Implementation Guide: Connecting Bot Metrics with Variable Weights

This guide will help you integrate the enhanced ranking system with your existing KnowDefeat project, connecting the `bot_metrics` table with the `variable_weights` table for dynamic bot ranking.

1. File Organization

Save the three new components in your project directory:

```
project/
  ├── bot_ranker.py            # Enhanced version
  ├── ai_weight_adjuster.py    # Enhanced version
  ├── weights_management_ui.py # New UI module
  ├── metrics_calculator.py    # Your existing file
  ├── metrics_updater.py       # Your existing file
  ├── trade_listener.py        # Your existing file
  ├── streamlit_app2.py        # Your existing Streamlit app
  └── ...
```

2. Database Setup

Connect to your PostgreSQL database and ensure you have the required tables:

```
-- Create variable_weights table if it doesn't exist
CREATE TABLE IF NOT EXISTS variable_weights (
    weight_id SERIAL PRIMARY KEY,
    variable_name VARCHAR(50) NOT NULL UNIQUE,
    weight DECIMAL(4,1) NOT NULL,
    description TEXT,
    last_updated TIMESTAMP DEFAULT NOW()
);

-- Create bot_rankings table if it doesn't exist
CREATE TABLE IF NOT EXISTS bot_rankings (
    ranking_id SERIAL PRIMARY KEY,
    bot_id INTEGER NOT NULL UNIQUE,
    rank_score DECIMAL(10,2) NOT NULL,
    timestamp TIMESTAMP DEFAULT NOW(),
    is_active BOOLEAN DEFAULT true
);

-- Initialize default weights (run once)
INSERT INTO variable_weights (variable_name, weight, description)
VALUES
    ('one_hour_performance', 15.0, 'Performance over the last hour'),
```

```
    ('two_hour_performance', 12.5, 'Performance over the last two hours'),
    ('one_day_performance', 10.0, 'Performance over the last day'),
    ('one_week_performance', 7.5, 'Performance over the last week'),
    ('one_month_performance', 5.0, 'Performance over the last month'),
    ('avg_win_rate', 10.0, 'Average win rate of trades'),
    ('avg_drawdown', 8.0, 'Average drawdown (lower is better)'),
    ('profit_per_second', 12.0, 'Average profit per second of trading'),
    ('price_model_score', 5.0, 'Score from price prediction model'),
    ('volume_model_score', 5.0, 'Score from volume prediction model'),
    ('price_wall_score', 3.0, 'Score based on order book price walls'),
    ('win_streak_2', 2.0, 'Frequency of 2-trade win streaks'),
    ('win_streak_3', 1.5, 'Frequency of 3-trade win streaks'),
    ('win_streak_4', 1.5, 'Frequency of 4-trade win streaks'),
    ('win_streak_5', 1.0, 'Frequency of 5-trade win streaks'),
    ('win_streak_6', 0.5, 'Frequency of 6-trade win streaks'),
    ('win_streak_7', 0.5, 'Frequency of 7-trade win streaks')
ON CONFLICT (variable_name) DO NOTHING;
```

3. Update Your Existing Code

Update `metrics_updater.py`

Modify your existing `metrics_updater.py` to include all metrics needed for ranking:

```
# In MetricsUpdater class, update the update_bot_metrics method:

async def update_bot_metrics(self, bot_id, ticker):
    # Calculate basic metrics (you already have these)
    one_hour_perf = await
self.metrics_calculator.calculate_one_hour_performance(bot_id, ticker)
    avg_win_rate = await self.metrics_calculator.calculate_avg_win_rate(bot_id,
ticker)

    # Calculate additional metrics needed for ranking
    two_hour_perf = await
self.metrics_calculator.calculate_two_hour_performance(bot_id, ticker)
    one_day_perf = await
self.metrics_calculator.calculate_one_day_performance(bot_id, ticker)
    one_week_perf = await
self.metrics_calculator.calculate_one_week_performance(bot_id, ticker)
    one_month_perf = await
self.metrics_calculator.calculate_one_month_performance(bot_id, ticker)
    profit_per_second = await
self.metrics_calculator.calculate_profit_per_second(bot_id, ticker)
    drawdown_info = await self.metrics_calculator.calculate_drawdowns(bot_id,
ticker)

    # Get algorithm ID (you may need to adjust this to get the correct algo_id)
    algo_id = 1  # Default value, replace with actual logic to get algo_id
```

```
    async with self.db_pool.acquire() as connection:
        # Update metrics in database with all needed values
        await connection.execute("""
            INSERT INTO bot_metrics (
                bot_id,
                ticker,
                algorithm_id,
                one_hour_performance,
                two_hour_performance,
                one_day_performance,
                one_week_performance,
                one_month_performance,
                avg_win_rate,
                avg_drawdown,
                profit_per_second,
                timestamp
            )
            VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11, NOW())
            ON CONFLICT (bot_id, algorithm_id) DO UPDATE
            SET
                one_hour_performance = $4,
                two_hour_performance = $5,
                one_day_performance = $6,
                one_week_performance = $7,
                one_month_performance = $8,
                avg_win_rate = $9,
                avg_drawdown = $10,
                profit_per_second = $11,
                timestamp = NOW();
        """,
        bot_id, ticker, algo_id,
        one_hour_perf, two_hour_perf, one_day_perf, one_week_perf,
one_month_perf,
        avg_win_rate, drawdown_info['avg_drawdown'], profit_per_second)

    # Calculate and update win streaks
    await self.metrics_calculator.calculate_and_insert_win_streaks(bot_id,
algo_id)

    logging.info(f"Updated metrics for bot {bot_id}, ticker {ticker}")
```

Update `trade_listener.py`

Modify your trade listener to trigger bot ranking after updating metrics:

```
# In TradeListener class, update the process_trade method:

async def process_trade(self, trade):
    # Mark trade as processed
    async with self.db_pool.acquire() as connection:
```

```
            await connection.execute("""
                UPDATE trades
                SET processed = TRUE
                WHERE trade_id = $1;
            """, trade['trade_id'])

        # Trigger metric recalculation
        await self.recalculate_metrics(trade['bot_id'], trade['ticker'])

        # NEW: Trigger bot ranking update
        await self.update_bot_rankings(trade['bot_id'])

# Add this new method to TradeListener class:

async def update_bot_rankings(self, bot_id):
    try:
        # Create a BotRanker instance and rank the bots
        from bot_ranker import BotRanker
        ranker = BotRanker(self.db_pool)
        await ranker.rank_bots()
        logging.info(f"Updated bot rankings after processing trade for bot
{bot_id}")
    except Exception as e:
        logging.error(f"Error updating bot rankings: {str(e)}")
```

4. Update the Streamlit App

Modify your `streamlit_app2.py` file to include the new weights management UI:

```
# Add to the imports at the top of your file:
from weights_management_ui import WeightsManagementUI

# Define your database configuration
DB_CONFIG = {
    'user': 'clayb',
    'password': 'musicman',
    'database': 'tick_data',
    'host': 'localhost'
}

# In your main function, add the ranking system tab:
def main():
    st.title("KnowDefeat Trading System")

    # Create tabs (modify your existing tabs if necessary)
    tab_dashboard, tab_controls, tab_rankings, tab_settings = st.tabs([
        "Dashboard", "Bot Control", "Rankings", "Settings"
    ])

    with tab_dashboard:
```

```
        # Your existing dashboard code
        st.header("Dashboard")
        # ...

    with tab_controls:
        # Your existing bot control code
        st.header("Bot Control")
        # ...

    with tab_rankings:
        # Create instance of WeightsManagementUI and render it
        weights_ui = WeightsManagementUI(DB_CONFIG)
        weights_ui.render()

    with tab_settings:
        # Your existing settings code
        st.header("Settings")
        # ...

if __name__ == "__main__":
    main()
```

5. Set Up the AI Weight Adjuster (Optional)

If you want to use the AI-based weight adjustment, add a scheduled task:

```
# Create a scheduled task file (e.g., scheduled_tasks.py)

import asyncio
import logging
import asyncpg
from ai_weight_adjuster import AIWeightAdjuster

# Database connection parameters
DB_CONFIG = {
    'user': 'clayb',
    'password': 'musicman',
    'database': 'tick_data',
    'host': 'localhost'
}

async def main():
    # Set up logging
    logging.basicConfig(
        level=logging.INFO,
        format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
        handlers=[
            logging.FileHandler('scheduled_tasks.log'),
            logging.StreamHandler()
        ]
```

```
    )

    # Create a database pool
    pool = await asyncpg.create_pool(**DB_CONFIG)

    try:
        # Create an AI weight adjuster
        adjuster = AIWeightAdjuster(pool)

        # Run the adjuster once
        success = await adjuster.adjust_weights()
        if success:
            logging.info("Successfully adjusted weights")
        else:
            logging.error("Failed to adjust weights")

        # Run scheduled adjustment (runs in a continuous loop)
        # await adjuster.run_scheduled_adjustment(hours=24)
    finally:
        await pool.close()

if __name__ == "__main__":
    asyncio.run(main())
```

You can run this task periodically using a cron job or similar scheduler.

6. Testing & Verification

1. **Database Tables:**
    - Verify the variable_weights table has been created and populated
    - Check that bot_rankings table is created
2. **Metric Updates:**
    - Confirm metrics_updater is calculating and storing all required metrics
    - Check that bot_metrics records have all the fields needed for ranking
3. **Ranking Functionality:**
    - Test the bot_ranker to make sure it correctly calculates weighted scores
    - Verify bot_rankings are being updated after trades
4. **UI Interface:**
    - Test the weights management UI to ensure it displays and updates weights
    - Verify the bot rankings display is showing correct information

7. Troubleshooting

Database Connection Issues:

- Check your PostgreSQL connection parameters
- Verify TimescaleDB extension is installed and enabled

Missing Metrics:

- Ensure all required metric calculation methods are implemented in MetricsCalculator
- Check that the MetricsUpdater is storing all calculated metrics

Ranking Not Working:

- Verify variable_weights table contains entries for all metrics
- Check for errors in the bot_ranker.py implementation
- Ensure transaction with trade_listener is correctly triggering ranking updates

UI Display Issues:

- Check for any errors in the Streamlit console
- Verify that asyncio is properly handling the asynchronous database operations

8. Future Enhancements

Once the basic integration is working, consider these enhancements:

1. **Automated ML-based weight optimization**
    - Implement a more sophisticated model in AIWeightAdjuster
    - Use historical performance to continuously optimize weights
2. **Real-time performance monitoring**
    - Add a real-time dashboard showing bot rankings and performance
3. **Fund allocation automation**
    - Implement automatic fund allocation based on bot rankings
    - Create risk management rules for allocation limits
4. **Backtesting with different weight configurations**
    - Add ability to test different weight configurations on historical data
    - Compare results to optimize weight settings