Advanced Installation and Configuration Guide

This guide provides detailed information about advanced installation options, comprehensive configuration settings, and environment optimizations for the Fibonacci Harmonic Trading System.

Advanced Installation Options

Docker-based Installation

For containerized deployment, you can use Docker to isolate the application and its dependencies:

bash

Copy

```bash
# Build the Docker image
docker build -t fibonacci-harmonic-trading .

# Run the container
docker run -p 8050:8050 -v ./config:/app/config -v ./data:/app/data fibonacci-harmonic-trading
```

The repository includes a Dockerfile with the following contents:

dockerfile

Copy

```dockerfile
FROM python:3.9-slim

# Install system dependencies
RUN apt-get update && apt-get install -y \
    build-essential \
    libta-lib-dev \
    wget \
    && rm -rf /var/lib/apt/lists/*

# Install TA-Lib
RUN wget http://prdownloads.sourceforge.net/ta-lib/ta-lib-0.4.0-src.tar.gz && \
    tar -xvzf ta-lib-0.4.0-src.tar.gz && \
    cd ta-lib/ && \
    ./configure --prefix=/usr && \
    make && \
    make install && \
    cd .. && \
    rm -rf ta-lib ta-lib-0.4.0-src.tar.gz

# Set working directory
WORKDIR /app

# Copy requirements file
COPY requirements.txt .

# Install Python dependencies
RUN pip install --no-cache-dir -r requirements.txt
RUN pip install --upgrade --no-cache-dir git+https://github.com/rongardF/tvdatafeed.git

# Copy application code
COPY . .

# Expose port for web interface
EXPOSE 8050

# Run the application
CMD ["python", "run.py"]
```

GPU-Accelerated Installation

For systems with NVIDIA GPUs, you can enable GPU acceleration for faster FFT computation:

bash

Copy

```
# Install CUDA toolkit (Ubuntu)
sudo apt-get update
sudo apt-get install -y nvidia-cuda-toolkit

# Install required Python packages
pip install cupy-cuda11x  # Replace with appropriate version for your CUDA installation
pip install pycuda

# Enable GPU acceleration in config.json
# Set "use_gpu": true in the performance section
```

Development Environment with Pre-commit Hooks

For development, set up a more comprehensive environment with pre-commit hooks:

bash

Copy

```
# Create and activate virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install development dependencies
pip install -e ".[dev]"

# Install pre-commit hooks
pre-commit install
```

Comprehensive Configuration

The system uses a comprehensive configuration file in JSON format. Here's a fully annotated configuration with all available options:

json

Copy

```json
{
  "general": {
    "default_exchange": "NSE",          // Default exchange for symbols
    "default_interval": "daily",        // Default time interval
    "default_lookback": 5000,           // Default number of bars to fetch
    "symbols_file_path": "./data/symbols/default_symbols.csv",  // Path to symbols file
    "report_dir": "./reports",          // Directory for reports
    "default_source": "tradingview",    // Default data source
    "timezone": "Asia/Kolkata",         // Default timezone for data
    "application_name": "Fibonacci Harmonic Trading System",  // Application name
    "log_level": "INFO"                 // Logging level
  },

  "performance": {
    "use_gpu": false,                   // Enable GPU acceleration
    "max_workers": 5,                   // Maximum number of concurrent workers
    "cache_enabled": true,              // Enable data caching
    "optimize_fft": true,               // Use optimized FFT implementation
    "gpu_device_id": 0,                 // GPU device ID to use
    "parallel_processing": {
      "enabled": true,                  // Enable parallel processing
      "batch_size": 10                  // Batch size for parallel processing
    }
  },

  "notifications": {
    "telegram_enabled": true,           // Enable Telegram notifications
    "telegram_token": "YOUR_TELEGRAM_BOT_TOKEN",  // Telegram bot token
    "telegram_chat_id": "YOUR_CHAT_ID", // Telegram chat ID
    "email_enabled": false,             // Enable email notifications
    "email_settings": {
      "smtp_server": "smtp.gmail.com",  // SMTP server
      "smtp_port": 587,                 // SMTP port
      "sender_email": "your-email@gmail.com",  // Sender email
      "receiver_email": "your-email@gmail.com",  // Receiver email
      "password": "your-app-password"   // Email password or app password
    },
    "notification_schedule": {
      "daily_report": "17:00",          // Time for daily report (24-hour format)
      "intraday_updates": ["10:00", "14:00", "16:00"]  // Times for intraday updates
    },
```

```
    "alert_thresholds": {
      "min_signal_strength": 0.5,       // Minimum signal strength for alerts
      "min_signal_alignment": 0.7       // Minimum alignment for alerts
    }
  },

  "tradingview": {
    "username": "YOUR_TRADINGVIEW_USERNAME",  // TradingView username
    "password": "YOUR_TRADINGVIEW_PASSWORD",  // TradingView password
    "use_selenium": false,              // Use Selenium for login (for captcha handling)
    "selenium_path": "./drivers/chromedriver",  // Path to Selenium WebDriver
    "request_timeout": 30,              // Request timeout in seconds
    "max_retries": 3                    // Maximum number of retries
  },

  "data": {
    "cache_dir": "./data/cache",        // Directory for data cache
    "cache_expiry_days": 30,            // Cache expiry in days
    "local_data_dir": "./data/local",   // Directory for local data files
    "backup_dir": "./data/backup",      // Directory for data backups
    "sources": {
      "yahoo": {
        "enabled": true,                // Enable Yahoo Finance data source
        "default_suffix": {             // Default suffixes for exchanges
          "NSE": ".NS",
          "BSE": ".BO"
        }
      },
      "alpha_vantage": {
        "enabled": false,               // Enable Alpha Vantage data source
        "api_key": "YOUR_API_KEY"       // Alpha Vantage API key
      }
    }
  },

  "analysis": {
    "fib_cycles": [21, 34, 55, 89, 144, 233],  // Fibonacci cycle lengths to focus on
    "min_period": 10,                   // Minimum cycle period to detect
    "max_period": 250,                  // Maximum cycle period to detect
    "power_threshold": 0.2,             // Minimum power threshold for cycles
    "cycle_tolerance": 0.15,            // Tolerance for matching to Fibonacci cycles
    "crossover_lookback": 5,            // Lookback days for crossover influence
    "detrend_method": "diff",           // Method for detrending ("diff", "ema", "linear")
    "window_function": "hanning",       // Window function for FFT ("hanning", "hamming", "blac
```

```
    "gap_threshold": 0.01,              // Threshold for gap detection (as percentage)
    "signal_parameters": {
      "strength_scaling": 1.0,          // Scaling factor for signal strength
      "alignment_weight": 0.5,          // Weight for cycle alignment
      "trend_confirmation": true,       // Require trend confirmation
      "volume_confirmation": false      // Require volume confirmation
    },
    "position_sizing": {
      "default_risk_pct": 1.0,          // Default risk percentage per trade
      "max_position_pct": 5.0,          // Maximum position size as percentage of capital
      "position_scaling": {             // Position scaling based on confidence
        "high": 1.0,
        "medium": 0.75,
        "low": 0.5
      }
    },
    "market_regime": {
      "enabled": true,                  // Enable market regime detection
      "adx_threshold": 25,              // ADX threshold for trend detection
      "volatility_threshold": 1.5,      // Volatility threshold multiplier
      "regime_adjustment": true         // Adjust signals based on market regime
    }
  },

  "backtesting": {
    "default_capital": 100000.0,        // Default initial capital
    "default_position_size_pct": 10.0,  // Default position size percentage
    "commission_model": {
      "percentage": 0.05,               // Commission percentage
      "min_commission": 20.0,           // Minimum commission per trade
      "max_commission": 100.0           // Maximum commission per trade
    },
    "slippage_model": {
      "percentage": 0.1,                // Slippage percentage
      "fixed_pips": 2                   // Fixed slippage in pips/points
    },
    "optimization": {
      "method": "grid",                 // Optimization method ("grid", "random", "bayesian")
      "max_evaluations": 100,           // Maximum number of evaluations
      "parallel": true                  // Enable parallel optimization
    }
  },

  "web": {
```

```
    "host": "0.0.0.0",                  // Host to bind to
    "port": 8050,                       // Port to listen on
    "debug": true,                      // Enable debug mode
    "theme": "dark",                    // UI theme ("dark" or "light")
    "reload_interval": 300,             // Auto-reload interval in seconds
    "chart_defaults": {
      "lookback": 250,                  // Default lookback for charts
      "show_volume": true,              // Show volume by default
      "chart_type": "candle"            // Default chart type ("candle", "line", "ohlc")
    },
    "authentication": {
      "enabled": false,                 // Enable authentication
      "username": "admin",              // Default username
      "password": "password"            // Default password
    }
  },

  "report": {
    "template_dir": "templates/reports",  // Directory for report templates
    "report_dir": "reports",            // Directory for generated reports
    "default_format": "html",           // Default report format
    "include_header_footer": true,      // Include header and footer in reports
    "company_name": "Your Company",     // Company name for reports
    "logo_path": "assets/logo.png",     // Path to logo for reports
    "disclaimer_text": "This report is for informational purposes only..."  // Disclaimer text
  },

  "storage": {
    "db_path": "data/database/market_data.db",  // Path to SQLite database
    "backup_enabled": true,             // Enable database backups
    "backup_interval": "daily",         // Backup interval
    "max_backups": 7,                   // Maximum number of backups to keep
    "json_storage_path": "data/results",  // Path for JSON result storage
    "compress_old_results": true        // Compress old results to save space
  }
}
```

Environment-Specific Configurations

Production Environment

For production environments, use a more secure and optimized configuration:

json

Copy

```json
{
  "general": {
    "log_level": "WARNING"
  },
  "performance": {
    "use_gpu": true,
    "max_workers": 8
  },
  "web": {
    "debug": false,
    "authentication": {
      "enabled": true,
      "username": "SECURE_USERNAME",
      "password": "SECURE_PASSWORD"
    }
  },
  "storage": {
    "backup_enabled": true,
    "backup_interval": "hourly"
  }
}
```

Low-resource Environment

For systems with limited resources:

json

Copy

```json
{
  "performance": {
    "use_gpu": false,
    "max_workers": 2,
    "parallel_processing": {
      "enabled": false
    }
  },
  "analysis": {
    "fib_cycles": [21, 55, 144],
    "detrend_method": "ema",
    "window_function": "hanning"
  },
  "web": {
    "reload_interval": 600,
    "chart_defaults": {
      "lookback": 100
    }
  }
}
```

Automated Trading Environment

For automated trading integration:

json

Copy

```
{
  "general": {
    "timezone": "UTC",
    "log_level": "INFO"
  },
  "performance": {
    "use_gpu": true,
    "max_workers": 12
  },
  "analysis": {
    "power_threshold": 0.3,
    "cycle_tolerance": 0.1,
    "signal_parameters": {
      "strength_scaling": 1.2,
      "trend_confirmation": true,
      "volume_confirmation": true
    }
  },
  "trading": {
    "enabled": true,
    "mode": "live",
    "broker": "interactive_brokers",
    "broker_settings": {
      "host": "127.0.0.1",
      "port": 7496,
      "client_id": 1
    },
    "order_types": {
      "default": "market",
      "entry": "limit",
      "stop_loss": "stop",
      "take_profit": "limit"
    },
    "risk_management": {
      "max_open_positions": 5,
      "max_risk_per_trade": 1.0,
      "max_daily_drawdown": 3.0,
      "position_sizing": "risk_based"
    },
    "trading_hours": {
      "start": "09:15",
```

```
      "end": "15:30",
      "time_zone": "Asia/Kolkata"
    },
    "automation": {
      "scan_interval": 300,
      "signal_threshold": 0.5,
      "alert_on_fill": true,
      "alert_on_stop": true
    }
  }
}
```

Environment Variable Configuration

For sensitive information, you can use environment variables instead of storing them in the configuration file. The system will check for the following environment variables:

bash Copy

```bash
# TradingView credentials
export TRADINGVIEW_USERNAME="your_username"
export TRADINGVIEW_PASSWORD="your_password"

# Telegram configuration
export TELEGRAM_BOT_TOKEN="your_bot_token"
export TELEGRAM_CHAT_ID="your_chat_id"

# Email configuration
export EMAIL_PASSWORD="your_email_password"

# Database connection
export DB_PATH="path/to/database"

# Web authentication
export WEB_AUTH_USERNAME="admin"
export WEB_AUTH_PASSWORD="secure_password"
```

In your code or configuration, use a pattern like this to read from environment variables with fallbacks:

python

Copy

```python
import os

# Get configuration with environment variable override
def get_config_value(config, *keys, env_var=None, default=None):
    """Get a configuration value with environment variable override."""
    # Try environment variable first
    if env_var and env_var in os.environ:
        return os.environ[env_var]

    # Navigate through nested config
    current = config
    for key in keys:
        if key not in current:
            return default
        current = current[key]

    return current or default

# Example usage
username = get_config_value(
    config, 'tradingview', 'username',
    env_var='TRADINGVIEW_USERNAME',
    default='defaultuser'
)
```

Logging Configuration

For detailed logging, create a `logging.json` file:

json

Copy

```
{
  "version": 1,
  "disable_existing_loggers": false,
  "formatters": {
    "standard": {
      "format": "%(asctime)s [%(levelname)s] %(name)s: %(message)s",
      "datefmt": "%Y-%m-%d %H:%M:%S"
    },
    "detailed": {
      "format": "%(asctime)s [%(levelname)s] %(name)s (%(filename)s:%(lineno)d): %(message)s",
      "datefmt": "%Y-%m-%d %H:%M:%S"
    }
  },
  "handlers": {
    "console": {
      "class": "logging.StreamHandler",
      "level": "INFO",
      "formatter": "standard",
      "stream": "ext://sys.stdout"
    },
    "file_handler": {
      "class": "logging.handlers.RotatingFileHandler",
      "level": "DEBUG",
      "formatter": "detailed",
      "filename": "logs/app.log",
      "maxBytes": 10485760,
      "backupCount": 5,
      "encoding": "utf8"
    },
    "error_file_handler": {
      "class": "logging.handlers.RotatingFileHandler",
      "level": "ERROR",
      "formatter": "detailed",
      "filename": "logs/error.log",
      "maxBytes": 10485760,
      "backupCount": 5,
      "encoding": "utf8"
    }
  },
  "loggers": {
    "": {
```

```
      "handlers": ["console", "file_handler", "error_file_handler"],
      "level": "DEBUG",
      "propagate": true
    },
    "urllib3": {
      "level": "WARNING"
    },
    "matplotlib": {
      "level": "WARNING"
    }
  }
}
```

Load this configuration in your application:

python Copy

```python
import json
import logging.config
import os

def setup_logging(
    default_path='config/logging.json',
    default_level=logging.INFO,
    env_key='LOG_CFG'
):
    """Setup logging configuration"""
    path = os.getenv(env_key, default_path)
    if os.path.exists(path):
        with open(path, 'rt') as f:
            config = json.load(f)
        logging.config.dictConfig(config)
    else:
        logging.basicConfig(level=default_level)
```

Custom Data Sources

To add a custom data source, create a new module in the `data/sources` directory:

python

Copy

```python
# data/sources/custom_source.py
from typing import Optional, Dict
import pandas as pd
import requests
from datetime import datetime, timedelta

class CustomDataSource:
    """Custom data source implementation."""

    def __init__(self, config: Dict):
        """Initialize the data source with configuration."""
        self.config = config
        self.api_key = config.get('api_key', '')
        self.base_url = config.get('base_url', 'https://api.example.com')

    def is_available(self) -> bool:
        """Check if the data source is available."""
        try:
            response = requests.get(f"{self.base_url}/status", timeout=5)
            return response.status_code == 200
        except:
            return False

    def fetch_data(self,
                   symbol: str,
                   exchange: str,
                   interval: str,
                   lookback: int) -> Optional[pd.DataFrame]:
        """Fetch data from the custom source."""
        try:
            # Determine date range
            end_date = datetime.now()
            start_date = end_date - timedelta(days=lookback)

            # Format dates
            start_str = start_date.strftime('%Y-%m-%d')
            end_str = end_date.strftime('%Y-%m-%d')

            # Build request URL
            url = f"{self.base_url}/data"
            params = {
```

```
                'api_key': self.api_key,
                'symbol': symbol,
                'exchange': exchange,
                'interval': interval,
                'start_date': start_str,
                'end_date': end_str
            }

            # Make request
            response = requests.get(url, params=params, timeout=30)
            response.raise_for_status()

            # Parse response
            data = response.json()

            # Convert to DataFrame
            df = pd.DataFrame(data['data'])

            # Format columns
            df.rename(columns={
                'timestamp': 'date',
                'o': 'open',
                'h': 'high',
                'l': 'low',
                'c': 'close',
                'v': 'volume'
            }, inplace=True)

            # Convert timestamp to datetime
            df['date'] = pd.to_datetime(df['date'])
            df.set_index('date', inplace=True)

            return df

        except Exception as e:
            print(f"Error fetching data from custom source: {e}")
            return None
```

Then register your custom data source in the `data/fetcher.py` file:

python

Copy

```python
# In the _init_data_sources method of DataFetcher class
def _init_data_sources(self):
    """Initialize available data sources."""
    self.sources = {}

    # ... existing sources

    # Initialize custom data source if configured
    custom_config = self.config.get('data', {}).get('sources', {}).get('custom', {})
    if custom_config.get('enabled', False):
        try:
            from .sources.custom_source import CustomDataSource
            self.sources['custom'] = CustomDataSource(custom_config)
            self.logger.info("Custom data source initialized")
        except Exception as e:
            self.logger.error(f"Failed to initialize custom data source: {e}")
```

Database Migrations

For database schema updates, create migration scripts:

python

Copy

```python
# migrations/001_initial_schema.py
def up(cursor):
    """Apply migration"""
    # Create cache_metadata table
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS cache_metadata (
        id TEXT PRIMARY KEY,
        symbol TEXT,
        exchange TEXT,
        interval TEXT,
        source TEXT,
        start_date TEXT,
        end_date TEXT,
        last_updated TEXT,
        expires TEXT,
        row_count INTEGER
    )
    ''')

    # Create price_data table
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS price_data (
        id TEXT,
        timestamp TEXT,
        open REAL,
        high REAL,
        low REAL,
        close REAL,
        volume REAL,
        PRIMARY KEY (id, timestamp)
    )
    ''')

def down(cursor):
    """Roll back migration"""
    cursor.execute('DROP TABLE IF EXISTS cache_metadata')
    cursor.execute('DROP TABLE IF EXISTS price_data')
```

Create a migration manager:

python

Copy

```python
# utils/migration_manager.py
import os
import sqlite3
import importlib.util
import logging

class MigrationManager:
    """Manage database migrations."""

    def __init__(self, db_path, migrations_dir='migrations'):
        """Initialize the migration manager."""
        self.db_path = db_path
        self.migrations_dir = migrations_dir
        self.logger = logging.getLogger(__name__)

        # Create database directory if it doesn't exist
        os.makedirs(os.path.dirname(db_path), exist_ok=True)

        # Create migrations table if it doesn't exist
        self._init_migrations_table()

    def _init_migrations_table(self):
        """Initialize migrations tracking table."""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
        CREATE TABLE IF NOT EXISTS migrations (
            id INTEGER PRIMARY KEY,
            name TEXT,
            applied_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
        ''')

        conn.commit()
        conn.close()

    def get_applied_migrations(self):
        """Get list of applied migrations."""
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()
```

```
        cursor.execute('SELECT name FROM migrations ORDER BY id')
        applied = [row[0] for row in cursor.fetchall()]

        conn.close()
        return applied

    def get_available_migrations(self):
        """Get list of available migrations."""
        if not os.path.exists(self.migrations_dir):
            return []

        files = os.listdir(self.migrations_dir)
        migrations = [f[:-3] for f in files if f.endswith('.py')]
        return sorted(migrations)

    def get_pending_migrations(self):
        """Get list of pending migrations."""
        applied = self.get_applied_migrations()
        available = self.get_available_migrations()

        return [m for m in available if m not in applied]

    def apply_migration(self, migration_name):
        """Apply a specific migration."""
        module_path = os.path.join(self.migrations_dir, f'{migration_name}.py')

        # Load migration module
        spec = importlib.util.spec_from_file_location(migration_name, module_path)
        migration = importlib.util.module_from_spec(spec)
        spec.loader.exec_module(migration)

        # Apply migration
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        try:
            migration.up(cursor)

            # Record migration
            cursor.execute(
                'INSERT INTO migrations (name) VALUES (?)',
                (migration_name,)
            )
```

```
            conn.commit()
            self.logger.info(f"Applied migration: {migration_name}")

        except Exception as e:
            conn.rollback()
            self.logger.error(f"Failed to apply migration {migration_name}: {e}")
            raise

        finally:
            conn.close()

    def run_migrations(self):
        """Run all pending migrations."""
        pending = self.get_pending_migrations()

        if not pending:
            self.logger.info("No pending migrations")
            return

        self.logger.info(f"Applying {len(pending)} migrations")

        for migration in pending:
            self.apply_migration(migration)
```

Use the migration manager in your application:

python

Copy

```
# In the main application startup
from utils.migration_manager import MigrationManager

def initialize_database():
    """Initialize the database with migrations."""
    db_path = config.get('storage', {}).get('db_path', 'data/database/market_data.db')
    migrations_dir = 'migrations'

    # Run migrations
    migration_manager = MigrationManager(db_path, migrations_dir)
    migration_manager.run_migrations()
```