Ensemble Trader – Skeleton Repository

> A ready-to-run scaffold for your 87-bot ensemble: data broker + feature store + LLM critic + aggregator + selector + risk + tests + migrations.

Repo Tree

```
ensemble-trader/
├─ pyproject.toml
├─ README.md
├─ requirements.txt
├─ .env.example
├─ config.yaml
├─ docker-compose.yml
├─ Dockerfile
├─ alembic.ini
├─ alembic/
│  ├─ env.py
│  └─ versions/
│     └─ 20251006_add_ops_fields.py
├─ scripts/
│  ├─ dev_up.sh
│  ├─ fmt.sh
│  └─ seed_demo_data.py
├─ src/
│  └─ ensemble_trader/
│     ├─ __init__.py
│     ├─ settings.py
│     ├─ cache.py
│     ├─ rate_limit.py
│     ├─ data_broker.py
│     ├─ feature_store.py
│     ├─ schemas.py
│     ├─ aggregator_dummy.py
│     ├─ selection.py
│     ├─ tool_handlers.py
│     ├─ regime/
│     │  ├─ __init__.py
│     │  └─ detector.py
│     ├─ risk/
│     │  ├─ __init__.py
│     │  └─ sizing.py
```

```
│     └─ app/
│        ├─ __init__.py
│        ├─ main.py
│        └─ routes.py
└─ tests/
   ├─ conftest.py
   ├─ test_schemas.py
   ├─ test_aggregator_dummy.py
   └─ test_selection_pipeline.py
```

---

pyproject.toml

```toml
[build-system]
requires = ["setuptools", "wheel"]
build-backend = "setuptools.build_meta"

[project]
name = "ensemble-trader"
version = "0.1.0"
description = "Ensemble futures/derivatives trading research scaffold"
authors = [{ name = "You" }]
requires-python = ">=3.10"
dependencies = [
  "pydantic>=2.7",
  "fastapi>=0.110",
  "uvicorn[standard]>=0.23",
  "sqlalchemy>=2.0",
  "alembic>=1.13",
  "psycopg2-binary>=2.9",
  "redis>=5.0",
  "numpy>=1.26",
  "pandas>=2.2",
  "httpx>=0.27",
  "python-dotenv>=1.0",
  "tenacity>=8.2",
  "pytest>=8.0",
]

[tool.pytest.ini_options]
pythonpath = ["src"]
addopts = "-q"
```

---

requirements.txt (alternative to pyproject)

```
pydantic>=2.7
fastapi>=0.110
uvicorn[standard]>=0.23
sqlalchemy>=2.0
alembic>=1.13
psycopg2-binary>=2.9
redis>=5.0
numpy>=1.26
pandas>=2.2
httpx>=0.27
python-dotenv>=1.0
tenacity>=8.2
pytest>=8.0
```

.env.example

```
# Database
DATABASE_URL=postgresql+psycopg2://postgres:postgres@db:5432/ensemble

# Redis (optional)
REDIS_URL=redis://redis:6379/0

# OpenAI
OPENAI_API_KEY=sk-...
OPENAI_MODEL=gpt-5-pro
```

---

config.yaml

```
runtime:
  cycle_seconds: 60
  timeframe: 1h
  require_min_bots: 5

selection:
  top_k_longs: 10
  top_k_shorts: 10
  min_confidence: 7.0
  min_volume_24h: 100000000
```

```
  max_abs_funding: 0.1
  min_open_interest: 5000000

risk:
  risk_pct: 0.005
  daily_loss_limit_pct: 0.03
  cluster_max_risk_pct: 0.02

regime:
  lookback_days: 120
  hmm_states: 3
```

---

docker-compose.yml

```
version: "3.9"
services:
  api:
    build: .
    env_file: .env
    command: uvicorn ensemble_trader.app.main:app --host 0.0.0.0 --port 8000 --
reload
    volumes: ["./src:/app/src", "./config.yaml:/app/config.yaml"]
    ports: ["8000:8000"]
    depends_on: [db, redis]
  db:
    image: postgres:15
    environment:
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: ensemble
    ports: ["5432:5432"]
  redis:
    image: redis:7
    ports: ["6379:6379"]
```

Dockerfile

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt ./
RUN pip install -r requirements.txt
COPY src ./src
```

```
COPY alembic.ini ./
COPY alembic ./alembic
COPY config.yaml ./
ENV PYTHONPATH=/app/src
CMD ["bash", "-lc", "alembic upgrade head && uvicorn
ensemble_trader.app.main:app --host 0.0.0.0 --port 8000"]
```

---

alembic.ini (minimal)

```
[alembic]
script_location = alembic
sqlalchemy.url = %(DATABASE_URL)s
```

---

alembic/env.py

```
from __future__ import annotations
from logging.config import fileConfig
from sqlalchemy import engine_from_config, pool
from alembic import context
import os

config = context.config
fileConfig(config.config_file_name)

db_url = os.getenv("DATABASE_URL")
if db_url:
    config.set_main_option("sqlalchemy.url", db_url)

# No models reflection here for brevity

def run_migrations_offline():
    context.configure(url=config.get_main_option("sqlalchemy.url"),
literal_binds=True)
    with context.begin_transaction():
        context.run_migrations()

def run_migrations_online():
    connectable =
engine_from_config(config.get_section(config.config_ini_section),
prefix="sqlalchemy.", poolclass=pool.NullPool)
    with connectable.connect() as connection:
```

```
        context.configure(connection=connection)
        with context.begin_transaction():
            context.run_migrations()

if context.is_offline_mode():
    run_migrations_offline()
else:
    run_migrations_online()
```

---

alembic/versions/20251006_add_ops_fields.py

```
from alembic import op
import sqlalchemy as sa

revision = "20251006_add_ops_fields"
down_revision = None
branch_labels = None
depends_on = None

def upgrade():
    op.create_table(
        "bot_signal",
        sa.Column("id", sa.BigInteger(), primary_key=True),
        sa.Column("ts", sa.TIMESTAMP(timezone=True), nullable=False),
        sa.Column("bot_id", sa.String(), nullable=False),
        sa.Column("symbol", sa.String(), nullable=False),
        sa.Column("regime", sa.String(), nullable=False),
        sa.Column("signal", sa.String(), nullable=False),
        sa.Column("confidence", sa.Float(), nullable=False),
        sa.Column("tp", sa.Float()),
        sa.Column("sl", sa.Float()),
    )
    op.create_table(
        "bot_weight",
        sa.Column("bot_id", sa.String(), primary_key=True),
        sa.Column("regime", sa.String(), primary_key=True),
        sa.Column("w", sa.Float(), nullable=False, server_default="1.0"),
        sa.Column("status", sa.String(), nullable=False,
server_default="active"),
        sa.Column("probation_until", sa.TIMESTAMP(timezone=True)),
        sa.Column("updated_at", sa.TIMESTAMP(timezone=True)),
    )
    op.create_table(
        "decision",
```

```
        sa.Column("id", sa.BigInteger(), primary_key=True),
        sa.Column("ts", sa.TIMESTAMP(timezone=True)),
        sa.Column("symbol", sa.String()),
        sa.Column("decision", sa.String()),
        sa.Column("consensus_conf", sa.Float()),
        sa.Column("entry", sa.Float()),
        sa.Column("tp", sa.Float()),
        sa.Column("sl", sa.Float()),
        sa.Column("regime", sa.String()),
        sa.Column("overridden_by", sa.String()),
        sa.Column("override_note", sa.String()),
    )
    op.create_table(
        "trade",
        sa.Column("id", sa.BigInteger(), primary_key=True),
        sa.Column("open_ts", sa.TIMESTAMP(timezone=True)),
        sa.Column("close_ts", sa.TIMESTAMP(timezone=True)),
        sa.Column("symbol", sa.String()),
        sa.Column("side", sa.String()),
        sa.Column("entry", sa.Float()),
        sa.Column("exit", sa.Float()),
        sa.Column("pnl", sa.Float()),
        sa.Column("regime", sa.String()),
        sa.Column("hit_tp", sa.Boolean()),
        sa.Column("hit_sl", sa.Boolean()),
        sa.Column("mfe", sa.Float()),
        sa.Column("mae", sa.Float()),
    )

def downgrade():
    op.drop_table("trade")
    op.drop_table("decision")
    op.drop_table("bot_weight")
    op.drop_table("bot_signal")
```

---

src/ensemble_trader/init.py

```
__all__ = []
```

---

src/ensemble_trader/settings.py

```
from __future__ import annotations
import os
from dotenv import load_dotenv

load_dotenv()

DATABASE_URL = os.getenv("DATABASE_URL", "postgresql+psycopg2://
postgres:postgres@localhost:5432/ensemble")
REDIS_URL = os.getenv("REDIS_URL", "")
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY", "")
OPENAI_MODEL = os.getenv("OPENAI_MODEL", "gpt-5-pro")
```

---

src/ensemble_trader/cache.py

```
# from prior message (unchanged)
from __future__ import annotations
import json, time, threading, hashlib
from typing import Any, Callable, Optional
try:
    import redis
except Exception:
    redis = None

def _stable_key(prefix: str, *parts: Any) -> str:
    raw = prefix + "|" + "|".join(map(lambda x: json.dumps(x, sort_keys=True,
default=str), parts))
    return hashlib.sha256(raw.encode()).hexdigest()

class BaseCache:
    def get(self, key: str) -> Optional[Any]: ...
    def set(self, key: str, value: Any, ttl_sec: int) -> None: ...

class InMemoryTTLCache(BaseCache):
    def __init__(self):
        self._store: dict[str, tuple[float, Any]] = {}
        self._lock = threading.RLock()
    def get(self, key: str) -> Optional[Any]:
        with self._lock:
            item = self._store.get(key)
            if not item:
                return None
```

```
            expires_at, value = item
            if time.time() >= expires_at:
                self._store.pop(key, None)
                return None
            return value
    def set(self, key: str, value: Any, ttl_sec: int) -> None:
        with self._lock:
            self._store[key] = (time.time() + ttl_sec, value)

class RedisCache(BaseCache):
    def __init__(self, url: str = "redis://localhost:6379/0"):
        if redis is None:
            raise RuntimeError("redis package not installed")
        self._r = redis.Redis.from_url(url, decode_responses=True)
    def get(self, key: str) -> Optional[Any]:
        raw = self._r.get(key)
        return json.loads(raw) if raw else None
    def set(self, key: str, value: Any, ttl_sec: int) -> None:
        self._r.setex(key, ttl_sec, json.dumps(value, default=str))

def make_cache() -> BaseCache:
    import os
    url = os.getenv("REDIS_URL")
    if url and redis is not None:
        try:
            return RedisCache(url)
        except Exception:
            pass
    return InMemoryTTLCache()

def ttl_cached(cache: BaseCache, prefix: str, ttl_sec: int):
    def deco(fn):
        def wrapped(*args, **kwargs):
            key = _stable_key(prefix, args, kwargs)
            hit = cache.get(key)
            if hit is not None:
                return hit
            value = fn(*args, **kwargs)
            cache.set(key, value, ttl_sec)
            return value
        return wrapped
    return deco
```

---

src/ensemble_trader/rate_limit.py

```
# from prior message (unchanged)
from __future__ import annotations
import time, threading, functools, asyncio
from typing import Callable

class TokenBucket:
    def __init__(self, rate_per_sec: float, burst: int):
        self.rate = float(rate_per_sec)
        self.capacity = float(burst)
        self.tokens = float(burst)
        self.timestamp = time.monotonic()
        self.lock = threading.RLock()
    def allow(self, tokens: float = 1.0) -> bool:
        with self.lock:
            now = time.monotonic()
            delta = now - self.timestamp
            self.timestamp = now
            self.tokens = min(self.capacity, self.tokens + delta * self.rate)
            if self.tokens >= tokens:
                self.tokens -= tokens
                return True
            return False

def ratelimit(tb: TokenBucket, cost: float = 1.0):
    def deco(fn: Callable):
        @functools.wraps(fn)
        def wrapped(*args, **kwargs):
            while not tb.allow(cost):
                time.sleep(0.01)
            return fn(*args, **kwargs)
        return wrapped
    return deco

def aratelimit(tb: TokenBucket, cost: float = 1.0):
    def deco(fn: Callable):
        @functools.wraps(fn)
        async def wrapped(*args, **kwargs):
            while not tb.allow(cost):
                await asyncio.sleep(0.01)
            return await fn(*args, **kwargs)
        return wrapped
    return deco
```

src/ensemble_trader/data_broker.py

```
from __future__ import annotations
from typing import Literal

# TODO: replace these stubs with real provider calls

def get_ohlcv(symbol: str, timeframe: Literal["1m","5m","1h","4h","1d"], limit:
int):
    raise NotImplementedError

def get_perp_metrics(symbol: str) -> dict:
    return {"funding_rate": 0.0, "open_interest": 0.0}

def get_sentiment(symbol: str) -> dict:
    return {"social_z": 0.0, "news_score": 0.0}

def get_whale_flow(symbol: str) -> dict:
    return {"netflow_24h": 0.0}

def get_options_flow(symbol: str) -> dict:
    return {"put_call_ratio": 1.0, "iv30": 0.0, "iv_skew_25d": 0.0,
"gamma_exposure": 0.0}
```

---

src/ensemble_trader/feature_store.py

```
from __future__ import annotations
from datetime import datetime
from typing import Literal
import math
from .cache import make_cache, ttl_cached
from .data_broker import get_ohlcv, get_perp_metrics, get_sentiment,
get_whale_flow, get_options_flow

_cache = make_cache()

def _atr_from_bars(bars, period: int = 14) -> float:
    if not bars or len(bars) < period + 1:
        return float("nan")
    trs = []
    for i in range(1, len(bars)):
        h = float(bars[i]["high"]); l = float(bars[i]["low"])
        pc = float(bars[i-1]["close"])
```

```
        tr = max(h - l, abs(h - pc), abs(l - pc))
        trs.append(tr)
    alpha = 1.0 / period
    atr = trs[0]
    for x in trs[1:]:
        atr = (1 - alpha) * atr + alpha * x
    return atr

def _realized_vol(closes, window: int = 30) -> float:
    if len(closes) < window + 1:
        return float("nan")
    rets = [math.log(closes[i]/closes[i-1]) for i in range(1, len(closes))][-
window:]
    mean = sum(rets)/len(rets)
    var = sum((r-mean)**2 for r in rets)/max(1,len(rets)-1)
    stdev = var ** 0.5
    return stdev * (8760 ** 0.5)  # annualize for 1h

@ttl_cached(_cache, prefix="features", ttl_sec=60)
def build_features(symbol: str, timeframe: Literal["1m","5m","1h","4h","1d"] =
"1h") -> dict:
    bars = get_ohlcv(symbol, timeframe, limit=240)
    closes = [float(b["close"]) for b in bars]
    atr = _atr_from_bars(bars, period=14)
    rv = _realized_vol(closes, window=30)
    mom20 = closes[-1] / closes[-20] - 1 if len(closes) >= 21 else float("nan")
    perp = get_perp_metrics(symbol)
    opts = get_options_flow(symbol)
    sent = get_sentiment(symbol)
    whale = get_whale_flow(symbol)
    return {
        "atr": float(atr),
        "rv30": float(rv),
        "mom20": float(mom20),
        "funding": float(perp.get("funding_rate", 0.0)),
        "oi": float(perp.get("open_interest", 0.0)),
        "put_call": float(opts.get("put_call_ratio", 1.0)),
        "iv30": float(opts.get("iv30", 0.0)),
        "iv_skew_25d": float(opts.get("iv_skew_25d", 0.0)),
        "gamma_exp": float(opts.get("gamma_exposure", 0.0)),
        "social_z": float(sent.get("social_z", 0.0)),
        "whale_net_24h": float(whale.get("netflow_24h", 0.0)),
        "asof": datetime.utcnow().isoformat() + "Z",
        "symbol": symbol.upper(),
        "timeframe": timeframe,
    }
```

src/ensemble_trader/schemas.py

```
# Pydantic v2 models from earlier (unchanged) – paste your full version here
```

---

src/ensemble_trader/aggregator_dummy.py

```
# Dummy aggregator from earlier (unchanged) – paste your full version here
```

---

src/ensemble_trader/selection.py

```
from __future__ import annotations
from dataclasses import dataclass
from datetime import datetime
from typing import Iterable, Dict, List, Tuple, Optional
from pydantic import BaseModel, Field
from .schemas import (
    BotSignal, AggregatorConfig, AggregatedDecision, AggregationInputs,
MarketRegime,
)
from .aggregator_dummy import aggregate  # wire dummy; swap when real is ready

class SymbolMD(BaseModel):
    symbol: str
    price: float
    volume24h: float
    funding_rate: float
    open_interest: float
    atr: float

class MarketSnapshot(BaseModel):
    asof: datetime
    regime: MarketRegime
    by_symbol: Dict[str, SymbolMD]

def group_signals_by_symbol(signals: Iterable[BotSignal]) -> Dict[str,
List[BotSignal]]:
    g: Dict[str, List[BotSignal]] = {}
    for s in signals: g.setdefault(s.symbol, []).append(s)
    return g
```

```
@dataclass
class SelectionConfig:
    top_k_longs: int = 10
    top_k_shorts: int = 10
    min_consensus_conf: float = 7.0
    min_volume_24h: float = 100e6
    max_abs_funding: float = 0.10
    min_oi: float = 5e6
    w_confidence: float = 0.70
    w_liquidity: float = 0.20
    w_stability: float = 0.10

def build_decisions_from_signals(*, signals: Iterable[BotSignal], now: datetime,
regime: MarketRegime, agg_cfg: AggregatorConfig, require_min_bots: int = 5) ->
List[AggregatedDecision]:
    grouped = group_signals_by_symbol(signals)
    out: List[AggregatedDecision] = []
    for symbol, sigs in grouped.items():
        if len(sigs) < require_min_bots: continue
        bundle = AggregationInputs(symbol=symbol, now=now, regime=regime,
config=agg_cfg, signals=sigs)
        try:
            out.append(aggregate(bundle))
        except Exception:
            continue
    return out

def _passes_filters(d: AggregatedDecision, md: SymbolMD, cfg: SelectionConfig) -
> bool:
    if d.decision == "flat": return False
    if d.consensus_confidence_1_10 < cfg.min_consensus_conf: return False
    if md.volume24h < cfg.min_volume_24h: return False
    if abs(md.funding_rate) > cfg.max_abs_funding: return False
    if md.open_interest < cfg.min_oi: return False
    return True

def _rank_key(d: AggregatedDecision, md: SymbolMD, cfg: SelectionConfig) ->
float:
    import math
    conf01 = (d.consensus_confidence_1_10 - 1.0) / 9.0
    liq = math.log(max(md.volume24h, 1.0)) / 20.0
    stab = (1.0 / max(md.atr, 1e-9)) / 0.01
    return cfg.w_confidence*conf01 + cfg.w_liquidity*liq + cfg.w_stability*stab

def select_top_futures_candidates(*, decisions: Iterable[AggregatedDecision],
snapshot: MarketSnapshot, sel_cfg: SelectionConfig) ->
Tuple[List[AggregatedDecision], List[AggregatedDecision]]:
    longs: List[tuple[AggregatedDecision,float]] = []
```

```
    shorts: List[tuple[AggregatedDecision,float]] = []
    for d in decisions:
        md = snapshot.by_symbol.get(d.symbol)
        if md is None: continue
        if not _passes_filters(d, md, sel_cfg): continue
        score = _rank_key(d, md, sel_cfg)
        if d.decision == "long": longs.append((d, score))
        elif d.decision == "short": shorts.append((d, score))
    longs = [d for d,_ in sorted(longs, key=lambda x:x[1], reverse=True)
[:sel_cfg.top_k_longs]]
    shorts = [d for d,_ in sorted(shorts, key=lambda x:x[1], reverse=True)
[:sel_cfg.top_k_shorts]]
    return longs, shorts

def run_selection_cycle(*, signals: Iterable[BotSignal], market_snapshot:
MarketSnapshot, agg_cfg: AggregatorConfig, sel_cfg: Optional[SelectionConfig] =
None, require_min_bots: int = 5):
    sel_cfg = sel_cfg or SelectionConfig()
    decisions = build_decisions_from_signals(signals=signals,
now=market_snapshot.asof, regime=market_snapshot.regime, agg_cfg=agg_cfg,
require_min_bots=require_min_bots)
    top_longs, top_shorts = select_top_futures_candidates(decisions=decisions,
snapshot=market_snapshot, sel_cfg=sel_cfg)
    return {"longs": top_longs, "shorts": top_shorts}
```

---

src/ensemble_trader/tool_handlers.py

```
from __future__ import annotations
import json
from typing import Any, Dict
from .feature_store import build_features

def handle_tool_call(name: str, arguments: Dict[str, Any]) -> Dict[str, Any]:
    if name == "get_features":
        symbol = arguments["symbol"]
        timeframe = arguments.get("timeframe", "1h")
        feats = build_features(symbol, timeframe)
        return {"ok": True, "features": feats}
    if name == "get_market_snapshot":
        n = int(arguments.get("n", 50))
        # TODO: return real top-N by volume
        symbols = [
            {"symbol": "BTCUSDT"},
            {"symbol": "ETHUSDT"},
```

```
            {"symbol": "SOLUSDT"},
        ][:n]
        return {"ok": True, "symbols": symbols}
    return {"ok": False, "error": f"unknown tool {name}"}
```

---

src/ensemble_trader/regime/detector.py

```
from __future__ import annotations
from dataclasses import dataclass
from typing import Literal

@dataclass
class RegimeVector:
    p_bull: float
    p_bear: float
    p_sideways: float
    p_high_vol: float

# TODO: Implement HMM / MS-AR; this is a placeholder

def detect_regime(closes: list[float]) -> RegimeVector:
    # naive split by slope & volatility
    if len(closes) < 30:
        return RegimeVector(0.25,0.25,0.25,0.25)
    slope = (closes[-1] / closes[-20]) - 1
    vol = max(closes) - min(closes)
    p_bull = max(0.0, min(1.0, 0.5 + slope))
    p_bear = max(0.0, 1.0 - p_bull)
    p_high_vol = 0.2 if vol > 0.1*closes[-1] else 0.1
    p_sideways = 1.0 - min(1.0, p_bull + p_bear + p_high_vol)
    return RegimeVector(p_bull, p_bear, p_sideways, p_high_vol)
```

---

src/ensemble_trader/risk/sizing.py

```
from __future__ import annotations

def position_size(acct_equity: float, entry: float, sl: float, risk_pct: float =
0.005) -> float:
    risk_$ = acct_equity * risk_pct
    stop_dist = abs(entry - sl)
```

```
    if stop_dist <= 0: return 0.0
    return risk_$ / stop_dist
```

---

src/ensemble_trader/app/main.py

```
from __future__ import annotations
from fastapi import FastAPI
from .routes import router

app = FastAPI(title="Ensemble Trader API")
app.include_router(router)
```

---

src/ensemble_trader/app/routes.py

```
from __future__ import annotations
from fastapi import APIRouter
from pydantic import BaseModel
from datetime import datetime, timezone
from ..schemas import AggregatorConfig, MarketRegime
from ..selection import MarketSnapshot, SymbolMD, run_selection_cycle

router = APIRouter()

class Health(BaseModel):
    status: str

@router.get("/healthz", response_model=Health)
def healthz():
    return {"status": "ok"}

# Demo endpoint: feed mock snapshot + empty signals list
@router.get("/demo")
def demo():
    snap = MarketSnapshot(
        asof=datetime.now(timezone.utc),
        regime=MarketRegime.bull,
        by_symbol={
            "BTCUSDT": SymbolMD(symbol="BTCUSDT", price=65000, volume24h=12e9,
funding_rate=0.01, open_interest=8e8, atr=450)
        }
    )
```

```
    res = run_selection_cycle(signals=[], market_snapshot=snap,
agg_cfg=AggregatorConfig())
    return res
```

---

tests/conftest.py

```
import os
import sys
sys.path.append("src")
```

---

tests/test_schemas.py

```
# Paste the schema tests from earlier
```

---

tests/test_aggregator_dummy.py

```
# Paste the aggregator tests from earlier
```

---

tests/test_selection_pipeline.py

```
# Paste the selection tests from earlier
```

---

scripts/dev_up.sh

```
#!/usr/bin/env bash
set -euo pipefail
python -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
export DATABASE_URL=${DATABASE_URL:-postgresql+psycopg2://
postgres:postgres@localhost:5432/ensemble}
export REDIS_URL=${REDIS_URL:-}
```

```
pytest -q || true
uvicorn ensemble_trader.app.main:app --reload
```

---

scripts/fmt.sh

```
#!/usr/bin/env bash
ruff check --fix || true
ruff format
```

---

scripts/seed_demo_data.py

```
# optional: insert a few demo rows to DB for dashboards
```

README.md

```
# Ensemble Trader (Skeleton)

This is a scaffold for an ensemble crypto futures research stack. It includes:

- Data Broker + Feature Store (cached & rate-limited)
- Pydantic contracts, Dummy Aggregator, Selection module
- Regime & Risk stubs, FastAPI app, Alembic migrations
- Tests for schemas, aggregator, and selection

## Quickstart
1. `cp .env.example .env` and fill secrets
2. `docker-compose up -d db redis`
3. `pip install -r requirements.txt`
4. `alembic upgrade head`
5. `uvicorn ensemble_trader.app.main:app --reload`
6. Visit `http://localhost:8000/healthz`

## Next Steps
- Implement real `data_broker` calls to your APIs
- Paste full `schemas.py`, `aggregator_dummy.py`, and tests from the chat
- Wire your LLM Critic & tool handlers
- Replace dummy regime with HMM/MS-AR, upgrade aggregator to weighted ensemble
```

UI — Bot Performance Dashboard & Probation Badges

This section adds a lightweight dashboard (served by the existing FastAPI app) and a simple React front end. You can keep it server-rendered (Jinja) or run the React page as a static asset.

1) API endpoints (FastAPI)

Create `src/ensemble_trader/app/perf_endpoints.py`:

```
from __future__ import annotations
from fastapi import APIRouter
from pydantic import BaseModel
from datetime import datetime, timedelta, timezone

# In real code, replace with SQLAlchemy queries to your DB

router = APIRouter(prefix="/api")
UTC = timezone.utc

class BotPerfRow(BaseModel):
    bot_id: str
    regime: str
    trades: int
    expectancy: float
    sharpe: float
    hit_rate: float
    weight: float
    status: str
    probation_until: str | None = None

class TPSSLBucket(BaseModel):
    bucket: str
    tp_hits: int
    sl_hits: int

@router.get("/bots/leaderboard")
def bots_leaderboard() -> list[BotPerfRow]:
    # TODO: query rolling 14/30d stats per bot+regime
    # SELECT bot_id, regime, COUNT(*) trades, AVG(pnl/abs(entry-sl))
expectancy, ... FROM trade JOIN bot_weight ...
    demo = [
        BotPerfRow(bot_id="momentum_v3", regime="bull", trades=182,
expectancy=0.18, sharpe=1.2, hit_rate=0.57, weight=1.15, status="active"),
        BotPerfRow(bot_id="mr_revert_v2", regime="sideways", trades=141,
expectancy=-0.05, sharpe=-0.3, hit_rate=0.46, weight=0.42, status="probation",
```

```
probation_until=(datetime.now(UTC)+timedelta(days=10)).isoformat()),
    ]
    return demo

@router.get("/bots/summary")
def bots_summary():
    # Topline KPIs for the dashboard header
    return {
        "active_bots": 72,
        "on_probation": 9,
        "retired": 6,
        "avg_weight": 0.97,
        "avg_hit_rate": 0.53,
    }

@router.get("/trades/tp_sl_stats")
def tp_sl_stats() -> list[TPSSLBucket]:
    # Hourly TP/SL hit counts for the last 7 days (demo data)
    out: list[TPSSLBucket] = []
    now = datetime.now(UTC).replace(minute=0, second=0, microsecond=0)
    for i in range(24):
        out.append(TPSSLBucket(bucket=(now - timedelta(hours=i)).isoformat(),
tp_hits=30+i%7, sl_hits=25+(i+3)%9))
    return list(reversed(out))

@router.get("/ops/guardrails")
def guardrails():
    return {
        "feeds_fresh": True,
        "exchange_ok": True,
        "kill_switch": False,
        "rate_limit_rejects": 0,
        "queue_backlog": 2,
        "daily_loss_limit_hit": False,
    }
```

Register in `src/ensemble_trader/app/main.py`:

```
from .perf_endpoints import router as perf_router
app.include_router(perf_router)
```

2) React dashboard (single file)

Create `src/ensemble_trader/app/static/BotPerformanceDashboard.jsx` (served as a static asset or embedded via a small HTML shell). This uses Tailwind + plain fetch; swap to your preferred UI kit.

```jsx
// BotPerformanceDashboard.jsx
import React, { useEffect, useState } from "react";

function Badge({ status, until }) {
  const color = status === "probation" ? "bg-yellow-100 text-yellow-800 border-
yellow-300" : status === "retired" ? "bg-gray-100 text-gray-700 border-
gray-300" : "bg-emerald-100 text-emerald-800 border-emerald-300";
  return (
    <span className={`px-2 py-1 rounded-full border text-xs font-medium ${color}
`}>
      {status}
      {status === "probation" && until ? ` • until ${new
Date(until).toLocaleDateString()}` : null}
    </span>
  );
}

export default function BotPerformanceDashboard() {
  const [summary, setSummary] = useState(null);
  const [rows, setRows] = useState([]);
  const [tpSl, setTpSl] = useState([]);
  const [ops, setOps] = useState(null);

  useEffect(() => {
    fetch("/api/bots/summary").then(r=>r.json()).then(setSummary);
    fetch("/api/bots/leaderboard").then(r=>r.json()).then(setRows);
    fetch("/api/trades/tp_sl_stats").then(r=>r.json()).then(setTpSl);
    fetch("/api/ops/guardrails").then(r=>r.json()).then(setOps);
  }, []);

  return (
    <div className="p-6 space-y-6">
      <header className="grid grid-cols-2 md:grid-cols-5 gap-4">
        {summary && [
          <KPI key="active" label="Active bots" value={summary.active_bots} />,
          <KPI key="prob" label="On probation" value={summary.on_probation} />,
          <KPI key="retired" label="Retired" value={summary.retired} />,
          <KPI key="avgw" label="Avg weight"
value={summary.avg_weight.toFixed(2)} />,
          <KPI key="hitrate" label="Avg hit rate"
value={(summary.avg_hit_rate*100).toFixed(1)+"%"} />
        ]}
      </header>

      <section className="bg-white rounded-2xl shadow p-4">
        <div className="flex items-center justify-between mb-3">
          <h2 className="font-semibold text-lg">Bot Leaderboard</h2>
```

```
          <div className="text-sm text-gray-500">Rolling window (configurable)</
div>
        </div>
        <table className="w-full text-sm">
          <thead>
            <tr className="text-left text-gray-600">
              <th className="py-2">Bot</th>
              <th>Regime</th>
              <th className="text-right">Trades</th>
              <th className="text-right">Expectancy (R)</th>
              <th className="text-right">Sharpe</th>
              <th className="text-right">Hit %</th>
              <th className="text-right">Weight</th>
              <th>Status</th>
            </tr>
          </thead>
          <tbody>
            {rows.map((r, i) => (
              <tr key={i} className="border-t">
                <td className="py-2 font-medium">{r.bot_id}</td>
                <td className="uppercase text-gray-600">{r.regime}</td>
                <td className="text-right">{r.trades}</td>
                <td className="text-right">{r.expectancy.toFixed(2)}</td>
                <td className="text-right">{r.sharpe.toFixed(2)}</td>
                <td className="text-right">{(r.hit_rate*100).toFixed(1)}%</td>
                <td className="text-right">{r.weight.toFixed(2)}</td>
                <td><Badge status={r.status} until={r.probation_until} /></td>
              </tr>
            ))}
          </tbody>
        </table>
      </section>

      <section className="bg-white rounded-2xl shadow p-4">
        <h2 className="font-semibold text-lg mb-3">TP/SL Hits (last 24
buckets)</h2>
        <TPChart data={tpSl} />
      </section>

      <section className="bg-white rounded-2xl shadow p-4">
        <h2 className="font-semibold text-lg mb-3">Guardrails</h2>
        {ops && (
          <ul className="grid grid-cols-2 md:grid-cols-3 gap-3 text-sm">
            <li className={pill(ops.feeds_fresh)}>Feeds fresh</li>
            <li className={pill(ops.exchange_ok)}>Exchange OK</li>
            <li className={pill(!ops.kill_switch)}>Kill switch OFF</li>
            <li className={pill(ops.rate_limit_rejects===0)}>Rate limit OK</li>
            <li className={pill(ops.queue_backlog<5)}>Queue healthy</li>
```

```
            <li className={pill(!ops.daily_loss_limit_hit)}>Loss limit safe</li>
          </ul>
        )}
      </section>
    </div>
  );
}

function KPI({ label, value }) {
  return (
    <div className="bg-white rounded-2xl shadow p-4">
      <div className="text-xs uppercase tracking-wide text-gray-500">{label}</
div>
      <div className="text-2xl font-semibold">{value}</div>
    </div>
  );
}

function pill(ok) {
  return `px-3 py-1 rounded-full inline-block border text-xs $
{ok?"bg-emerald-50 text-emerald-700 border-emerald-200":"bg-rose-50 text-
rose-700 border-rose-200"}`;
}

function TPChart({ data }) {
  // super-lightweight SVG chart (no deps)
  if (!data || !data.length) return <div className="text-gray-400">No data</
div>;
  const max = Math.max(...data.map(d=>Math.max(d.tp_hits, d.sl_hits)));
  const W=800, H=180, pad=24; const dx=(W-2*pad)/data.length;
  const scaleY = (v)=> H - pad - (v/max)*(H-2*pad);
  const tpPath = data.map((d,i)=> `${i?"L":"M"}${pad+i*dx},${scaleY(d.tp_hits)}
`).join(" ");
  const slPath = data.map((d,i)=> `${i?"L":"M"}${pad+i*dx},${scaleY(d.sl_hits)}
`).join(" ");
  return (
    <svg viewBox={`0 0 ${W} ${H}`} className="w-full h-48">
      <path d={tpPath} fill="none" strokeWidth="2" />
      <path d={slPath} fill="none" strokeWidth="2" />
      <g className="text-[10px] fill-gray-500">
        {data.map((d,i)=> (
          <text key={i} x={pad+i*dx} y={H-6} transform={`rotate(45 ${pad+i*dx} $
{H-6})`}>{new Date(d.bucket).getHours()}</text>
        ))}
      </g>
    </svg>
  );
}
```

Add a tiny HTML shell to serve it at `/dashboard` — `src/ensemble_trader/app/static/index.html`:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <script src="https://unpkg.com/react@18/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></
script>
    <script src="https://cdn.tailwindcss.com"></script>
    <title>Ensemble Trader – Performance</title>
  </head>
  <body class="bg-slate-50">
    <div id="root"></div>
    <script type="module">
      import BotPerformanceDashboard from './BotPerformanceDashboard.jsx';
      const e = React.createElement;

ReactDOM.createRoot(document.getElementById('root')).render(e(BotPerformanceDashboard));
    </script>
  </body>
</html>
```

Serve the static files. Update `src/ensemble_trader/app/routes.py`:

```
from fastapi.staticfiles import StaticFiles
from pathlib import Path

static_dir = Path(__file__).parent / "static"
router.mount("/dashboard", StaticFiles(directory=static_dir, html=True),
name="dashboard")
```

Navigate to `http://localhost:8000/dashboard` to view.

3) Probation policy + write-paths

- Back-end toggle: set `bot_weight.status='probation'` and `probation_until=NOW()+interval '14 days'` when rules trigger (low weight, negative Sharpe, etc.).
- UI reads those fields and renders the yellow badge + Until date.
- Optional: add admin endpoints to set/clear probation manually.

```
# PATCH /api/bots/{bot_id}/probation { status: "probation" | "active" |
"retired", days: 14 }
```

Sharing this plan with Bolt (Vibe Coder)

Sharing a ChatGPT link usually doesn't give Bolt access to the content. Two solid ways that work:

Option A — Handoff file 1) Create `bolt_handoff.md` at the repo root with: - a one-paragraph project brief - explicit TODO list ("Implement data_broker adapters", "Swap dummy aggregator", "Write SQL queries for perf endpoints", etc.) - paths/files to edit 2) Paste that file's contents into Bolt as the starting instruction, or upload the repo and point Bolt to that file.

Option B — Repo-first 1) Push this scaffold to GitHub (private is fine). 2) In Bolt, connect the repo and issue a task: *"Open* `bolt_handoff.md` *and complete items 1–3. Run* `pytest` *then start the API. Fix failing tests."*

I can generate `bolt_handoff.md` for you now if you want me to spell out the tasks with checkboxes and exact file paths.