack Implementation Plan

This plan outlines building a private mini-HFT system end-to-end, covering high-throughput market data ingestion, an in-memory limit order book, strategy execution, backtesting, live mode deployment, monitoring, profiling, and an advanced ML-alpha overlay.

---

1. Project Overview & Goals

Objective:

1. Ingest market data (FIX or WebSocket) at ≥100K messages/sec
2. Maintain an in-memory limit order book (LOB) with price-time matching
3. Execute algorithmic strategies (e.g., TWAP, VWAP)
4. Backtest on historical data and measure P&L, slippage, fill rate, latency
5. Switch seamlessly to live mode for streaming data
6. Expose real-time metrics via Prometheus/Grafana
7. Incorporate a streaming ML-driven alpha model for predictive order placement

Key Performance Indicators:

- Throughput: ≥100K msg/sec ingestion & parsing
- Latency: ≤1 ms book update & match
- Strategy performance: P&L improvement vs. naive baseline
- ML alpha lift: e.g., 5–10% reduction in slippage

Tech Stack:

- **Core Engine:** C++ (for LOB, matching, ingestion) or Rust
- **Orchestration & Scripting:** Python 3.8+ (asyncio, testing, ML)
- **Messaging:** Plain TCP sockets or Kafka (confluent-kafka Python client)
- **Serialization:** FlatBuffers or custom binary structs
- **Metrics & Monitoring:** Prometheus C++ client + python-prometheus-client, Grafana
- **Profiling:** Linux `perf`, `gprof`, flame graphs
- **ML Libraries:** scikit-learn (partial_fit), river (streaming models), PyTorch for prototype neural net

---

2. Repository Structure

```
mini-hft/
├── ingestion/            # Feed handlers, parsers, replay scripts
├── lob/                  # Limit order book & matching engine
├── strategy/             # Strategy interface & built-in algos (TWAP, VWAP)
├── backtester/           # Historical data runner, performance metrics
```

```
├── live/                # Live feed connector & adapter
├── monitoring/          # Prometheus exporters, Grafana dashboards
├── profiling/           # Perf and gprof scripts, analysis reports
├── ml_alpha/            # Feature extraction, streaming model, evaluation
├── tests/               # Unit, integration, and regression tests
├── docker/              # Dockerfiles & docker-compose setups
├── README.md            # High-level overview & quickstart
└── requirements.txt     # Python dependencies
```

---

3. Module Breakdown

3.1 Ingestion

- **FIX/WebSocket Parser (C++/Python):**
- Parse raw TCP socket frames or WebSocket JSON into structured messages: `{ timestamp, symbol, side, price, size }`
- Support both live feed (TCP/Kafka) and historical replay (file-based)
- **Lock-Free Queue (C++):**
- Single-producer, single-consumer ring buffer for raw message handoff
- API methods: `push(Message)`, `pop(Message&)`
- **Replay Script (Python):**
- Reads historical FIX file, pushes messages into C++ ingester at configurable speed

3.2 Limit Order Book (LOB)

- **Data Structures:**
- Bids and asks as `std::map<double, deque<Order>>` keyed by price (descending for bids, ascending for asks)
- **API:**

```cpp
struct Order { string id; enum Side { BUY, SELL }; double price; int size; };
struct Trade { string buy_id, sell_id; double price; int size; };

class OrderBook {
  void addOrder(const Order& o);       // Matches and/or queues
  void cancelOrder(const string& id);
  vector<Trade> getRecentTrades();      // For metrics
};
```

- **Matching Logic:**
- On `addOrder`, traverse the opposite side map until the incoming order is filled or no matching price levels remain
- Generate `Trade` events for partial and full fills

3.3 Strategy Module

- **Interface (Python):**

```
class Strategy:
    def __init__(self, parameters: dict): ...
    def on_market_snapshot(self, book_snapshot) -> List[Order]: ...
```

- **Built-In Algos:**
- **TWAP:** Split a parent order into N equal slices over a time window [T_start, T_end]
- **VWAP:** Weight child order sizes by traded volume in historical intervals
- **Execution:**
- Strategy fetches periodic snapshots of the LOB via an IPC mechanism (e.g., gRPC or shared memory)
- Emits child orders to the LOB via its API

3.4 Backtester

- **Pipeline Orchestration (Python):**
- Ingestion (historical replay) → LOB → strategy → trade logger
- **Metrics Calculation:**
- **P&L:** Sum of (executed price – reference price) × size
- **Slippage:** Difference between child order price and mid-price at submission
- **Fill Rate:** Executed size / total target size
- **Latency:** Time from order generation to trade event
- **Reports:**
- Export JSON/CSV with time series of metrics
- Generate basic plots via Matplotlib for quick analysis

3.5 Live Mode

- **Mode Switch:** Single CLI flag (`--mode backtest|live`) to select data source
- **Live Adapter:** Kafka or TCP socket consumer in Python to feed the same ingestion API
- **Resilience:** Auto-reconnect, exponential backoff, checkpointing of last sequence number

3.6 Monitoring & Profiling

- **Prometheus Metrics (C++ & Python):**
- Ingestion: `messages_received_total`, `ingestion_latency_seconds`
- LOB: `orders_added_total`, `trades_executed_total`, `book_depth_levels`
- Strategy: `orders_submitted_total`, `strategy_latency_seconds`
- **Grafana Dashboards:**
- Panel for throughput, queue backlog, P&L over time, slippage distribution
- **Profiling Setup:**
- Bash scripts that record `perf record -F 99 -g --` and generate flame graphs
- `gprof` instrumentation for C++ builds

---

4. ML-Alpha Overlay (In Depth)

Add a real-time machine learning layer that predicts short-term price movements to inform order placement.

4.1 Feature Engineering

- **Order Flow Imbalance (OFI):**\ $OFI_t = (\Delta BidSize_t) - (\Delta AskSize_t)$
- **Mid-Price:**\ $Mid_t = (BestBid_t + BestAsk_t)/2$
- **Spread:**\ $Spread_t = BestAsk_t - BestBid_t$
- **Volume-Weighted Trade Price:** aggregated over short sliding windows (e.g., 100 ms)
- **Derivative Features:** momentum (ΔMid), normalized OFI, rolling std-dev of mid-price

Implementation:

- In C++ LOB module, expose a snapshot struct containing these features every N ms
- Push snapshots into a Python async queue for ML processing

4.2 Streaming Model Training

- **Library Options:**
- **river** (formerly creme) for online learning
- **scikit-learn** `partial_fit` for incremental updates
- **PyTorch** with tiny MLP and manual weight updates for prototyping
- **Model Choice:**
- **Logistic Regression** to predict next-tick mid-price up/down
- **Online Random Forest** for better nonlinearity, if performance allows
- **Training Loop (Python):**

```python
from river import linear_model, metrics
model = linear_model.LogisticRegression()
metric = metrics.LogLoss()

async for snapshot in snapshot_queue:
    X, y = snapshot.features, snapshot.label  # label: sign(mid_future - mid_current)
    y_pred = model.predict_proba_one(X)
    metric.update(y, y_pred)
    model.learn_one(X, y)
```

- **Labeling:**
- Compute `mid_future` as mid-price Δ after a fixed horizon (e.g., 1 s)
- Maintain a small ring buffer of past snapshots to calculate labels

4.3 ML-Driven Strategy Integration

- **Hybrid TWAP:**

- For each child slice, consult model: if predicted up-tick, post aggressive buy at ask; else post passive buy at mid-price
- **Risk Controls:**
- Limit max daily loss, max order size based on model confidence
- **Latency Considerations:**
- Ensure model inference <1 ms (keep features & weights small)
- Profile Python async loop; consider Cython or ONNX runtime if too slow

4.4 Evaluation & Metrics

- **Alpha Quality:**
- **Hit Rate:** fraction of correct direction predictions
- **Log Loss:** model calibration
- **Strategy Uplift:**
- Compare slippage & P&L vs. baseline TWAP
- Use A/B backtests: 50% naive vs. 50% ML-informed slices
- **Visualization:**
- ROC curves, slippage histograms, cumulative P&L curves

4.5 Extension Ideas

- **Ensemble Methods:** add a streaming decision tree ensemble for non-linear patterns
- **Reinforcement Learning:** simple policy gradient to adjust child slice sizing dynamically
- **Anomaly Detection:** flag extreme order flow imbalances and throttle execution

---

5. Milestones & Timeline

Phase	Duration	Deliverable
1	1 wk	Ingestion + lock-free queue
2	2 wk	LOB + matching engine
3	1 wk	Strategy interface + TWAP/VWAP
4	1 wk	Backtester + core metrics
5	1 wk	Live mode connector
6	1 wk	Monitoring (Prometheus/Grafana) + profiling
7	2 wk	ML-alpha overlay (end-to-end streaming ML)

Total: \~9 weeks

Next Steps:

1. Scaffold repositories and CI pipeline
2. Implement Phase 1–3 core modules
3. Iteratively build and validate ML features
4. Measure and optimize performance at each step
5. Package up and prepare a demo script for recruiting interviews