Macro-Aware DGRCL: Design Document for Systematic Trading

Architecture Version 1.1 (Confidence-Aware)

1. Executive Summary

Project Name: The Hydra (Macro-Aware DGRCL)

Objective: Develop a market-neutral, ranking-based trading system utilizing Graph Neural Networks (GNN) with Virtual Macro Nodes to predict relative stock performance.

Core Hypothesis: Stock movements are not isolated events but result from the propagation of shocks through a network. By explicitly modeling "Macro Nodes" (Oil, Yields, VIX) as graph entities, the model can detect systemic sector rotations.

Confidence Update: The system now incorporates uncertainty estimation to filter low-confidence signals, allowing for dynamic portfolio sizing and capital preservation during uncertain market regimes.

2. Architecture Specification

2.1 The Heterogeneous Graph Topology

Unlike standard stock GNNs which are Homogeneous (Stock-to-Stock), this architecture is **Heterogeneous** (Mixed Types).

Node Type A: Stock Nodes (N_s)

- **Entities:** Constituents of S&P 500 or Nasdaq 100.
- **Input Features (X_s):** $[Close, High, Low, Volume, RSI_{14}, MACD, Volatility_5]$.
- **Feature Dim:** $R^{N_s \times T \times F}$ (e.g., 500 stocks, 30 days history, 7 features).

Node Type B: Virtual Macro Nodes (N_m)

- **Entities:** Global market drivers.
  1. **Energy Node:** Crude Oil Futures (CL=F)
  2. **Rates Node:** 10-Year Treasury Yield (^TNX)
  3. **Currency Node:** USD Index (DX-Y.NYB)
  4. **Fear Node:** VIX Index (^VIX)

- **Input Features (X_m):** $[Close, \%Change, MA_{50}, MA_{200}]$.

2.2 The Edge Logic (The Connectivity Layer)

The graph uses three distinct edge sets (E) to facilitate message passing:

1. **Macro-to-Stock Edges ($E_{m \to s}$):**
   - **Definition:** Directed edges from Virtual Nodes to specific Stock Nodes.
   - **Initialization:**
     - *Hardcoded (Prior Knowledge):* Connect Energy Node $\to$ All Energy Sector Stocks (Weight = 1.0).
     - *Learned (Dynamic):* Connect Rates Node $\to$ All Tech Stocks (Weight = Learnable Parameter).
   - **Function:** Allows macro shocks to broadcast instantly to relevant sectors.
2. **Stock-to-Stock Edges ($E_{s \to s}$):**
   - **Definition:** Dynamic edges based on latent similarity.
   - **Logic:** Calculated daily via the Graph Learner module (detailed below).
   - **Function:** Captures ripple effects (e.g., Nvidia jumps $\to$ AMD jumps).
3. **Stock-to-Macro Feedback ($E_{s \to m}$):**
   - **Definition:** Aggregated feedback from sectors back to the virtual nodes.
   - **Function:** Allows the model to update the "state" of the macro node based on the market's reaction.

2.3 The DGRCL Model Pipeline

Module 1: Temporal Encoder (LSTM)

- Input: Raw time-series features for Stocks and Macro nodes.
- Process: h_i = LSTM(x_i) for all nodes.
- Output: A latent embedding vector h_i representing the current "state" of the node.

Module 2: Dynamic Graph Learner (Self-Attention)

- Process: Calculates an adjacency matrix A based on the similarity of embeddings.
- Formula: $A_{ij} = \frac{\exp(\text{LeakyReLU}(a^T [Wh_i || Wh_j]))}{\sum \dots}$
- *Innovation:* This learner dynamically assigns weights to the "Virtual Nodes" based on current market conditions.

Module 3: Relational GNN (Propagation)

- Process: Message passing across the Heterogeneous Graph.

- h_i' = ReLU(Sum(Neighbors) + Sum(Virtual_Macro_Inputs))
- The virtual nodes inject global context into the local stock embeddings.

Module 4: Uncertainty Estimation (The Confidence Layer)

- **Method:** Monte Carlo Dropout (Epistemic Uncertainty).
- **Process:** During inference, the model is run T times (e.g., $T = 10$) with dropout enabled.
- **Outputs per Stock:**
  - **Raw Score (μ):** Mean of the 10 runs.
  - **Confidence (σ^2):** Variance of the 10 runs.
- **Logic:** High Variance = The model is "guessing" based on weak features. Low Variance = The model detects a strong, persistent structural pattern.

Module 5: Contrastive Learning (The Guardrail)

- **Objective:** Maximize agreement between the *Dynamic Graph* representation and a *Static Prior* (Sector Map).
- **Loss Function:** InfoNCE Loss.

3. Data Engineering & Preprocessing

3.1 Data Sources (Cost: Free)

- **Primary API:** yfinance (Python wrapper for Yahoo Finance).
- **Frequency:** Daily (OHLCV).
- **Ticker Universe:** Download the current S&P 500 list from Wikipedia using pandas.read_html.

3.2 Preprocessing Rules

1. **Alignment:** Ensure Macro data is aligned with Stock data. Fill missing Macro values (e.g., holidays) with ffill() (Forward Fill).
2. **Normalization (Critical):**
   - Use **Z-Score** over a rolling window (e.g., 60 days).
   - $x'_t = \frac{x_t - \mu_{t-60:t}}{\sigma_{t-60:t}}$
3. **Target Variable:**
   - We are predicting **Forward 1-Day Return**.
   - $y_t = \ln(Close_{t+1} / Close_t)$

4. Trading Strategy & Execution (Dynamic Gating)

4.1 The "Sniper" Selection Logic

Instead of a fixed Top 10 / Bottom 10, we use a **Dynamic Basket Size**.

Signal Generation Steps:

1. **Inference:** Run the model 10 times (Monte Carlo Dropout) to get Mean Score (μ) and Variance (σ^2) for all 500 stocks.
2. **Confidence Ratio:** Calculate $C = \mu/\sigma^2$ (Reward per unit of Model Uncertainty).
3. **Filtration:**
    - **Candidate List:** Filter stocks where $\mu > 0$ (Predicted Up) and $\mu < 0$ (Predicted Down).
    - **Confidence Cutoff:** Discard any stock where $\sigma^2 > \text{Threshold}$ (e.g., top 20% variance).
4. **Dynamic Basket Construction:**
    - Select Top N Longs where $C > \text{MinimumConfidence}$.
    - Select Top N Shorts where $C > \text{MinimumConfidence}$.
    - *Result:* On a clear day, N might be 10. On a noisy/sideways day, N might be 1 or 0.

4.2 Capital Allocation (The Gating Function)

We do not use 100% capital every day.

- **Global Confidence Index (GCI):** The average confidence of the Top 5 candidates.
- **Allocation Rule:**
    - If $GCI < 0.5$: **Allocation = 0% (Sit in Cash).**
    - If $0.5 < GCI < 0.8$: **Allocation = 50% (Half Size).**
    - If $GCI > 0.8$: **Allocation = 100% (Full Size).**

4.3 Execution Logic

- **Entry:** Market On Close (MOC) orders for the filtered basket only.
- **Exit:** Hold for 1 day. Re-rank and re-balance the next day.
- **Net Exposure:** Maintain roughly Dollar Neutrality even with small baskets (e.g., if you have 2 High-Confidence Longs and 0 Shorts, either skip the trade or hedge with a general market ETF inverse like SH).

5. Evaluation Framework

5.1 Backtesting Setup (Walk-Forward)

Do not use a simple Train/Test split. Use **Rolling Window Validation**.

- **Train:** 3 Years (sliding).
- **Test:** Next 3 Months.
- *Slide forward 3 months and repeat.*

5.2 Key Performance Indicators (KPIs)

Metric	Target	Description
Information Coefficient (IC)	>	Correlation between predicted rank and actual rank.
Sharpe Ratio	>	Risk-adjusted return.
Participation Rate	40-60%	Percentage of days the model actually trades. (Lower is better if it avoids losses).
Win Rate (High Conf)	>	Accuracy on "High Confidence" signals only.

6. Implementation Roadmap

1. **Data Ingestion Script:** Write a script to fetch S&P 500 + Macro tickers and save to .pt (PyTorch) files.
2. **Graph Construction:** Build the static edge list (Sector map) and the virtual node connections.
3. **Model Coding:** Implement the DGRCL class in PyTorch Geometric (PyG).
    - *Update:* Ensure dropout is active during prediction for MC Dropout.
4. **Loss Function:** Implement a Pairwise Ranking Loss.
5. **Backtest Loop:** Run the simulation from 2020–2024 with the **Dynamic Gating** logic enabled.

7. Estimated Costs & Resources

- **Hardware:** AMD GPU (8GB VRAM) - **Sufficient** (MC Dropout adds slight inference time, but negligible for 500 nodes).
- **Data Cost: $0** (Yahoo Finance).
- **Trading Costs: Low** (Zero-commission broker).
- **Time Investment:** High (Initial coding: 20-40 hours).