4-Day GitHub Contribution Plan

High-Frequency Limit Order Book Dynamics Project

Pre-Project Setup (Day 0 - Evening Before)

Person 1 (Data Engineer) - GitHub Repository Setup:

1. Create GitHub repository: orderbook-market-making
2. Initialize with Python .gitignore template
3. Create branch protection rules (require PR reviews)
4. Set up initial directory structure:

```
mkdir -p data/{raw,processed,simulated}
mkdir -p src/{data_pipeline,visualization,models,strategy,backtesting}
mkdir -p notebooks tests docs dashboard
touch requirements.txt README.md .gitignore
```

5. Create initial requirements.txt:

```
numpy==1.24.3
pandas==2.0.3
scipy==1.11.1
matplotlib==3.7.2
plotly==5.15.0
streamlit==1.25.0
statsmodels==0.14.0
jupyter==1.0.0
pytest==7.4.0
```

6. Push to main branch

7. Invite team members as collaborators

8. Create GitHub Project board with columns: Backlog, In Progress, Review, Done

All Team Members:

1. Clone repository locally

2. Create personal development branch: git checkout -b dev-[name]

3. Install dependencies: pip install -r requirements.txt

4. Test setup with: python -c "import numpy, pandas, scipy; print('Setup OK')"

Day 1: Foundation and Data Pipeline

Morning Session (9 AM - 1 PM)

Team Standup (9:00-9:15 AM)

• Review Day 1 objectives
• Assign specific tasks
• Set up communication channel (Discord/Slack)

Person	Task	Branch	Files to Create	Commits Due
Person 1 **Data Engineer**	• Download NSE sample LOB data • Create LOB data structure • Implement data loader	feature/lob-data	src/data_pipeline/lob_loader.py src/data_pipeline/lob_structure.py	12:30 PM
Person 2 **Visualization**	• Set up Streamlit dashboard • Create basic LOB plot • Implement bid-ask spread viz	feature/dashboard-v1	dashboard/app.py src/visualization/lob_plots.py	12:30 PM
Person 3 **Quant Modeler**	• Literature review: Cont-Stoikov paper • Implement mid-price calculator • Create order flow stats	feature/statistics	src/models/statistics.py notebooks/01_exploratory_analysis.ipynb	12:30 PM
Person 4 **Strategy Dev**	• Study Avellaneda-Stoikov paper • Design strategy interface • Create config file structure	feature/strategy-base	src/strategy/base.py config/strategy_params.yaml	12:30 PM

Detailed Task Breakdown:

Person 1 - Data Pipeline (Branch: feature/lob-data)

```
# File: src/data_pipeline/lob_structure.py
class LimitOrderBook:
    def __init__(self):
        self.bids = {}  # price -> quantity
        self.asks = {}
        self.timestamp = None

    def add_order(self, side, price, quantity):
        pass

    def cancel_order(self, side, price, quantity):
        pass

    def get_mid_price(self):
        pass

    def get_spread(self):
        pass

    def get_depth(self, levels=5):
        pass

# File: src/data_pipeline/lob_loader.py
import pandas as pd

def load_nse_data(filepath):
    """Load NSE tick data and convert to LOB snapshots"""
    pass

def simulate_lob_data(n_ticks=10000):
    """Generate synthetic LOB data if real data unavailable"""
    # Use zero-intelligence model or exponential arrival/cancellation
    pass
```

Commit Messages:

- "feat: implement LOB data structure with bid/ask queues"
- "feat: add NSE data loader with timestamp parsing"
- "feat: add synthetic LOB data generator for testing"

Person 2 - Dashboard (Branch: feature/dashboard-v1)

```
# File: dashboard/app.py
import streamlit as st
import plotly.graph_objects as go

st.title("Limit Order Book Analyzer")

# Sidebar controls
stock = st.sidebar.selectbox("Stock", ["RELIANCE", "TCS", "INFY"])
```

```python
# Main plot
fig = go.Figure()
# Add bid-ask spread visualization
st.plotly_chart(fig)

# File: src/visualization/lob_plots.py
def plot_lob_snapshot(lob, levels=10):
    """Create bid-ask ladder visualization"""
    pass

def plot_spread_evolution(timestamps, spreads):
    """Time series of bid-ask spread"""
    pass
```

Commit Messages:

- "feat: initialize Streamlit dashboard with basic layout"
- "feat: add LOB snapshot visualization with Plotly"
- "feat: add spread time-series plot"

Afternoon Session (2 PM - 6 PM)

Mid-Day Standup (2:00-2:10 PM)

- Demo morning progress
- Identify blockers
- Adjust afternoon tasks if needed

Person	Task	Integration Goal
Person 1	Add rolling statistics: volume, volatility Implement order flow imbalance (OFI)	Provide data to Person 2 dashboard
Person 2	Add interactive depth chart Implement real-time update simulation	Accept data from Person 1
Person 3	Calculate spread-return correlation Implement basic descriptive stats	Prepare for Hawkes model (Day 2)
Person 4	Design order execution simulator Create basic PnL tracker	Foundation for backtesting (Day 3)

Evening Session (7 PM - 10 PM)

Integration and Testing:

- 7:00 PM: All team members merge feature branches via PR
- 7:30 PM: Person 2 integrates all components into dashboard
- 8:00 PM: Team testing session - identify bugs
- 8:30 PM: Bug fixes and re-testing
- 9:00 PM: Final Day 1 commit to main branch
- 9:15 PM: Literature review time (all members)
- 9:45 PM: Day 1 retrospective and Day 2 planning

Day 1 End-of-Day Checklist:

✓ LOB data structure implemented and tested

✓ Dashboard shows live LOB visualization

✓ Basic statistics (mid-price, spread, OFI) calculated

✓ All code merged to main branch

✓ Team has read at least 1 core paper

Day 2: Statistical Modeling and Order Flow Analysis

Morning Session (9 AM - 1 PM)

Team Standup (9:00-9:15 AM)

Person	Task	Branch	Key Deliverable
Person 1	Create Hawkes process simulator Generate synthetic order arrivals	feature/hawkes-sim	Working Hawkes data generator
Person 2	Add Hawkes visualization to dashboard Plot intensity function	feature/hawkes-viz	Interactive Hawkes plots
Person 3	Implement Hawkes MLE estimation Parameter fitting algorithm	feature/hawkes-model	Calibrated Hawkes model
Person 4	Price impact regression model Estimate permanent/temporary impact	feature/price-impact	Impact coefficients

Person 3 - Hawkes Process (CRITICAL PATH)

```
# File: src/models/hawkes.py
import numpy as np
from scipy.optimize import minimize

class HawkesProcess:
    def __init__(self, mu=1.0, alpha=0.5, beta=1.0):
        self.mu = mu      # baseline intensity
        self.alpha = alpha  # excitation
        self.beta = beta   # decay

    def intensity(self, t, event_times):
        """Calculate intensity at time t given past events"""
        intensity = self.mu
        for ti in event_times[event_times < t]:
            intensity += self.alpha * np.exp(-self.beta * (t - ti))
        return intensity

    def log_likelihood(self, event_times):
        """Compute log-likelihood for MLE"""
        T = event_times[-1]
        n = len(event_times)

        # Log of intensity at each event
        log_sum = 0
        for i, ti in enumerate(event_times):
            log_sum += np.log(self.intensity(ti, event_times[:i]))

        # Compensator integral
        compensator = self.mu * T
        for ti in event_times:
            compensator += (self.alpha / self.beta) * (1 - np.exp(-self.beta * (T - ti)))

        return log_sum - compensator

    def fit(self, event_times):
```

```
"""Estimate parameters via MLE"""
def neg_log_likelihood(params):
    self.mu, self.alpha, self.beta = params
    return -self.log_likelihood(event_times)

result = minimize(neg_log_likelihood,
                 x0=[1.0, 0.5, 1.0],
                 bounds=[(0.01, None), (0, None), (0.01, None)])

self.mu, self.alpha, self.beta = result.x
return result
```

Testing:

- Generate synthetic Hawkes events
- Fit model and verify parameter recovery
- Plot residuals to check goodness-of-fit

Afternoon Session (2 PM - 6 PM)

Person	Task
Person 1	Implement VPIN (Volume-Synchronized PIN) calculator Validate against literature values
Person 2	Add VPIN heatmap to dashboard Create order flow toxicity alerts
Person 3	Run Hawkes model on real NSE data Perform residual diagnostics
Person 4	Build linear price impact model Regression: ΔPrice ~ OrderSize

Person 4 - Price Impact Model

```
# File: src/models/price_impact.py
import numpy as np
from scipy.stats import linregress

class PriceImpactModel:
    def __init__(self):
        self.permanent_coeff = None  # γ in Almgren-Chriss
        self.temporary_coeff = None  # η in Almgren-Chriss

    def fit_permanent_impact(self, order_sizes, price_changes):
        """Fit: ΔP = γ * volume"""
        slope, intercept, r_value, p_value, std_err = linregress(order_sizes,
price_changes)
        self.permanent_coeff = slope
        return {'gamma': slope, 'r_squared': r_value**2, 'p_value': p_value}

    def fit_temporary_impact(self, trading_rates, price_changes):
        """Fit: ΔP = η * dV/dt"""
        slope, intercept, r_value, p_value, std_err = linregress(trading_rates,
price_changes)
        self.temporary_coeff = slope
        return {'eta': slope, 'r_squared': r_value**2, 'p_value': p_value}
```

Day 2 Evening Integration (7 PM - 10 PM)

- Merge all feature branches
- Person 2 adds all new models to dashboard
- Create Jupyter notebook with statistical analysis
- Team review: Are models performing as expected?
- Final commit and Day 2 wrap-up

Day 3: Market-Making Strategy and Backtesting

Morning Session (9 AM - 1 PM)

Person	Task	Branch
Person 1	Build event-driven backtesting engine Handle order fills, cancellations	feature/backtest-engine
Person 2	Create backtest visualization Equity curve, drawdown charts	feature/backtest-viz
Person 3	Implement Avellaneda-Stoikov quoting logic Optimal bid/ask spreads	feature/market-making
Person 4	Add inventory risk management Position limits, PnL tracking	feature/risk-mgmt

Person 3 - Avellaneda-Stoikov Strategy (CRITICAL)

```
# File: src/strategy/avellaneda_stoikov.py
import numpy as np

class AvellanedaStoikovMarketMaker:
    def __init__(self, gamma=0.1, k=1.5, T=1.0):
        self.gamma = gamma  # risk aversion
        self.k = k         # order arrival rate
        self.T = T         # time horizon

    def reservation_price(self, mid_price, inventory, sigma, time_left):
        """Calculate reservation price r(t)"""
        return mid_price - inventory * self.gamma * sigma**2 * time_left

    def optimal_spread(self, sigma, time_left):
        """Calculate optimal bid-ask spread δ"""
        return self.gamma * sigma**2 * time_left + (2 / self.gamma) * np.log(1 + self.gamma / self.k)

    def quote(self, mid_price, inventory, sigma, time_left):
        """Generate bid and ask quotes"""
        r = self.reservation_price(mid_price, inventory, sigma, time_left)
        delta = self.optimal_spread(sigma, time_left)

        bid_price = r - delta / 2
        ask_price = r + delta / 2

        return {'bid': bid_price, 'ask': ask_price, 'reservation': r, 'spread': delta}

    def should_adjust_quotes(self, current_inventory, max_inventory):
        """Inventory risk check"""
        return abs(current_inventory) > 0.8 * max_inventory
```

Person 1 - Backtesting Engine

```
# File: src/backtesting/engine.py
from collections import deque

class BacktestEngine:
    def __init__(self, initial_capital=100000):
        self.capital = initial_capital
        self.inventory = 0
```

```python
        self.pnl_history = []
        self.trades = []

    def process_fill(self, side, price, quantity, timestamp):
        """Execute a fill and update state"""
        if side == 'buy':
            self.inventory += quantity
            self.capital -= price * quantity
        else:
            self.inventory -= quantity
            self.capital += price * quantity

        self.trades.append({
            'timestamp': timestamp,
            'side': side,
            'price': price,
            'quantity': quantity
        })

    def calculate_metrics(self):
        """Compute Sharpe, drawdown, etc."""
        pnl_series = np.array(self.pnl_history)
        returns = np.diff(pnl_series) / pnl_series[:-1]

        sharpe = np.mean(returns) / np.std(returns) * np.sqrt(252)
        max_dd = self._max_drawdown(pnl_series)

        return {'sharpe': sharpe, 'max_drawdown': max_dd}

    def _max_drawdown(self, equity_curve):
        """Calculate maximum drawdown"""
        peak = np.maximum.accumulate(equity_curve)
        drawdown = (equity_curve - peak) / peak
        return np.min(drawdown)
```

Afternoon Session (2 PM - 6 PM)

Integration and Testing:

Time	Activity	Owner
2:00-3:00	Connect strategy to backtesting engine	Person 3 + Person 1
3:00-4:00	Run first backtest on sample data	All team
4:00-5:00	Debug issues, fix edge cases	Person 4 + Person 1
5:00-6:00	Parameter optimization (grid search)	Person 3

Evening Session (7 PM - 10 PM)

- Run comprehensive backtests on multiple stocks
- Person 2: Create performance dashboard
- Analyze results: What works? What doesn't?
- Team discussion: Strategy improvements for Day 4
- Commit all code with proper documentation

Day 4: Analysis, Documentation, and Presentation

Morning Session (9 AM - 1 PM)

Person	Task
Person 1	Write comprehensive README.md Add setup instructions and usage examples
Person 2	Polish dashboard UI Add comparison charts (strategy vs. benchmark)
Person 3	Create technical report (methodology section) Document all formulas and algorithms
Person 4	Run sensitivity analysis Test different parameter combinations

Sensitivity Analysis Tasks (Person 4):

- Vary risk aversion γ: [0.01, 0.05, 0.1, 0.5, 1.0]
- Vary order arrival rate k: [0.5, 1.0, 1.5, 2.0]
- Test on different market conditions (high/low volatility days)
- Generate heatmap: Sharpe ratio vs. (γ, k)

README Structure (Person 1):

````
# Limit Order Book Market Making

## Overview
[Brief description of project]

## Features
- Real-time LOB visualization
- Hawkes process order flow modeling
- Avellaneda-Stoikov market making strategy
- Event-driven backtesting with realistic costs

## Installation
```bash
git clone https://github.com/yourteam/orderbook-market-making
cd orderbook-market-making
pip install -r requirements.txt
```

## Quick Start
```bash
streamlit run dashboard/app.py
```

## Results
| Metric | Value |
|--------|-------|
| Sharpe Ratio | 1.82 |
| Max Drawdown | -12.3% |
| Win Rate | 64.5% |
````

References

1. Avellaneda & Stoikov (2008)
2. Sirignano & Cont (2023)

Afternoon Session (2 PM - 6 PM)

Time	Activity	Owner
2:00-3:30	Create presentation slides (15-20 slides)	Person 3 + Person 4
3:30-4:30	Record demo video (2-3 min)	Person 2
4:30-5:30	Final testing and bug fixes	All team
5:30-6:00	Code review and cleanup	Person 1

Presentation Outline (IISc Symposium):

Slide 1-2: Title and Motivation

Slide 3-4: Market Microstructure Background

Slide 5-7: Technical Approach (Hawkes, A-S model)

Slide 8-10: Implementation and Dashboard Demo

Slide 11-14: Results and Performance Analysis

Slide 15-16: Sensitivity Analysis

Slide 17-18: Limitations and Future Work

Slide 19: Conclusion and Q&A

Evening Session (7 PM - 10 PM)

- 7:00 PM: Final presentation rehearsal (all team)
- 8:00 PM: Upload all materials to GitHub
- 8:30 PM: Create LinkedIn posts with demo video
- 9:00 PM: Final project retrospective
- 9:30 PM: Celebrate! 🎉

Git Workflow Best Practices

Branch Naming Convention:

- feature/[feature-name] - for new features
- bugfix/[issue-description] - for bug fixes
- docs/[doc-type] - for documentation
- refactor/[component] - for code cleanup

Commit Message Format:

```
# Good commit messages:
feat: implement Hawkes process MLE estimation
fix: resolve mid-price calculation bug in LOB class
docs: add mathematical formulation to README
refactor: simplify backtesting engine state management
perf: optimize order book update loop (2x faster)

# Bad commit messages:
updated stuff
fixed bug
asdf
changes
```

Pull Request Process:

1. Create feature branch from main
2. Make changes and commit regularly
3. Push to GitHub: git push origin feature/your-feature
4. Open PR on GitHub with description
5. Request review from at least 1 team member
6. Address review comments
7. Merge to main after approval
8. Delete feature branch

Daily Commit Schedule:

Time	Action	Who
12:30 PM	Morning session commits - work in progress	All
6:00 PM	Afternoon session commits - completed features	All
9:00 PM	Final daily commit + PR creation	All
9:30 PM	Code review and PR merging	Reviewer (rotates daily)

Code Quality Checklist

Before Each Commit:

□ Code runs without errors
□ Added docstrings to all functions
□ Removed debug print statements
□ Updated requirements.txt if new packages used
□ No hardcoded file paths (use config files)
□ Consistent naming conventions (snake_case for Python)
□ Added comments for complex logic

Before Pull Request:

□ Feature is complete and tested
□ No merge conflicts with main
□ Added unit tests (if applicable)
□ Updated documentation
□ PR description explains what and why
□ Linked to relevant issue/task

Before Final Submission:

□ All features merged to main
□ Dashboard runs smoothly
□ README is comprehensive
□ All dependencies listed
□ No sensitive data in repository
□ License file added (MIT recommended)
□ Demo video uploaded and linked
□ Presentation slides finalized

Emergency Protocols and Fallbacks

If NSE Data is Unavailable:

- Use Binance cryptocurrency order book API (real-time, free)
- Generate synthetic LOB data using zero-intelligence traders
- Use academic LOB datasets (LOBSTER database samples)

If Hawkes Model Doesn't Converge:

- Use simpler Poisson process as baseline
- Try different initial parameter values
- Reduce data complexity (filter outliers)
- Use pre-computed parameters from literature

If Behind Schedule:

Priority ranking (focus on these first):
1. LOB visualization dashboard (Must Have)
2. Basic market-making strategy (Must Have)
3. Backtesting with simple metrics (Must Have)
4. Hawkes process (Should Have)
5. Advanced visualizations (Nice to Have)
6. Sensitivity analysis (Nice to Have)

Resources and References

Key Papers (Download PDFs Day 0):

1. Avellaneda & Stoikov (2008) - High-Frequency Trading in a Limit Order Book
   https://www.math.nyu.edu/faculty/avellane/HighFrequencyTrading.pdf

2. Cont, Stoikov, Talreja (2010) - A Stochastic Model for Order Book Dynamics
   Available on SSRN

3. Cartea et al. (2015) - Algorithmic and High-Frequency Trading (Book)
   Cambridge University Press

4. Easley et al. (2012) - Flow Toxicity and Liquidity
   https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1695596

Code Examples and Tutorials:

- GitHub: awesome-quant (curated list of quant resources)
- QuantStart.com (tutorials on market making)
- Quantopian Lectures (archived on GitHub)
- arXiv.org (search "limit order book machine learning")

Data Sources:

- NSE India: https://www.nseindia.com/market-data/historical-data
- LOBSTER: https://lobsterdata.com (academic license available)
- Binance API: https://binance-docs.github.io/apidocs (crypto LOB)
- Kaggle: Search "limit order book datasets"

Communication Plan:

- Daily standups: 9:00 AM, 2:00 PM, 9:00 PM (15-10-5 min)
- Slack/Discord channel for async communication
- Google Meet for pair programming sessions
- GitHub Issues for task tracking
- Shared Google Doc for notes and decisions

Final Deliverables Checklist

GitHub Repository:

- ☐ All code committed to main branch
- ☐ README.md with clear setup instructions
- ☐ requirements.txt with all dependencies
- ☐ LICENSE file (MIT recommended)
- ☐ .gitignore properly configured
- ☐ No sensitive data or credentials
- ☐ All branches merged and cleaned up

Code:

- ☐ LOB data structure implementation
- ☐ Hawkes process model (MLE fitting)
- ☐ Avellaneda-Stoikov market making strategy
- ☐ Event-driven backtesting engine
- ☐ Streamlit dashboard with visualizations
- ☐ Statistical analysis notebooks
- ☐ Unit tests for critical functions

Documentation:

- ☐ Technical report (8-10 pages PDF)
- ☐ Mathematical formulations documented
- ☐ Code comments and docstrings
- ☐ Usage examples in README
- ☐ Results and analysis section

Presentation:

- ☐ Slide deck (15-20 slides)
- ☐ Demo video (2-3 minutes)
- ☐ Practice run completed
- ☐ Q&A preparation notes

Analysis:

- ☐ Backtest results on multiple stocks
- ☐ Performance metrics calculated (Sharpe, drawdown)
- ☐ Sensitivity analysis plots
- ☐ Model validation (Hawkes residuals)
- ☐ Statistical significance tests