WSB Tracker - Technical Specification

Project Overview

Name: wsb-tracker
Version: 0.1.0
Language: Python 3.10+
License: MIT

Purpose

A CLI tool that monitors Reddit's r/wallstreetbets (and configurable other subreddits) for stock ticker discussions, extracts mentions, analyzes sentiment, detects unusual activity patterns, and surfaces "interesting" trading opportunities based on a composite scoring model.

Target Users

- Retail traders wanting to monitor WSB sentiment
- Quantitative researchers studying social sentiment
- Developers building trading signal pipelines

Core Requirements

Functional Requirements

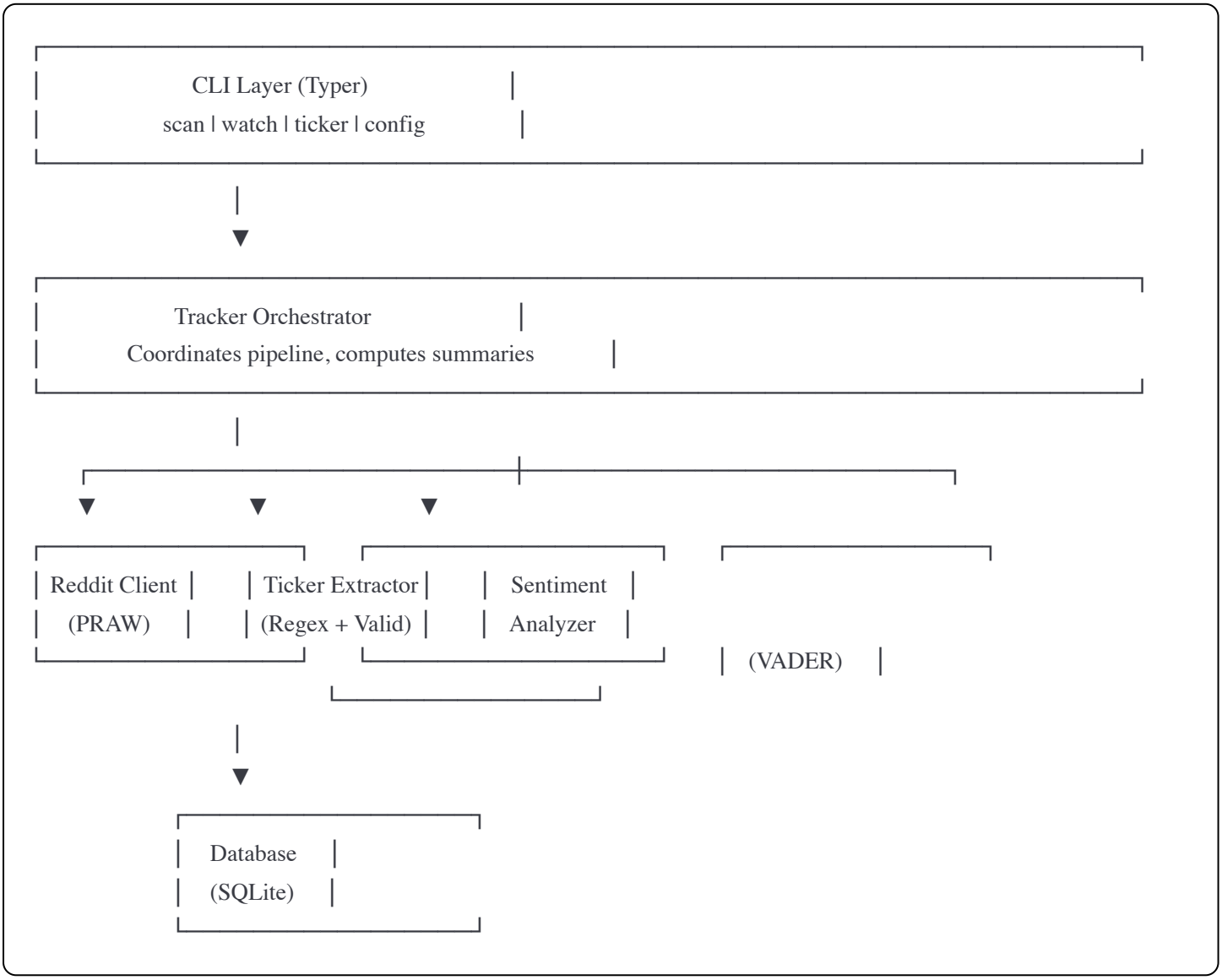
ID	Requirement	Priority
FR-1	Extract stock ticker symbols from Reddit posts and comments	Must
FR-2	Analyze sentiment of text surrounding ticker mentions	Must
FR-3	Track mention frequency over time to detect velocity changes	Must
FR-4	Compute composite "heat score" ranking interesting opportunities	Must
FR-5	Persist data to SQLite for historical trend analysis	Must
FR-6	Provide CLI interface with scan, watch, and ticker detail commands	Must
FR-7	Export results to JSON for pipeline integration	Should
FR-8	Support multiple subreddits	Should
FR-9	Optional Discord webhook notifications	Could
FR-10	Quiet mode for cron job integration	Should

Non-Functional Requirements

ID	Requirement
NFR-1	Run without Reddit API credentials (read-only mode)
NFR-2	Complete a scan of 100 posts in < 60 seconds
NFR-3	Database should handle 30+ days of data without performance issues
NFR-4	CLI output should be human-readable with colors and tables
NFR-5	Code should follow PEP 8 and include docstrings

Architecture

High-Level Design



Data Flow

1. **Fetch**: Reddit Client retrieves posts from configured subreddits (hot, new, rising, top)
2. **Extract**: Ticker Extractor identifies stock symbols using pattern matching + validation
3. **Analyze**: Sentiment Analyzer scores text around each mention using VADER + custom lexicon
4. **Store**: Mentions saved to SQLite with deduplication (ticker + post_id unique)
5. **Aggregate**: Tracker computes summaries with trend comparison to historical baseline
6. **Present**: CLI renders results as tables or exports to JSON

Data Models

Core Entities

Sentiment

python

```python
class Sentiment(BaseModel):
    compound: float       # -1.0 to 1.0, overall score
    positive: float       # 0.0 to 1.0
    negative: float       # 0.0 to 1.0
    neutral: float        # 0.0 to 1.0

    @computed_field
    def label(self) -> SentimentLabel:
        # Maps compound score to: very_bullish, bullish, neutral, bearish, very_bearish
        # Thresholds: >=0.5 very_bullish, >=0.15 bullish, <=-0.5 very_bearish, <=-0.15 bearish
```

RedditPost

python

```python
class RedditPost(BaseModel):
    id: str                 # Reddit post ID
    title: str
    selftext: str           # Post body
    author: str
    subreddit: str
    score: int              # Upvotes
    upvote_ratio: float
    num_comments: int
    created_utc: datetime
    flair: Optional[str]
    url: str
    permalink: str
    is_dd: bool             # Due Diligence flair detected
    awards_count: int

    @computed_field
    def engagement_ratio(self) -> float:
        # num_comments / score (higher = more discussion)

    @computed_field
    def full_text(self) -> str:
        # title + selftext combined
```

TickerMention

```python
class TickerMention(BaseModel):
    ticker: str             # 1-5 uppercase letters
    post_id: str
    post_title: str
    sentiment: Sentiment
    context: str            # ~50 chars around mention
    timestamp: datetime
    subreddit: str
    post_score: int
    post_flair: Optional[str]
```

TickerSummary

```python
```

```python
class TickerSummary(BaseModel):
    ticker: str
    mention_count: int
    unique_posts: int
    avg_sentiment: float
    sentiment_std: float      # Sentiment volatility
    bullish_ratio: float      # % of mentions with sentiment > 0.15
    total_score: int          # Sum of post scores
    dd_count: int             # Number of DD-flaired posts
    avg_engagement: float
    first_seen: datetime
    last_seen: datetime

    # Trend indicators (vs previous period)
    mention_change_pct: Optional[float]
    sentiment_change: Optional[float]

    @computed_field
    def heat_score(self) -> float:
        # Composite score for ranking (see Heat Score Algorithm below)
```

TrackerSnapshot

```python
class TrackerSnapshot(BaseModel):
    timestamp: datetime
    subreddits: list[str]
    posts_analyzed: int
    tickers_found: int
    summaries: list[TickerSummary]
    top_movers: list[str]     # Tickers with biggest changes
```

Heat Score Algorithm

The heat score ranks tickers by "interestingness" using multiple weighted factors:

```python
```

```python
def heat_score(self) -> float:
    # Factor 1: Mention velocity (capped at 5x weight)
    mention_factor = min(self.mention_count / 10, 5.0)

    # Factor 2: Sentiment strength (absolute value, regardless of direction)
    sentiment_factor = abs(self.avg_sentiment) * 2

    # Factor 3: DD presence (quality indicator, up to 1.5 bonus)
    dd_factor = min(self.dd_count, 3) * 0.5

    # Factor 4: Engagement level
    engagement_factor = min(self.avg_engagement, 1.0)

    # Factor 5: Trending bonus
    trend_bonus = 1.0 if (self.mention_change_pct and self.mention_change_pct > 50) else 0.0

    return mention_factor + sentiment_factor + dd_factor + engagement_factor + trend_bonus
```

Factor	Max Contribution	Rationale
Mentions	5.0	High volume = community attention
Sentiment	~2.0	Strong sentiment (either direction) = conviction
DD Posts	1.5	Quality analysis posts
Engagement	1.0	Discussion depth
Trend	1.0	Momentum indicator

Ticker Extraction

Strategy

1. **High confidence ($TICKER)**: Explicit cashtag format, confidence 0.95
2. **Contextual patterns**: "buying X", "calls on X", etc., confidence 0.8
3. **ALL CAPS words**: Validated against known tickers, confidence 0.6

Validation Rules

- Length: 1-5 characters
- Single-letter tickers require $ prefix (e.g., $F for Ford)
- Filter false positives: common words (I, A, AM, PM, DD, OP, etc.)

- Filter WSB slang: YOLO, FOMO, HODL, FD, etc.
- Filter time references: MON, TUE, JAN, FEB, etc.
- Filter financial acronyms: EPS, PE, ROI, ATH, etc.

Known Valid Tickers Override

Some single-letter or common tickers are valid with $ prefix:

- $A (Agilent), $F (Ford), $T (AT&T), $V (Visa), $X (US Steel)
- $DD (DuPont), $AI (C3.ai), $ON (ON Semiconductor)

Sentiment Analysis

Base: VADER

Use `vaderSentiment` library as foundation (rule-based, fast, handles social media text well).

Custom Lexicon Additions

Extend VADER with financial/WSB vocabulary:

```python
FINANCIAL_LEXICON = {
    # Bullish (positive scores)
    "bullish": 2.5, "moon": 2.0, "mooning": 2.5, "tendies": 2.0,
    "gains": 1.5, "squeeze": 1.5, "breakout": 1.5, "undervalued": 1.5,
    "diamond hands": 2.0, "🚀": 2.0, "💎": 1.5, "📈": 1.5,

    # Bearish (negative scores)
    "bearish": -2.5, "crash": -2.0, "dump": -2.0, "dumping": -2.5,
    "bag holder": -2.0, "paper hands": -2.0, "rug pull": -2.5,
    "overvalued": -1.5, "🐻": -1.5, "📉": -1.5,

    # Contextual
    "yolo": 0.5, "dd": 0.5, "priced in": -0.5,
}
```

Ticker-Specific Analysis

When analyzing sentiment for a specific ticker, extract only sentences containing that ticker to avoid noise from multi-ticker posts.

Database Schema

SQLite Tables

sql

```sql
-- Individual ticker mentions
CREATE TABLE mentions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    ticker TEXT NOT NULL,
    post_id TEXT NOT NULL,
    post_title TEXT,
    subreddit TEXT NOT NULL,
    sentiment_compound REAL,
    sentiment_positive REAL,
    sentiment_negative REAL,
    sentiment_neutral REAL,
    context TEXT,
    post_score INTEGER DEFAULT 0,
    post_flair TEXT,
    timestamp DATETIME NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(ticker, post_id)  -- Deduplication
);

-- Indexes
CREATE INDEX idx_mentions_ticker ON mentions(ticker);
CREATE INDEX idx_mentions_timestamp ON mentions(timestamp);
CREATE INDEX idx_mentions_ticker_timestamp ON mentions(ticker, timestamp);

-- Periodic snapshots for historical analysis
CREATE TABLE snapshots (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    timestamp DATETIME NOT NULL,
    subreddits TEXT NOT NULL,      -- JSON array
    posts_analyzed INTEGER,
    tickers_found INTEGER,
    summaries TEXT NOT NULL,       -- JSON array
    top_movers TEXT,               -- JSON array
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_snapshots_timestamp ON snapshots(timestamp);

-- Alert history (prevent duplicate notifications)
CREATE TABLE alerts (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    ticker TEXT NOT NULL,
    alert_type TEXT NOT NULL,
    message TEXT,
    triggered_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

```
CREATE INDEX idx_alerts_ticker_time ON alerts(ticker, triggered_at);
```

Configuration

Environment Variables

```bash
# Reddit API (optional - enables higher rate limits)
REDDIT_CLIENT_ID=
REDDIT_CLIENT_SECRET=
REDDIT_USER_AGENT=wsb-tracker/1.0

# Scraping
SUBREDDITS=wallstreetbets        # Comma-separated
POSTS_PER_FETCH=100              # Per subreddit
FETCH_COMMENTS=true
COMMENTS_PER_POST=20

# Database
DB_PATH=wsb_tracker.db

# Analysis
MIN_MENTIONS_TO_TRACK=2
LOOKBACK_HOURS=24                # For trend comparison

# Alerts
ALERT_MIN_MENTIONS=5
ALERT_SENTIMENT_CHANGE=0.3
ALERT_MENTION_SPIKE_PCT=100.0
ALERT_MIN_HEAT_SCORE=3.0

# Output
OUTPUT_DIR=output

# Notifications (optional)
DISCORD_WEBHOOK_URL=
```

Config Management

Use `pydantic-settings` for type-safe configuration with `.env` file support.

CLI Interface

Commands

`wsb-tracker scan`

Run a single analysis cycle.

```
Options:
  -s, --subreddits TEXT   Comma-separated subreddits [default: from config]
  -n, --top INTEGER       Number of top tickers to show [default: 15]
  -e, --export PATH       Export results to JSON file
  -q, --quiet             Minimal output for cron jobs
```

Output (normal mode):

- Scan statistics panel (posts analyzed, tickers found, new mentions)
- Table of top tickers by heat score with columns: Rank, Ticker, Mentions, Sentiment, Heat, DD, Trend, Sentiment Δ
- "Top Movers" callout for tickers exceeding alert thresholds

Output (quiet mode):

- Tab-separated: $TICKER, mentions, sentiment, heat_score

`wsb-tracker ticker <SYMBOL>`

Show detailed analysis for a specific ticker.

Output:

- Detailed panel with all metrics
- Trend data vs 24h baseline
- First/last seen timestamps

`wsb-tracker watch`

Continuous monitoring mode.

```
Options:
  -i, --interval INTEGER  Minutes between scans [default: 60]
  -s, --subreddits TEXT   Comma-separated subreddits
```

Behavior:

- Loop: scan → display compact results → sleep → repeat

- Ctrl+C to exit gracefully

wsb-tracker cleanup

Remove old data from database.

```
Options:
  -d, --days INTEGER      Days of data to keep [default: 30]
```

wsb-tracker config

Display current configuration.

Project Structure

```
wsb-tracker/
├── wsb_tracker/
│   ├── __init__.py          # Package exports, version
│   ├── models.py            # Pydantic data models
│   ├── config.py            # Settings with pydantic-settings
│   ├── ticker_extractor.py  # Ticker extraction and validation
│   ├── sentiment.py         # VADER + custom lexicon
│   ├── database.py          # SQLite operations
│   ├── reddit_client.py     # PRAW wrapper
│   ├── tracker.py           # Core orchestration
│   └── cli.py               # Typer CLI with Rich output
├── tests/
│   ├── __init__.py
│   ├── test_ticker_extractor.py
│   ├── test_sentiment.py
│   ├── test_database.py
│   └── test_tracker.py
├── main.py                  # Entry point
├── pyproject.toml           # Modern packaging
├── requirements.txt         # Dependencies
├── .env.example             # Template config
├── .gitignore
├── LICENSE
└── README.md
```

Dependencies

Runtime

```
praw>=7.0.0              # Reddit API
pydantic>=2.0.0          # Data validation
pydantic-settings>=2.0.0 # Config management
typer>=0.9.0             # CLI framework
rich>=13.0.0             # Terminal formatting
vaderSentiment>=3.3.2    # Sentiment analysis
python-dotenv>=1.0.0     # .env file loading
schedule>=1.2.0          # Optional: daemon scheduling
```

Development

```
pytest>=7.0.0
pytest-cov>=4.0.0
black>=23.0.0
ruff>=0.1.0
mypy>=1.0.0
```

Testing Requirements

Unit Tests

Module	Test Coverage
ticker_extractor	$TICKER extraction, false positive filtering, context extraction
sentiment	Compound score calculation, custom lexicon, ticker-specific analysis
database	CRUD operations, deduplication, aggregation queries, cleanup
models	Computed fields (heat_score, engagement_ratio, sentiment label)

Integration Tests

- Full scan pipeline with mocked Reddit responses
- Database persistence across multiple scans
- Trend calculation with historical data

Test Data

Create fixtures with sample Reddit posts containing:

- Various ticker formats ($GME, GME, buying GME)
- Multiple tickers per post

- DD flaired posts
- Different sentiment patterns

Error Handling

Scenario	Behavior
No Reddit credentials	Continue in read-only mode with warning
Reddit API rate limit	Retry with exponential backoff (max 3 attempts)
Invalid subreddit	Log error, skip, continue with other subreddits
Database locked	Retry with short delay
Malformed post data	Skip post, log warning

Future Extensions

These are out of scope for v0.1.0 but the architecture should accommodate:

1. **Discord/Telegram notifications** - Webhook integration for alerts
2. **Web dashboard** - FastAPI backend serving React frontend
3. **Options flow tracking** - Track calls vs puts sentiment
4. **Cross-platform data** - Twitter/X, StockTwits integration
5. **ML sentiment model** - Fine-tuned transformer replacing VADER
6. **Backtesting** - Correlate historical mentions with price movements

---

Implementation Notes for Claude Code

Priority Order

1. Start with `models.py` - define all data structures first
2. Implement `config.py` - settings management
3. Build `ticker_extractor.py` with comprehensive test coverage
4. Build `sentiment.py` with custom lexicon
5. Implement `database.py` with all SQL operations

6. Create `reddit_client.py` with PRAW wrapper
7. Build `tracker.py` orchestrating the pipeline
8. Finally, implement `cli.py` with Rich formatting

Key Design Principles

- **Separation of concerns**: Each module has single responsibility
- **Dependency injection**: Tracker accepts client/database instances for testing
- **Fail gracefully**: Never crash on bad data, log and continue
- **Type safety**: Use Pydantic models throughout, enable mypy strict mode
- **Idempotency**: Re-running scans should deduplicate via UNIQUE constraint

Code Style

- Follow PEP 8
- Use type hints everywhere
- Docstrings for all public functions (Google style)
- Comments explaining non-obvious logic
- No magic numbers - use named constants

Acceptance Criteria

The project is complete when:

- [ ] `wsb-tracker scan` successfully fetches posts and displays results
- [ ] `wsb-tracker scan -q` outputs parseable tab-separated data
- [ ] `wsb-tracker scan -e output.json` creates valid JSON export
- [ ] `wsb-tracker ticker GME` shows detailed analysis
- [ ] `wsb-tracker watch` runs continuously until Ctrl+C
- [ ] `wsb-tracker config` displays all settings
- [ ] Database persists between runs and tracks trends
- [ ] Works without Reddit API credentials (read-only mode)
- [ ] All tests pass with >80% coverage
- [ ] README documents installation and usage