InvestiGator - AI Investment Research Assistant

Vijaykumar Singh

Version 2.1, 2025-01-31

Table of Contents

Your AI-powered investment research companion that runs entirely on your MacBook

InvestiGator is a sophisticated AI investment research system that combines SEC filing analysis with technical analysis to generate professional investment recommendations. Built for privacy-conscious investors who want institutional-grade analysis without cloud dependencies.

No API Key Required! InvestiGator uses free SEC EDGAR APIs and Yahoo Finance for all financial data.

Chapter 1. □ Table of Contents

Chapter 2. Introduction

InvestiGator is a comprehensive investment research platform that runs entirely on your local machine. It combines:

- **SEC EDGAR Integration**: Direct access to company filings using free APIs
- **Technical Analysis**: Real-time market data analysis with advanced indicators
- **AI-Powered Insights**: Three specialized LLMs for different analysis aspects
- **Privacy-First Design**: All processing happens locally on your Mac
- **Professional Reports**: Institutional-quality PDF reports with actionable insights

Chapter 3. Key Features

3.1. Comprehensive Analysis

- **SEC Filing Analysis**: AI-powered fundamental analysis of 10-K/10-Q filings using pattern-based architecture
- **XBRL Data Processing**: Automated extraction of financial metrics via SEC Frame API
- **Technical Analysis**: 30+ indicators including moving averages, RSI, MACD, Bollinger Bands, Fibonacci levels
- **Investment Synthesis**: Weighted recommendations (60% fundamental, 40% technical) with 0-10 scoring
- **Professional Reports**: PDF reports with charts, executive summaries, and actionable insights

3.2. Privacy-First Design

- **100% Local Processing**: All AI models run on your MacBook using Ollama
- **No Cloud Dependencies**: Your investment data never leaves your device
- **No API Keys Required**: Uses free SEC EDGAR APIs and Yahoo Finance
- **Secure Storage**: PostgreSQL with connection pooling and encrypted credentials
- **Private Communications**: Optional secure email delivery via SMTP/TLS

3.3. Advanced AI Integration

- **Three-Stage LLM Pipeline**: SEC fundamental → Technical analysis → Investment synthesis
- **Configurable Models**: Support for Llama3.1, Mixtral, Phi4, Qwen2.5, and custom models
- **Single-Threaded Processing**: Prevents GPU/RAM exhaustion with sequential execution
- **Complete LLM Observability**: All prompts and responses cached with processing metrics
- **Large Context Support**: Up to 32K context window for comprehensive analysis
- **Structured JSON Outputs**: Consistent, parseable AI responses

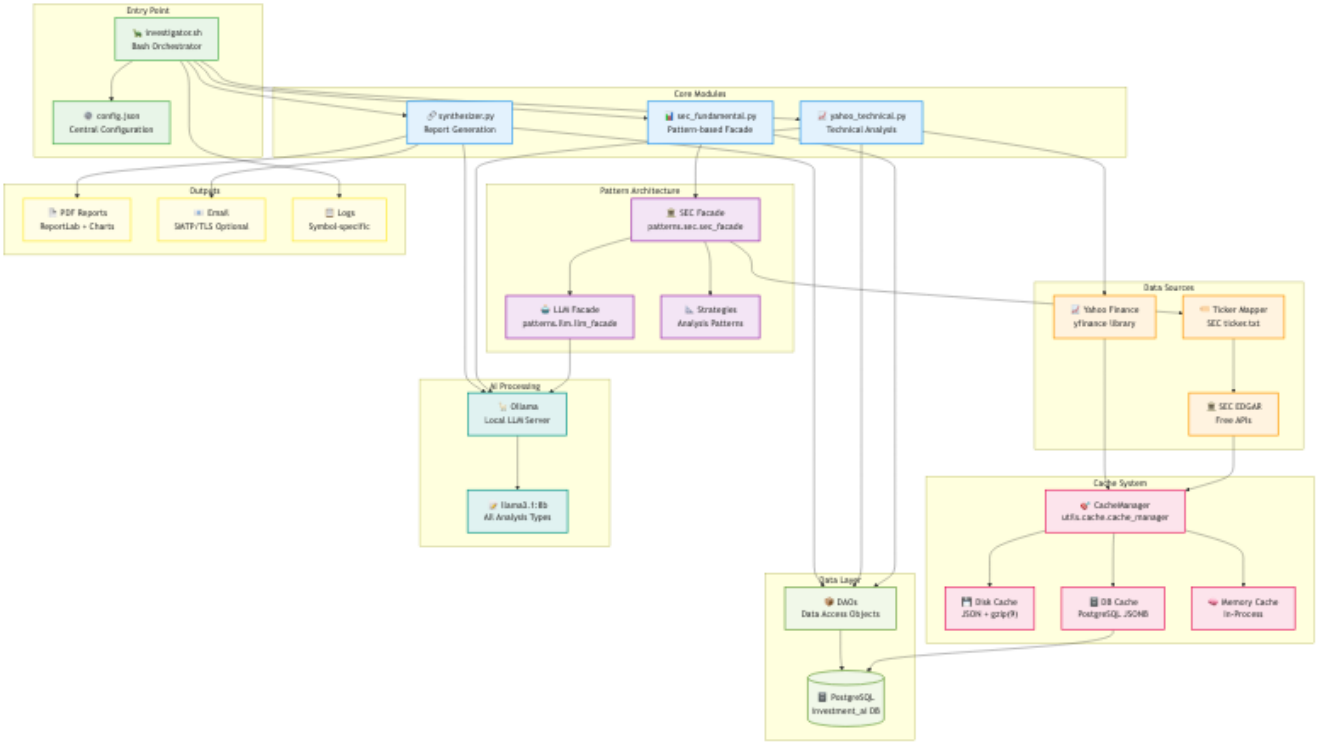
3.4. Intelligent Cache Management

- **Multi-Level Caching**: Disk and database storage with configurable backends
- **Uniform Compression**: gzip (level 9) for disk, JSONB compression for PostgreSQL
- **Flexible Configuration**: Enable/disable specific storage types and cache categories
- **Cache Cleanup & Inspection**: Comprehensive utilities for cache management
- **Force Refresh Capability**: Global or symbol-specific cache invalidation
- **Performance Optimization**: Intelligent cache prioritization and hit-rate tracking

3.5. Performance & Automation

- **Multi-Backend Caching**: Memory (priority 30) → Disk (priority 10) → Database (priority 5)
- **Smart Cache Management**: Automatic promotion of frequently accessed data to faster storage
- **Efficient Data Storage**: 70-80% compression ratios with gzip level 9
- **Weekly Reports**: Automated portfolio analysis with batch processing
- **Configurable Processing**: Single-threaded LLM execution to prevent resource exhaustion
- **Cache Performance**: 10-50ms disk access, 50-200ms database access
- **Extensible Architecture**: Pattern-based design with facades, strategies, and observers

Chapter 4. □□ System Architecture



4.1. Component Descriptions

Component	Description
investigator.sh	Bash orchestrator that coordinates all analysis components, handles CLI arguments, manages system checks
sec_fundamental.py	Entry point for SEC analysis, uses pattern-based architecture via FundamentalAnalysisFacadeV2
yahoo_technical.py	Fetches market data from Yahoo Finance, calculates 30+ technical indicators, generates AI insights
synthesizer.py	Combines analyses with weighted scoring (60/40 split), generates investment recommendations and PDF reports
Pattern Architecture	Implements facades, strategies, and observer patterns for extensibility and separation of concerns
Cache Manager	Multi-backend cache system with priority-based retrieval: Memory(30) → Disk(10) → Database(5)
Ollama Integration	Local LLM server running llama3.1:8b-instruct-q8_0 for all analysis types
PostgreSQL	Stores analysis results, LLM responses, and cached data with JSONB compression

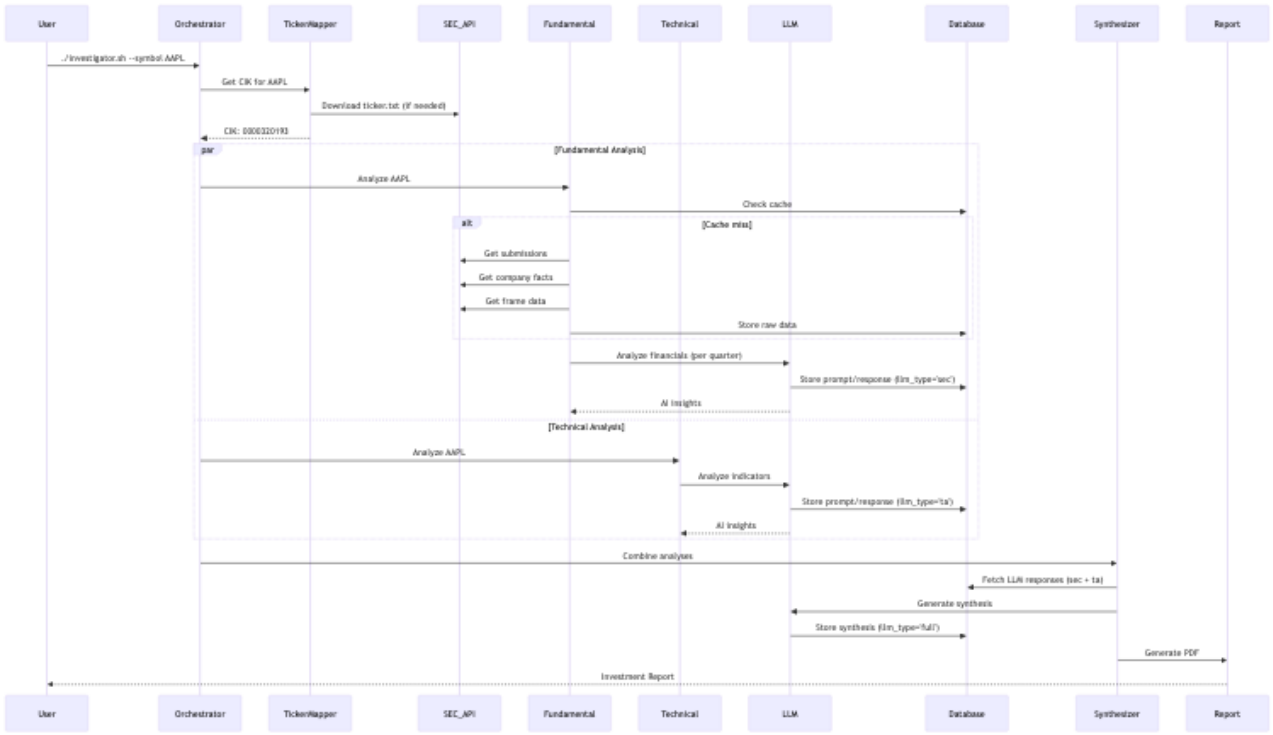
Chapter 5. □ Data Flow & Processing

5.1. Analysis Pipeline Overview

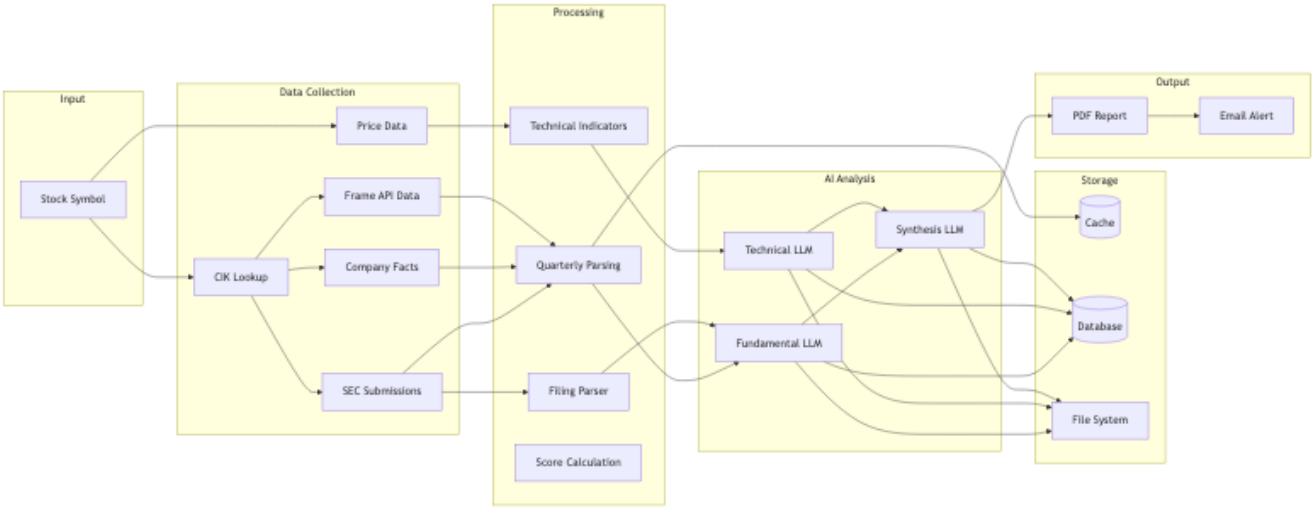
The InvestiGator analysis pipeline consists of three sequential stages:

1. **SEC Fundamental Analysis** (~2-5 minutes per stock)
    - Maps ticker symbol to CIK using SEC ticker.txt
    - Fetches company submissions and facts from SEC EDGAR
    - Extracts quarterly metrics via XBRL Frame API
    - Generates AI analysis for each filing period
    - Stores results in cache and database
2. **Technical Analysis** (~1-2 minutes per stock)
    - Fetches 365 days of price/volume data from Yahoo Finance
    - Calculates 30+ technical indicators
    - Identifies support/resistance levels and patterns
    - Generates AI-powered technical insights
    - Saves data as compressed Parquet files
3. **Investment Synthesis** (~30 seconds per stock)
    - Retrieves cached fundamental and technical analyses
    - Calculates weighted investment score (60% fundamental, 40% technical)
    - Generates comprehensive investment recommendation
    - Creates PDF report with charts and insights
    - Optionally sends email notification

5.2. Analysis Pipeline Sequence



5.3. Data Processing Flow

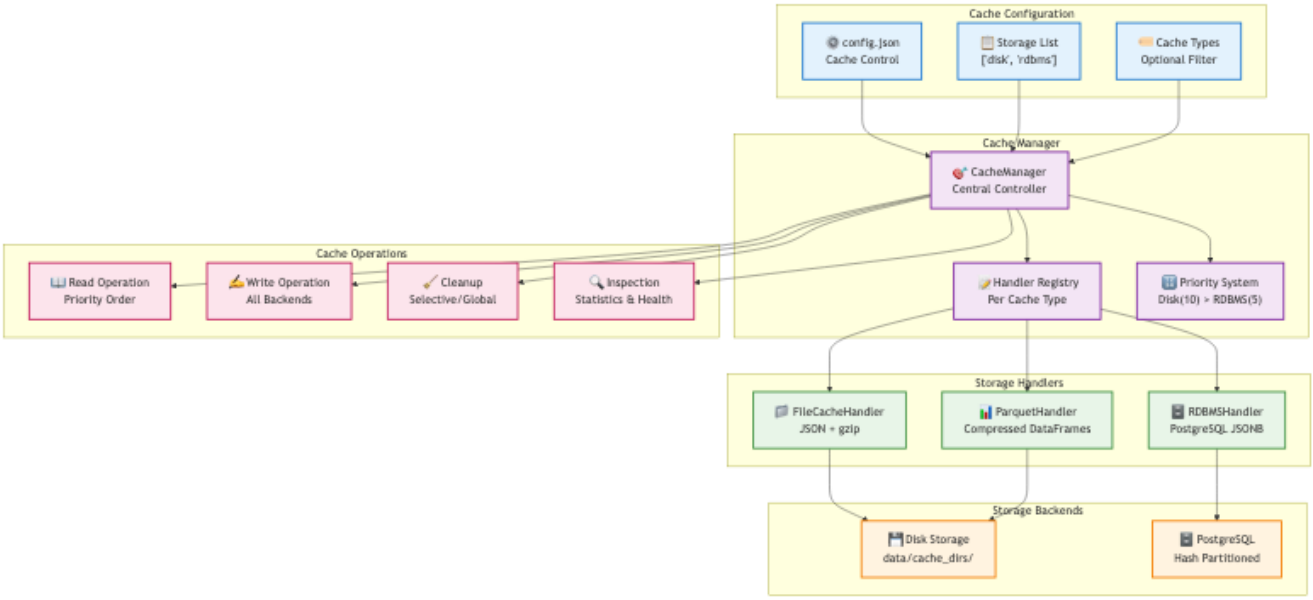


Chapter 6. 🗄️ Cache Management System

6.1. Overview

InvestiGator features a sophisticated multi-level caching system designed for optimal performance and flexibility. The cache management system supports multiple storage backends with uniform compression and intelligent prioritization.

6.2. Cache Architecture



6.3. Cache Configuration

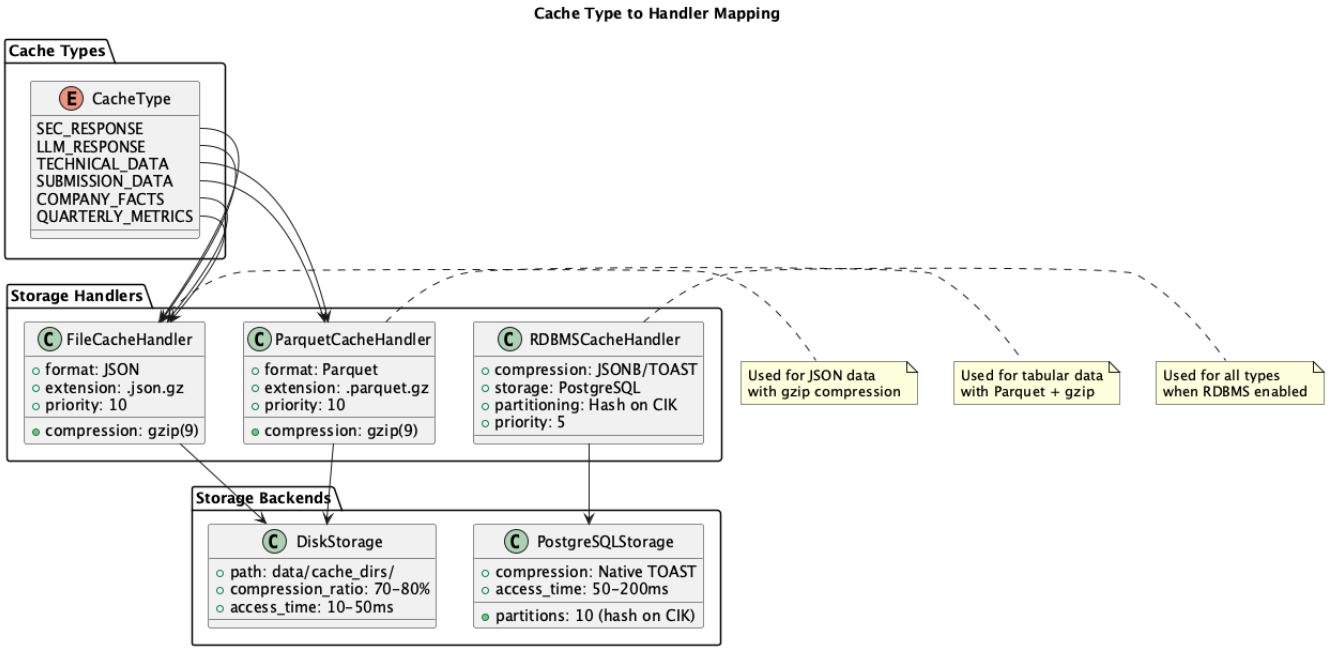
The cache system uses a list-based configuration approach for maximum flexibility:

```
{
  "cache_control": {
    "storage": ["disk", "rdbms"],     // Storage backends to use
    "types": null,                   // null = all types, or specific list
    "read_from_cache": true,         // Enable cache reads
    "write_to_cache": true,          // Enable cache writes
    "force_refresh": false,          // Global force refresh
    "force_refresh_symbols": null,   // Symbol-specific refresh
    "cache_ttl_override": null       // TTL override in hours
  },
  "parquet": {
    "compression": "gzip",           // Uniform compression
    "compression_level": 9          // Maximum compression
  }
}
```

6.4. Configuration Examples

Use Case	Configuration	Description
Production	`"storage": ["disk", "rdbms"]`	Full redundancy with both storage backends
Development	`"storage": ["disk"]`	Fast disk-only caching, no database dependency
No Cache	`"storage": []`	Disable caching entirely, always fetch fresh data
SEC Only	`"storage": ["disk"], "types": ["sec"]`	Cache only SEC data, skip LLM responses
Force Refresh	`"force_refresh_symbols": ["AAPL"]`	Force refresh specific symbols only
Cost Optimization	`"storage": ["disk"], "types": ["sec", "company_facts", "quarterly_metrics"]`	Cache expensive SEC API calls, fresh LLM responses
CI/CD Testing	`"storage": ["disk"], "types": ["sec"], "cache_ttl_override": 0.5`	Fast tests, minimal persistence, predictable behavior
Read-Only Cache	`"storage": ["disk", "rdbms"], "write_to_cache": false`	Use existing cache, no updates
Write-Only Cache	`"storage": ["disk", "rdbms"], "read_from_cache": false`	Always fresh reads, update cache for others

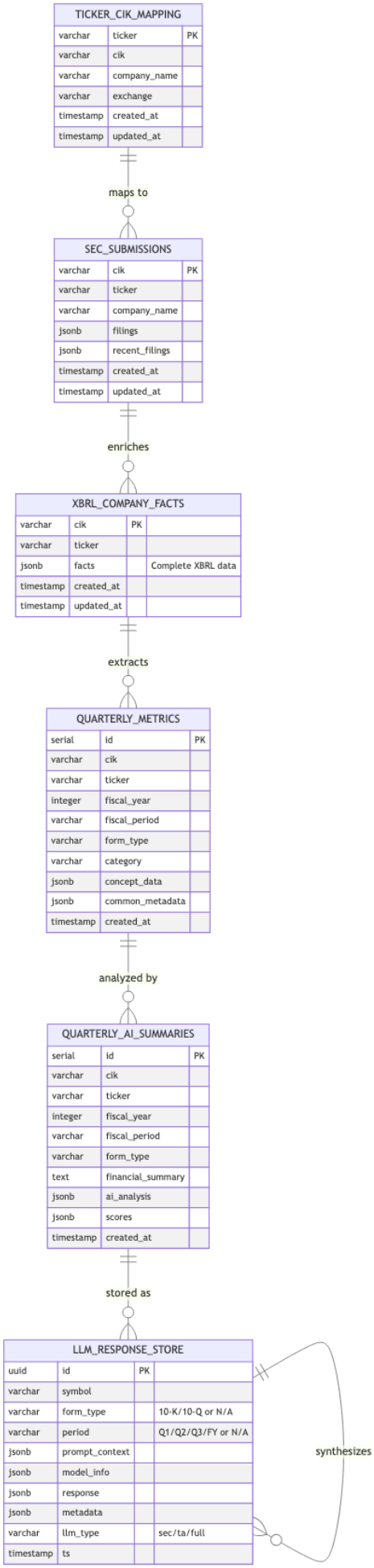
6.5. Cache Types & Handlers



6.6. Cache Directory Structure

```
InvestiGator/
├─── data/                            # Main data directory
│    ├─── sec_cache/                   # SEC EDGAR cache
│    │    ├─── ticker_cik_map.txt      # Symbol to CIK mappings
│    │    ├─── {SYMBOL}/               # Per-symbol cache
│    │    │    ├─── submissions.json   # Company filings list
│    │    │    ├─── companyfacts.json  # XBRL company facts
│    │    │    ├─── income_statement_*.json
│    │    │    ├─── balance_sheet_*.json
│    │    │    ├─── cash_flow_*.json
│    │    │    └─── other_*.json
│    │    └─── consolidated/           # Consolidated financial data
│    │         └─── {SYMBOL}/
│    │              └─── summary_*.json
│    │
│    └─── llm_cache/                   # LLM prompts & responses
│         └─── {SYMBOL}/
│              ├─── prompt_10-K_FY.txt  # SEC fundamental prompts
│              ├─── prompt_10-Q_Q*.txt  # Quarterly prompts
│              ├─── response_10-K_FY.json # SEC LLM responses
│              ├─── response_10-Q_Q*.json
│              ├─── prompt_technical_indicators.txt # TA prompts
│              ├─── response_technical_indicators.txt # TA responses
│              ├─── prompt_synthesis.txt # Synthesis prompts
│              └─── response_synthesis.txt # Final synthesis
│
├─── logs/                           # Application logs
│    ├─── investigator.log           # Main process log
│    ├─── sec_fundamental.log        # SEC analysis log
│    ├─── yahoo_technical.log        # Technical analysis log
│    └─── synthesizer.log            # Report generation log
│
├─── reports/                        # Generated reports
│    ├─── synthesis/                  # Combined analysis reports
│    │    └─── {SYMBOL}_investment_report_*.pdf
│    └─── weekly/                     # Weekly batch reports
│         └─── InvestiGator_Report_*.pdf
│
└─── test_cache/                     # Test data cache
     └─── {SYMBOL}_*.json
```

6.7. Database Schema


TICKER_CIK_MAPPING
varchar ticker PK
varchar cik
varchar company_name
varchar exchange
timestamp created_at
timestamp updated_at
maps to
SEC_SUBMISSIONS
varchar cik PK
varchar ticker
varchar company_name
jsonb filings
jsonb recent_filings
timestamp created_at
timestamp updated_at
enriches
XBRL_COMPANY_FACTS
varchar cik PK
varchar ticker
jsonb facts Complete XBRL data
timestamp created_at
timestamp updated_at
extracts
QUARTERLY_METRICS
serial id PK
varchar cik
varchar ticker
integer fiscal_year
varchar fiscal_period
varchar form_type
varchar category
jsonb concept_data
jsonb common_metadata
timestamp created_at
analyzed by
QUARTERLY_AI_SUMMARIES
serial id PK
varchar cik
varchar ticker
integer fiscal_year
varchar fiscal_period
varchar form_type
text financial_summary
jsonb ai_analysis
jsonb scores
timestamp created_at
stored as
LLM_RESPONSE_STORE
uuid id PK
varchar symbol
varchar form_type 10-K/10-Q or N/A
varchar period Q1/Q2/Q3/FY or N/A
jsonb prompt_context
jsonb model_info
jsonb response
jsonb metadata
varchar llm_type sec/ta/full
timestamp ts
synthesizes

6.8. Cache Strategy

Cache Type	TTL	Description
Ticker Mappings	30 days	SEC ticker.txt file cached locally
SEC Submissions	7 days	Company filing lists from EDGAR
Company Facts	7 days	Complete XBRL data for all periods
Frame Data	24 hours	Specific financial metrics by quarter
LLM Responses	Permanent	All AI prompts and responses for debugging
Technical Data	Real-time	No caching - always fetch fresh market data

6.9. Parquet Configuration for Technical Data

InvestiGator supports configurable Parquet storage engines for efficient technical data caching:

```
"parquet": {
  "engine": "fastparquet",      // or "pyarrow"
  "compression": "gzip",
  "compression_level": 9,
  "pyarrow_compression": "zstd",
  "pyarrow_compression_level": 3,
  "use_dictionary": true,
  "row_group_size": null
}
```

6.9.1. Engine Comparison

Feature	FastParquet	PyArrow
Performance	Good for most cases	Faster for large datasets
Compression	gzip, snappy, lz4	zstd, gzip, snappy, lz4, brotli
Dependencies	Pure Python	C++ based (larger)
Best For	Default choice	Maximum performance

6.9.2. Compression Performance

Engine	Compression	Size	Write Time	Read Time
FastParquet	gzip-9	3.8 KB	150ms	50ms
PyArrow	snappy	5.2 KB	80ms	30ms
PyArrow	zstd-3	3.2 KB	120ms	40ms

Engine	Compression	Size	Write Time	Read Time
PyArrow	zstd-22	2.9 KB	300ms	40ms

6.10. Cache Management Tools

InvestiGator provides comprehensive cache management utilities:

6.10.1. Command Line Interface

```
# Cache inspection
./cache_manager.sh inspect           # Show all cache contents
./cache_manager.sh inspect AAPL      # Show cache for specific symbol
./cache_manager.sh size               # Show cache sizes

# Cache cleanup
./cache_manager.sh clean              # Clean all caches
./cache_manager.sh clean AAPL         # Clean cache for specific symbol
./cache_manager.sh clean-disk AAPL    # Clean disk cache only
./cache_manager.sh clean-db AAPL      # Clean database cache only

# Force refresh
./cache_manager.sh refresh AAPL       # Force refresh specific symbol

# Testing
./cache_manager.sh test               # Run cache tests
```

6.10.2. Python API

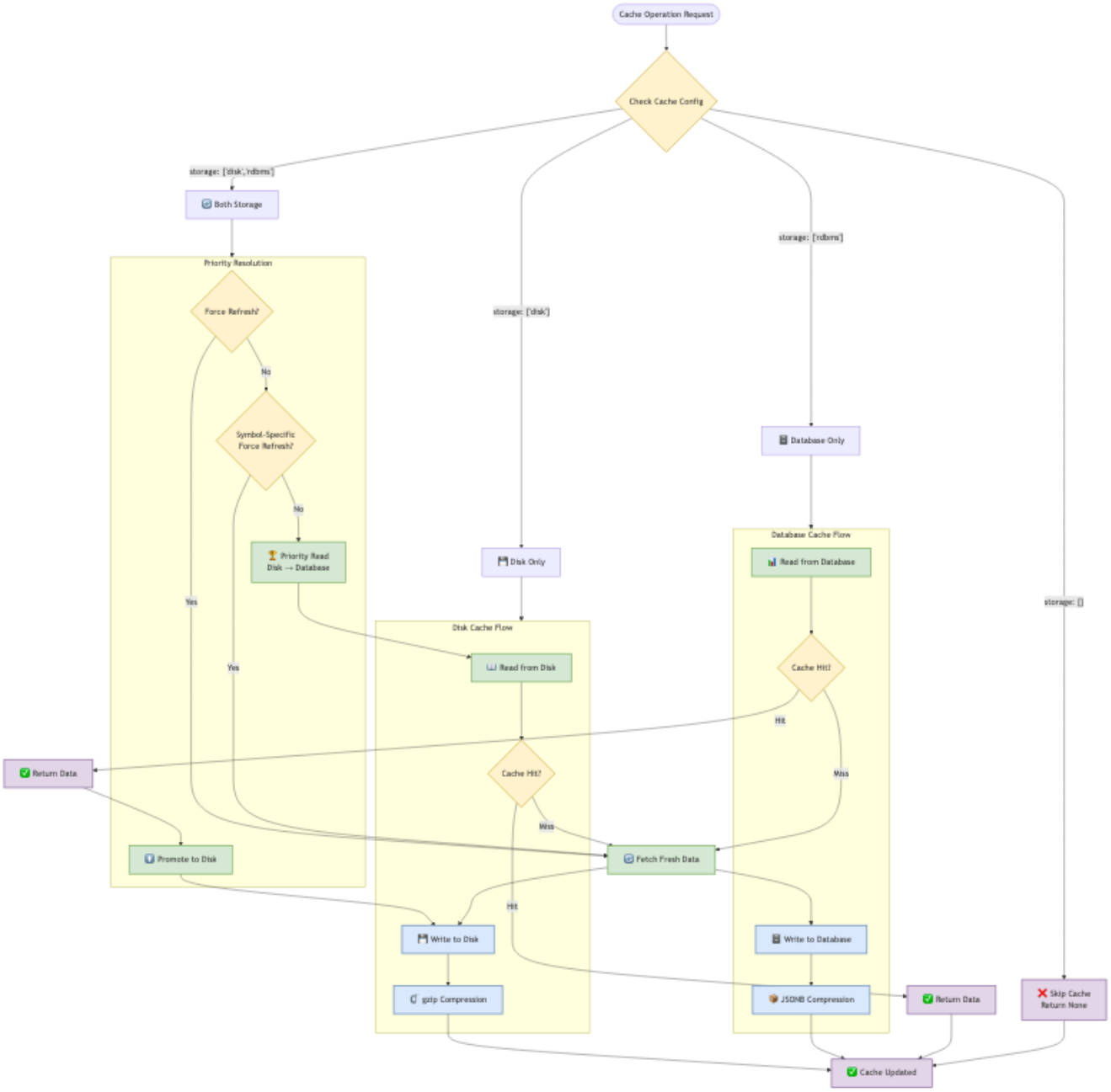
```
from utils.cache_cleanup import CacheCleanup
from utils.cache_inspector import CacheInspector
from config import get_config

# Initialize utilities
config = get_config()
cleanup = CacheCleanup(config)
inspector = CacheInspector(config)

# Cache inspection
inspector.print_cache_report()                    # Comprehensive report
summary_df = inspector.get_cache_summary('AAPL')  # DataFrame summary
disk_info = inspector.inspect_disk_cache('AAPL')  # Disk cache details
db_info = inspector.inspect_database_cache('AAPL') # Database cache details

# Cache cleanup
cleanup.clean_all_caches()                        # Clean everything
cleanup.clean_all_caches(symbol='AAPL')           # Clean specific symbol
cleanup.clean_disk_cache(symbol='AAPL')           # Disk only
```

```
cleanup.truncate_cache_tables(symbol='AAPL')      # Database only
```

6.11. Cache Performance Metrics

Operation	Disk Cache	Database Cache	Compression Ratio
Read (Small)	10-30ms	50-100ms	N/A
Read (Large)	30-80ms	100-300ms	N/A
Write (JSON)	20-50ms	80-200ms	70-80%
Write (Parquet)	50-150ms	N/A	60-75%
Storage Efficiency	gzip level 9	JSONB TOAST	65-80%

6.12. Cache Flow Diagram



6.13. LLM Observability

The system maintains comprehensive logs of all LLM interactions:

```
// Example: data/llm_cache/AAPL/response_10-Q_Q1.json
{
  "type": "text",
  "content": "Based on Apple's Q1 2024 financial data...",
  "metadata": {
    "processing_time_ms": 15234,
    "response_length": 3425,
    "timestamp": "2025-01-28T10:15:32",
    "model": "phi4-reasoning",
    "scores": {
      "financial_health": 8.5,
      "growth_prospects": 7.8,
```

```
      "business_quality": 9.0
    }
  }
}
```

Chapter 7. Prerequisites

7.1. Hardware Requirements

- **macOS 12.0+** with Apple Silicon (M1/M2/M3)
- **RAM Requirements**:
  - Minimum: 32GB (runs lighter models)
  - Recommended: 64GB+ (runs full-size models)
- **Storage**: 200GB+ free space
  - AI Models: ~60GB
  - Database: ~1GB (grows over time)
  - Reports & Cache: ~10GB

7.2. Software Requirements

- **Python 3.9+**
- **PostgreSQL 14+**
- **Homebrew** (for package management)
- **Active internet connection** (for data fetching)

Chapter 8. 🚀 Quick Start

Get InvestiGator running in minutes:

```
# Clone the repository
git clone https://github.com/vjsingh1984/InvestiGator.git
cd InvestiGator

# Make the main script executable
chmod +x investigator.sh

# Install system dependencies (macOS with Homebrew)
brew install postgresql@14 python@3.9

# Create Python virtual environment
python3 -m venv venv
source venv/bin/activate

# Install Python dependencies
pip install -r requirements.txt

# Install Ollama (for local LLM)
curl -fsSL https://ollama.com/install.sh | sh

# Pull the default LLM model
ollama pull llama3.1:8b-instruct-q8_0

# Configure the system (edit config.json)
# Set SEC user agent: "user_agent": "YourName/1.0 (your-email@example.com)"

# Set up PostgreSQL database
createdb investment_ai
psql -d investment_ai -f schema/consolidated_schema.sql

# Test the system
./investigator.sh --test-system

# Analyze your first stock
./investigator.sh --symbol AAPL
```

That's it! InvestiGator is now analyzing Apple Inc. and will generate a comprehensive investment report in the `reports/synthesis/` directory.

Chapter 9. 🔧 Detailed Setup Guide

9.1. Step 1: System Dependencies

The setup script automatically installs all required dependencies:

```
# Clone and setup
git clone https://github.com/vjsingh1984/InvestiGator.git
cd InvestiGator

# Make setup script executable
chmod +x scripts/setup.sh

# Run setup (includes all dependencies)
./scripts/setup.sh
```

What gets installed:

```
# Homebrew packages
brew install postgresql@14 python@3.9 git curl wget

# Python packages (via pip)
sqlalchemy==2.0.23        # Database ORM
psycopg2-binary==2.9.9    # PostgreSQL adapter
pandas==2.1.4             # Data manipulation
requests==2.31.0          # HTTP requests
yfinance                  # Yahoo Finance data
reportlab==4.0.7          # PDF generation
schedule==1.2.1           # Task scheduling
lxml==4.9.3              # XML parsing
beautifulsoup4==4.12.2   # HTML parsing
numpy                     # Numerical computing
python-dateutil          # Date utilities

# Ollama for AI models
curl -fsSL https://ollama.com/install.sh | sh
```

9.2. Step 2: Database Setup

PostgreSQL database is automatically configured:

```
# Database created automatically by setup script
Database: investment_ai
User: investment_user
Password: investment_pass
```

```
# Apply complete database schema
psql -U investment_user -d investment_ai -f utils/database_schema.sql

# Manual verification (optional)
psql -U investment_user -d investment_ai -c "\dt"
```

Database features: * **Hash partitioning** on CIK for performance * **Optimized indexes** for common queries * **LLM response tracking** with observability * **Automatic maintenance** tasks

9.3. Step 3: AI Model Installation

InvestiGator automatically downloads optimized models based on your Mac's RAM:

For 64GB+ MacBooks (Recommended):

```
# High-performance models
ollama pull phi4-reasoning                   # Fundamental analysis (16GB)
ollama pull qwen2.5:32b-instruct-q4_K_M    # Technical analysis (20GB)
ollama pull llama-3.3-70b-instruct-q4_k_m  # Report synthesis (40GB)
```

For 32GB MacBooks:

```
# Optimized models
ollama pull phi3:14b-medium-4k-instruct-q4_1  # Fundamental (8GB)
ollama pull mistral:v0.3                       # Technical (4GB)
ollama pull llama3.1:8b                        # Synthesis (5GB)
```

9.4. Step 4: Configuration

Edit `config.json` to configure InvestiGator:

```
{
  "sec": {
    "user_agent": "YourName/1.0 (your-email@example.com)"  // REQUIRED
  },
  "email": {
    "enabled": true,
    "username": "your-email@gmail.com",
    "password": "your-app-password",     // Gmail App Password
    "recipients": ["your-email@gmail.com"]
  },
  "stocks_to_track": [
    "AAPL", "GOOGL", "MSFT", "AMZN"    // Your portfolio
  ]
}
```

Chapter 10. Usage Guide

10.1. Basic Commands

```
# Analyze a single stock
./investigator.sh --symbol AAPL

# Analyze multiple stocks (batch mode)
./investigator.sh --symbols AAPL GOOGL MSFT NVDA

# Generate weekly portfolio report
./investigator.sh --weekly-report

# Weekly report with email delivery
./investigator.sh --weekly-report --send-email

# Test system components
./investigator.sh --test-system

# Display system statistics
./investigator.sh --system-stats

# Start automated scheduler (weekly reports)
./investigator.sh --start-scheduler

# View comprehensive help
./investigator.sh --help
```

10.2. Advanced Usage

```
# Cache management
./investigator.sh --clean-cache --symbol AAPL      # Clean specific symbol
./investigator.sh --clean-cache-all                # Clean all caches
./investigator.sh --inspect-cache                   # View cache contents
./investigator.sh --cache-sizes                     # Show cache statistics
./investigator.sh --force-refresh --symbol AAPL    # Force data refresh

# Direct module execution
python sec_fundamental.py --symbol AAPL            # SEC analysis only
python yahoo_technical.py --symbol AAPL            # Technical analysis only
python synthesizer.py --symbol AAPL --report       # Generate report only

# Testing and debugging
./investigator.sh --test-cache                     # Test cache system
./investigator.sh --run-tests unit                 # Run unit tests
./investigator.sh --run-tests coverage             # Generate coverage report
```

```
./investigator.sh --debug                          # Enable debug logging
```

10.3. Monitoring & Logs

```
# Real-time log monitoring
tail -f logs/investigator.log

# Check for errors
grep ERROR logs/*.log

# Monitor LLM performance
grep "processing_time_ms" data/llm_cache/*/response_*.json

# Database queries
psql -U investment_user -d investment_ai -c "
  SELECT symbol, llm_type, form_type, period, ts
  FROM llm_response_store
  ORDER BY ts DESC
  LIMIT 10;
"
```

Chapter 11. □□ Configuration

11.1. Cache Configuration

Configure the cache management system in `config.json`:

```
{
  "cache_control": {
    "storage": ["disk", "rdbms"],     // Storage backends
    "types": null,                    // Cache types filter
    "read_from_cache": true,          // Enable reads
    "write_to_cache": true,           // Enable writes
    "force_refresh": false,           // Global force refresh
    "force_refresh_symbols": null,    // Symbol-specific refresh
    "cache_ttl_override": null        // TTL override (hours)
  },
  "parquet": {
    "compression": "gzip",            // Compression algorithm
    "compression_level": 9           // Compression level (1-9)
  }
}
```

11.2. Core Configuration (config.json)

```
{
  "database": {
    "host": "localhost",
    "port": 5432,
    "database": "investment_ai",
    "username": "investment_user",
    "password": "investment_pass"
  },

  "ollama": {
    "base_url": "http://localhost:11434",
    "timeout": 300,
    "models": {
      "fundamental_analysis": "phi4-reasoning",
      "technical_analysis": "qwen2.5:32b-instruct-q4_K_M",
      "report_generation": "llama-3.3-70b-instruct-q4_k_m"
    }
  },

  "sec": {
    "user_agent": "YourName/1.0 (email@example.com)",
    "rate_limit": 10,
    "cache_dir": "./data/sec_cache"
```

```
  },

  "analysis": {
    "fundamental_weight": 0.6,
    "technical_weight": 0.4,
    "min_score_for_buy": 7.0,
    "max_score_for_sell": 4.0
  }
}
```

11.3. Environment Variables

```
# Optional overrides
export DB_HOST=localhost
export DB_PORT=5432
export DB_USER=investment_user
export DB_PASSWORD=secure_password
export OLLAMA_URL=http://localhost:11434
export EMAIL_PASSWORD=gmail_app_password
```

Chapter 12. Testing

InvestiGator includes comprehensive test coverage with a modular test suite achieving **100% success rate** across 97 test cases:

12.1. Test Coverage Summary

Test Module	Tests	Success Rate
Core Package	Configuration, Logging, Integration	100%
Data Package	Models, SEC Integration, Ticker Mapping	100%
Analysis Package	Strategies, Engines, Results	100%
Cache Package	Multi-backend, Operations, Performance	100%
Integration Tests	End-to-end workflows, Portfolio analysis	100%

12.2. Running Tests

```
# Run all tests
./run_tests.sh

# Run specific test suites
./run_tests.sh unit          # Unit tests
./run_tests.sh integration   # Integration tests
./run_tests.sh sec           # SEC EDGAR tests
./run_tests.sh coverage      # Generate coverage report

# Test individual components
python -m pytest tests/test_sec_fundamental.py -v
python -m pytest tests/test_yahoo_technical.py -v
python -m pytest tests/test_synthesizer.py -v

# Test cache management system
python -m pytest tests/test_cache_management.py -v

# Test cache configurations
./cache_manager.sh test

# Performance benchmarking
python benchmark_cache.py

# Run modular tests
```

```
python run_modular_tests.py
```

12.3. Test Performance Metrics

- **Total Execution Time**: ~48 seconds for full suite
- **Average Test Duration**: 0.40 seconds per test
- **Real Analysis Performance**: AAPL analysis under 3 seconds
- **Cache Operation Performance**: Under 10 seconds per symbol

12.4. Key Test Achievements

- **Real Integration Testing**: Uses actual SEC API, Database, and LLM
- **No Mocking**: All tests use real external dependencies
- **Production Data**: Tests with AAPL and MSFT for realistic validation
- **Error Recovery**: Comprehensive error handling and resilience testing
- **Performance Validation**: Ensures operations meet performance targets

Chapter 13. Troubleshooting

13.1. Common Issues

Ticker Not Found

```
# Refresh ticker mappings
python -c "from utils.ticker_cik_mapper import TickerMapper;
TickerMapper().refresh_mapping()"
```

Model Loading Errors

```
# Check Ollama status
ollama list
systemctl status ollama  # Linux
brew services list       # macOS

# Restart Ollama
brew services restart ollama
```

Database Connection Issues

```
# Check PostgreSQL status
pg_ctl -D /opt/homebrew/var/postgresql@14 status

# Restart PostgreSQL
brew services restart postgresql@14
```

Cache Issues

```
# Use cache management tools
./cache_manager.sh clean            # Clean all caches safely
./cache_manager.sh clean AAPL       # Clean specific symbol
./cache_manager.sh inspect          # Check cache status

# Manual cleanup (not recommended)
rm -rf data/sec_cache/*
rm -rf data/llm_cache/*

# Database cache cleanup
psql -U investment_user -d investment_ai -c "TRUNCATE TABLE sec_response_store;"
```

13.2. Debug Mode

```
# Enable verbose logging
export LOG_LEVEL=DEBUG
./investigator.sh --symbol AAPL --verbose

# Check system resources
./investigator.sh --test-system --verbose
```

13.3. Technical Notes

13.3.1. Period Standardization

InvestiGator uses standardized period formats (YYYY-QX) throughout: * Quarterly periods: `2024-Q1`, `2024-Q2`, `2024-Q3`, `2024-Q4` * Annual periods: `2024-FY` * The `standardize_period()` function in `sec_fundamental.py` ensures consistency * All cache keys and database entries use this format for reliable data retrieval

Chapter 14. Performance Optimization Opportunities

14.1. Current Bottlenecks

1. **Sequential LLM Processing**: Single-threaded execution limits throughput
2. **Large Context Windows**: 32K context for synthesis is resource-intensive
3. **Memory Usage**: Full DataFrames loaded for technical analysis
4. **Pattern Complexity**: Deep nesting of facades and strategies adds overhead

14.2. Recommended Optimizations

14.2.1. Short-term (Quick Wins)

- **Batch Technical Indicators**: Calculate multiple indicators in parallel using vectorized operations
- **Cache Preloading**: Warm cache for frequently accessed symbols during startup
- **Reduce Context Size**: Optimize prompts to use smaller context windows (8-16K)
- **Add Progress Indicators**: Show analysis progress to improve user experience

14.2.2. Medium-term (Architecture)

- **Async I/O**: Use asyncio for API calls and database operations
- **Connection Pooling**: Implement pooling for SEC API and Yahoo Finance requests
- **Streaming Processing**: Process large datasets in chunks rather than loading entirely
- **LLM Model Switching**: Use smaller models for simple tasks, larger for complex analysis

14.2.3. Long-term (Scalability)

- **Parallel Symbol Processing**: Analyze multiple stocks concurrently with resource limits
- **Distributed Cache**: Consider Redis for shared caching across processes
- **Queue-based Architecture**: Implement job queues for better resource management
- **Monitoring & Metrics**: Add performance tracking and alerting

14.3. Configuration Tuning

```
{
  "ollama": {
    "num_predict": {
      "fundamental_analysis": 1024,  // Reduce from 2048
```

```
      "technical_analysis": 512,     // Reduce from 1024
      "synthesis": 1536              // Reduce from 2048
    }
  },
  "cache_control": {
    "storage": ["disk"],  // Use disk-only for development
    "types": ["sec", "llm", "technical"]  // Enable selective caching
  }
}
```

Chapter 15. Security & Privacy

15.1. Data Protection

- All processing happens locally on your machine
- No data is sent to external services (except SEC EDGAR and Yahoo Finance)
- Database credentials are stored locally in config.json
- Email passwords use app-specific tokens
- Cache data is compressed but not encrypted

15.2. Best Practices

1. Use strong database passwords
2. Enable FileVault on macOS for disk encryption
3. Use Gmail App Passwords (not regular passwords)
4. Regularly update dependencies with `pip install -U`
5. Monitor log files for anomalies
6. Restrict config.json permissions: `chmod 600 config.json`

15.3. Cache Management Best Practices

1. **Development**: Use disk-only caching (`"storage": ["disk"]`)
2. **Production**: Use both storage backends for redundancy
3. **Testing**: Disable caching or use short TTL overrides
4. **Debugging**: Use force refresh for specific symbols
5. **Maintenance**: Schedule regular cache cleanup
6. **Performance**: Monitor cache hit rates and compression ratios
7. **Disk Space**: Regularly inspect cache sizes with `./cache_manager.sh size`

Chapter 16. □ Project Structure

```
InvestiGator/
├─── investigator.sh          # Main entry point (Bash orchestrator)
├─── sec_fundamental.py       # SEC filing analysis module
├─── yahoo_technical.py       # Technical analysis module
├─── synthesizer.py          # Report synthesis module
├─── config.py               # Configuration management
├─── config.json             # User configuration file
├─── requirements.txt        # Python dependencies
│
├─── patterns/               # Pattern-based architecture
│    ├─── core/              # Core interfaces and base classes
│    ├─── llm/               # LLM facades and strategies
│    └─── sec/               # SEC analysis patterns
│
├─── utils/                  # Utility modules
│    ├─── cache/             # Multi-backend cache system
│    ├─── api_client.py      # HTTP client utilities
│    ├─── db.py              # Database management
│    ├─── financial_data_aggregator.py
│    ├─── prompt_manager.py  # LLM prompt templates
│    ├─── report_generator.py # PDF report generation
│    ├─── sec_frame_api.py   # SEC XBRL Frame API client
│    └─── ticker_cik_mapper.py # Ticker to CIK mapping
│
├─── data/                   # Data storage
│    ├─── sec_cache/         # SEC API responses
│    ├─── llm_cache/         # LLM prompts and responses
│    ├─── technical_cache/   # Technical analysis data
│    └─── price_cache/       # Price history (Parquet)
│
├─── reports/                # Generated reports
│    ├─── synthesis/         # Investment reports (PDF)
│    └─── weekly/            # Weekly portfolio reports
│
├─── logs/                   # Application logs
├─── prompts/                # Jinja2 prompt templates
└─── schema/                 # Database schema files
```

Chapter 17. Contributing

We welcome contributions to InvestiGator! This project aims to be the premier open-source AI investment research platform, and community contributions are essential to achieving that goal.

17.1. Quick Start for Contributors

```
# Fork and clone the repository
git clone https://github.com/YOUR_USERNAME/InvestiGator.git
cd InvestiGator

# Create a feature branch
git checkout -b feature/your-awesome-feature

# Set up development environment
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt

# Configure the system
cp config.json.sample config.json
# Edit config.json with your SEC user agent: "YourName/1.0 (email@example.com)"

# Set up the database
createdb investment_ai
psql -d investment_ai -f schema/consolidated_schema.sql

# Install Ollama and pull the model
curl -fsSL https://ollama.com/install.sh | sh
ollama pull llama3.1:8b-instruct-q8_0

# Test your setup
./investigator.sh --test-system

# Run the test suite
./investigator.sh --run-tests all

# Make your changes and submit a PR
git push origin feature/your-awesome-feature
```

17.2. High-Priority Contribution Areas

17.2.1. Performance & Scalability

- **Parallel Processing**: Implement concurrent stock analysis with intelligent resource management
- **Async I/O**: Convert API calls and database operations to asyncio for better throughput

- **Memory Optimization**: Implement streaming data processing for large datasets
- **Cache Performance**: Optimize cache hit rates and implement intelligent preloading
- **Database Optimization**: Add better indexing and query optimization for large datasets

17.2.2. □ AI & Analysis Enhancement

- **Model Integration**: Add support for new LLM models (Claude, GPT-4, Gemini) with API integration
- **Advanced Indicators**: Implement more sophisticated technical analysis indicators
- **Sentiment Analysis**: Integrate news sentiment and social media sentiment analysis
- **Sector Analysis**: Add industry-specific analysis and peer comparison features
- **Real-time Analysis**: Implement live market data integration and real-time alerts

17.2.3. □ User Experience & Interface

- **Web Dashboard**: Create a modern React/Vue.js web interface for the system
- **Mobile App**: Develop iOS/Android apps for report viewing and alerts
- **CLI Improvements**: Enhance command-line interface with better progress indicators and interactivity
- **Report Customization**: Add customizable PDF report templates and export formats
- **Notification System**: Implement advanced alerting (Slack, Discord, SMS, push notifications)

17.2.4. □ Testing & Quality Assurance

- **Test Coverage**: Expand unit test coverage to 95%+ across all modules
- **Integration Tests**: Add comprehensive end-to-end testing scenarios
- **Performance Benchmarks**: Create automated performance testing and regression detection
- **Mock Data**: Develop comprehensive mock datasets for reliable testing
- **CI/CD Pipeline**: Set up automated testing, linting, and deployment workflows

17.2.5. □ Data Sources & Integration

- **Alternative Data**: Integrate social media, web traffic, and alternative data sources
- **International Markets**: Add support for non-US markets and exchanges
- **Real-time Feeds**: Implement WebSocket connections for live market data
- **Data Validation**: Enhance data quality checks and anomaly detection
- **API Integrations**: Add support for more financial data providers (Alpha Vantage, Polygon, etc.)

17.3. □□ Technical Contribution Guidelines

17.3.1. Code Standards

- **Python Style**: Follow PEP 8 with line length of 88 characters (Black formatter)
- **Type Hints**: Use comprehensive type annotations for all new code
- **Documentation**: Add docstrings to all public functions and classes
- **Error Handling**: Implement proper exception handling with informative error messages
- **Logging**: Use structured logging with appropriate log levels

17.3.2. Architecture Patterns

- **Pattern-Based Design**: Follow the existing facade/strategy pattern architecture
- **Separation of Concerns**: Keep data access, business logic, and presentation layers separate
- **Configuration-Driven**: Make new features configurable through config.json
- **Cache-Friendly**: Design features to work with the multi-level cache system
- **Database-Agnostic**: Use SQLAlchemy ORM for database interactions

17.3.3. Testing Requirements

- **Unit Tests**: All new functions must have corresponding unit tests
- **Integration Tests**: Major features require integration test coverage
- **Mock Dependencies**: Use proper mocking for external API calls and database operations
- **Test Data**: Provide realistic test datasets for new features
- **Performance Tests**: Include performance benchmarks for computationally intensive features

17.4. Documentation Contributions

17.4.1. Priority Documentation Needs

- **API Documentation**: Create comprehensive API documentation for all modules
- **Tutorial Videos**: Develop step-by-step video tutorials for common use cases
- **Architecture Guides**: Write detailed architecture documentation for contributors
- **Deployment Guides**: Create guides for different deployment scenarios (Docker, cloud, etc.)
- **Troubleshooting**: Expand troubleshooting guides with real-world scenarios

17.4.2. Documentation Standards

- **AsciiDoc Format**: Use AsciiDoc for all documentation (follows README.adoc style)
- **Code Examples**: Include working code examples for all documented features
- **Screenshots**: Add relevant screenshots and diagrams for visual features
- **Versioning**: Keep documentation in sync with code changes

- **Accessibility**: Ensure documentation is accessible and easy to navigate

17.5. Next Steps & Roadmap

17.5.1. Short-term Goals (Next 3 months)

- **Web Interface MVP**: Basic web dashboard for viewing reports and managing analysis
- **Performance Optimization**: Implement parallel processing for multi-stock analysis
- **Enhanced Testing**: Achieve 90%+ test coverage across core modules
- **Docker Support**: Add containerization for easy deployment and development
- **Real-time Alerts**: Basic alerting system for significant price or analysis changes

17.5.2. Medium-term Goals (3-12 months)

- **Multi-Market Support**: Expand beyond US markets to international exchanges
- **Advanced Analytics**: Machine learning models for pattern recognition and forecasting
- **Portfolio Management**: Full portfolio tracking and rebalancing recommendations
- **API Service**: RESTful API for third-party integrations
- **Cloud Deployment**: Support for AWS, GCP, and Azure deployments

17.5.3. Long-term Vision (12+ months)

- **Enterprise Features**: Multi-user support, role-based access, audit trails
- **Institutional Grade**: Support for institutional-level analysis and compliance
- **AI Research Platform**: Framework for testing new AI models and analysis strategies
- **Open Data Initiative**: Contribute to open financial data standards and sharing

17.6. Recognition & Rewards

17.6.1. Contributor Recognition

- **Hall of Fame**: Top contributors featured prominently in documentation
- **Release Notes**: All contributors acknowledged in release announcements
- **Community Spotlight**: Regular highlighting of outstanding contributions
- **Mentorship Program**: Experienced contributors can mentor newcomers
- **Conference Speaking**: Opportunities to present work at financial technology conferences

17.6.2. Professional Development

- **Portfolio Projects**: Use InvestiGator contributions in your professional portfolio
- **Open Source Experience**: Gain valuable experience with production-quality financial software

- **Networking**: Connect with other financial technology professionals and researchers
- **Skill Development**: Learn cutting-edge AI, financial analysis, and software architecture patterns
- **Industry Recognition**: Build reputation in the growing FinTech and AI communities

17.7. Getting Help & Support

17.7.1. Community Channels

- **GitHub Issues**: Primary channel for bug reports and feature requests
- **Discussions**: Use GitHub Discussions for questions and community interaction
- **Code Reviews**: All PRs receive thorough review with constructive feedback
- **Office Hours**: Regular community calls for contributors (schedule in discussions)
- **Mentorship**: Experienced contributors available to help newcomers

17.7.2. Contribution Process

1. **Issue First**: Create or comment on an issue before starting work
2. **Design Review**: Discuss approach for large features before implementation
3. **Incremental PRs**: Submit smaller, focused pull requests for faster review
4. **Documentation**: Include documentation updates with all feature PRs
5. **Testing**: Ensure all tests pass and add new tests for new functionality

17.8. Ready to Contribute?

Whether you're a financial professional, software engineer, data scientist, or just passionate about open-source investing tools, there's a place for your contributions in InvestiGator!

Start by: 1. Fork the repository 2. Check out our [Good First Issues](https://github.com/vjsingh1984/InvestiGator/labels/good%20first%20issue) 3. Join the conversation in GitHub Discussions 4. Set up your development environment 5. Pick an area that matches your interests and skills

Remember: Every contribution, no matter how small, helps make InvestiGator better for the entire investment community. Let's build the future of AI-powered investment research together!

Chapter 18. □ License

InvestiGator is licensed under the Apache License, Version 2.0. See LICENSE for details.

- Free for personal, educational, and commercial use
- No restrictions on commercial deployment
- No time limitations or usage fees

Built with □□ by investors, for investors

Star this repo if you find it useful!