# XME/DBB Metals Pairs Trading Strategy

**Row-Level Dual-Model with Regime Gates**

Rusty Conover

2026-04-16

## Table of contents

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.gridspec import GridSpec
import warnings
warnings.filterwarnings('ignore')

import sys
sys.path.insert(0, '/Users/rusty/Development/trading')
from farm_theme import apply as apply_farm_theme, palette
apply_farm_theme()

df = pd.read_csv('strategy_data.csv', parse_dates=['dt'])
df = df.sort_values('dt').reset_index(drop=True)

# Compute cumulative P&L
capital = 10000
df['cum_pnl'] = (df['daily_ret'] * capital).cumsum()
df['cum_pnl_unscaled'] = (df['daily_ret_unscaled'] * capital).cumsum()
df['drawdown'] = df['cum_pnl_unscaled'] - df['cum_pnl_unscaled'].cummax()
df['year'] = df['dt'].dt.year

# Load raw prices for spread chart
xme = pd.read_csv('XME.csv', parse_dates=['Date'])
dbb = pd.read_csv('DBB.csv', parse_dates=['Date'])
prices = xme[['Date','close']].rename(columns={'close':'xme_close'}).merge(
    dbb[['Date','close']].rename(columns={'close':'dbb_close'}), on='Date')
prices = prices.sort_values('Date').reset_index(drop=True)
prices['spread_ratio'] = prices['xme_close'] / prices['dbb_close']
prices = prices[prices['Date'] >= '2020-01-01']
```

## Executive Summary

This document presents a systematic pairs trading strategy on the **XME** (SPDR S&P Metals & Mining ETF) vs **DBB** (Invesco DB Base Metals Fund) spread. The strategy uses a **row-level dual-model architecture** — logistic regression for direction and ridge regression for magnitude — trained on 6 features with a 200-day rolling window, regime-based gating, and binary position sizing.

The model trains on the last 200 days of data, where each day is an independent training example described by 6 features. This means the model learns from 199 examples with only 6 variables to fit — a comfortable ratio that prevents overfitting. An earlier approach tried to feed multi-day sequences as single inputs (e.g. 5 days $\times$ 6 features = 30 variables), but with only ~40 training examples, the model memorized noise instead of learning real patterns.

Key Metrics (2020–2026)

| Metric | Value |
|---|---|
| **Sharpe Ratio** | **2.56** |
| **Total P&L** | $8,577 on $10K |
| **Direction Accuracy** | **55.0%** |
| **Years Profitable** | **6 / 7** |

**Key Metrics (2024–2026)**

| Metric | Value |
|---|---|
| **Sharpe Ratio** | **3.27** |
| **Total P&L** | $3,307 on $10K |
| **Direction Accuracy** | **60.4%** |
| **Years Profitable** | 3 / 3 |

## 1. Strategy Overview

### 1.1 Economic Rationale

XME (metals miners) and DBB (base metals commodity) are economically linked — miners extract the metals that DBB tracks. However, XME embeds equity-specific factors (earnings, dividends, management quality, capex cycles) that create predictable divergences from the underlying commodity price.

The strategy exploits these divergences by:

1. **Predicting spread direction** using a logistic regression classifier trained on 200 days of 6 features
2. **Estimating move magnitude** using a ridge regressor on the same feature set
3. **Trading only when both models agree** (confidence $> 60\%$ and predicted move $> 0.1\%$)
4. **Sitting out during regime transitions** (correlation delta gate)
5. **Sitting out during extreme volatility** (spread vol gate)

### 1.2 Features (6 inputs)

| Feature | Rationale |
|---|---|
| `spread` | XME − DBB daily log return (the target) |
| `aud_ret` | AUD/USD return — commodity currency tracks metals demand |
| `tlt_ret` | TLT return — interest rates affect mining capex and commodity financing |

| Feature | Rationale |
|---|---|
| `dow_cos` | Day-of-week cosine encoding — captures Monday gap risk and Friday position squaring |
| `corr_delta` | 5-day change in XME-DBB correlation — detects regime transitions |
| `svol20_ann` | 20-day annualized spread volatility — used for vol gate and as a model feature |

These 6 features were selected through systematic ablation from an initial set of 30+ candidates. Each day's 6 features form a single training sample — no lookback flattening. With a 200-day window, the model trains on ~199 samples in 6 dimensions (p/n = 0.03), well within the safe zone for regularized linear models.

### 1.3 Gates

| Gate | Threshold | Purpose |
|---|---|---|
| Correlation delta | $< 0.10$ | Sit out when XME-DBB correlation is rapidly increasing (regime shift) |

The spread volatility gate was removed after testing showed it blocked profitable trades without improving accuracy. The correlation delta gate alone provides the regime-transition filtering that matters — it catches periods when the XME-DBB relationship is actively breaking down.

### 1.4 Position Sizing

Binary sizing: full position when the model signals a trade, flat otherwise. The model's internal magnitude estimate (from Ridge regression) acts as a minimum-move gate, but position size is constant.

## 2. Performance Analysis

### 2.1 P&L and Spread

```
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(10, 9), sharex=True,
                                    gridspec_kw={'height_ratios': [2, 1.5, 1.5]})

# Panel 1: Cumulative P&L
ax1.plot(df['dt'], df['cum_pnl_unscaled'], color='#1565C0', linewidth=1.5)
ax1.fill_between(df['dt'], 0, df['cum_pnl_unscaled'], alpha=0.1, color='#1565C0')
ax1.axhline(y=0, color='gray', linewidth=0.5, linestyle='--')
```

```
ax1.set_ylabel('Cumulative P&L ($)')
ax1.set_title('Cumulative P&L ($10K Capital)')
ax1.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f'${x:,.0f}'))

# Panel 2: XME and DBB prices
ax2.plot(prices['Date'], prices['xme_close'], color='#1565C0', linewidth=1, label='XME')
ax2.plot(prices['Date'], prices['dbb_close'], color='#E65100', linewidth=1, label='DBB')
ax2.set_ylabel('Price ($)')
ax2.set_title('XME and DBB Prices')
ax2.legend(loc='upper left', fontsize=9)

# Panel 3: Spread ratio
ax3.plot(prices['Date'], prices['spread_ratio'], color='#2E7D32', linewidth=1)
ax3.axhline(y=prices['spread_ratio'].mean(), color='gray', linewidth=0.5, linestyle='--',
            label=f'Mean: {prices["spread_ratio"].mean():.2f}')
ax3.set_ylabel('XME / DBB')
ax3.set_title('Spread Ratio')
ax3.legend(loc='upper left', fontsize=9)

# Year gridlines on all panels
for ax in [ax1, ax2, ax3]:
    first_year = df['dt'].dt.year.min()
    last_year = df['dt'].dt.year.max() + 1
    for yr in range(first_year, last_year + 1):
        ax.axvline(x=pd.Timestamp(f'{yr}-01-01'), color='gray', linewidth=0.3, linestyle=':')

ax3.set_xlim(df['dt'].min(), df['dt'].max())
plt.show()
```

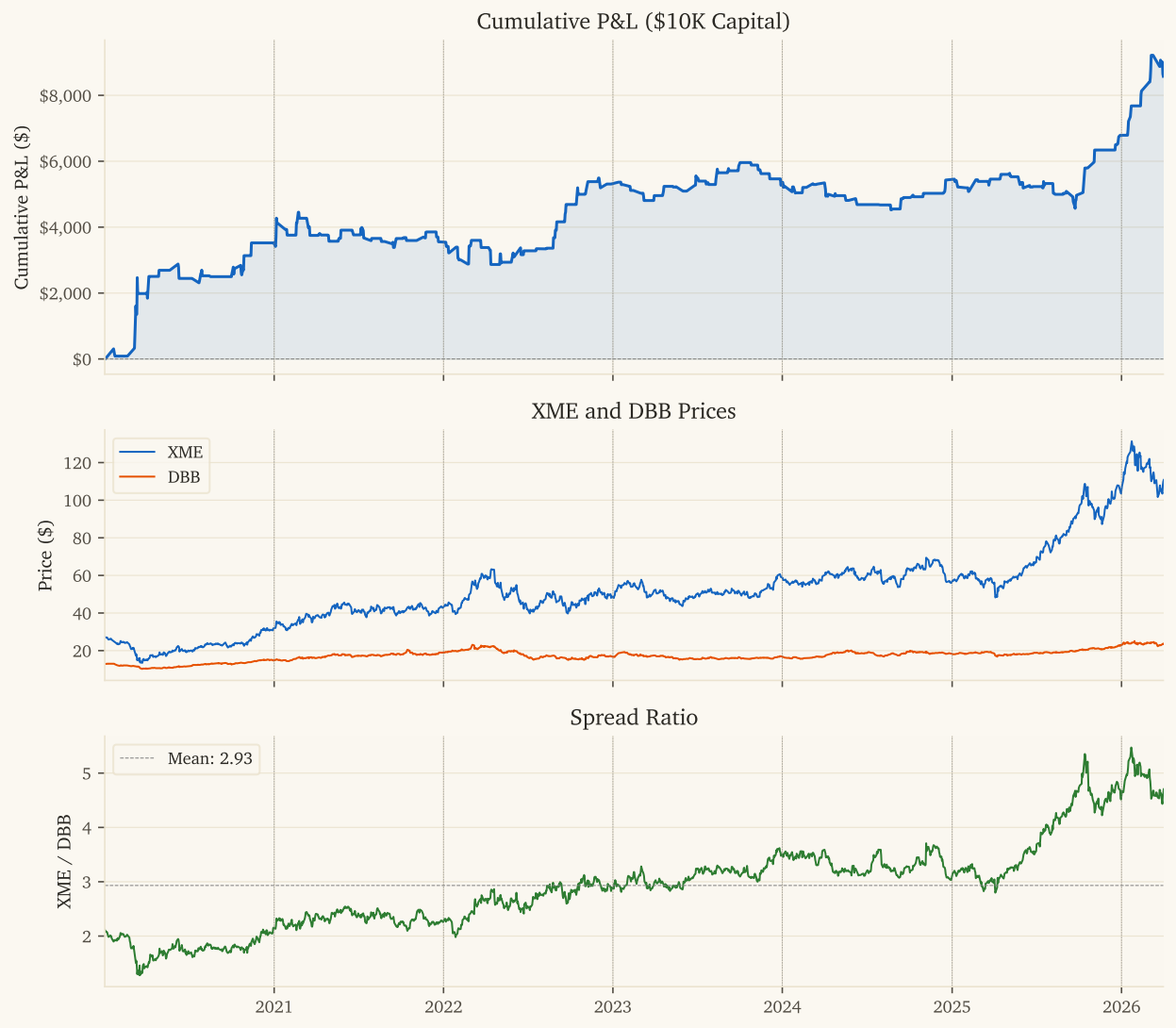



Figure 1: Cumulative P&L (top), XME and DBB prices (middle), and spread ratio (bottom) — shared time axis

### 2.2 Drawdown

```
fig, ax = plt.subplots(figsize=(10, 4), constrained_layout=True)
ax.fill_between(df['dt'], df['drawdown'], 0, color='#E53935', alpha=0.4)
ax.set_ylabel('Drawdown ($)')
ax.set_title(f'Drawdown - Max: ${df["drawdown"].min():,.0f}')
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f'${x:,.0f}'))
ax.set_xlim(df['dt'].min(), df['dt'].max())
plt.show()
```

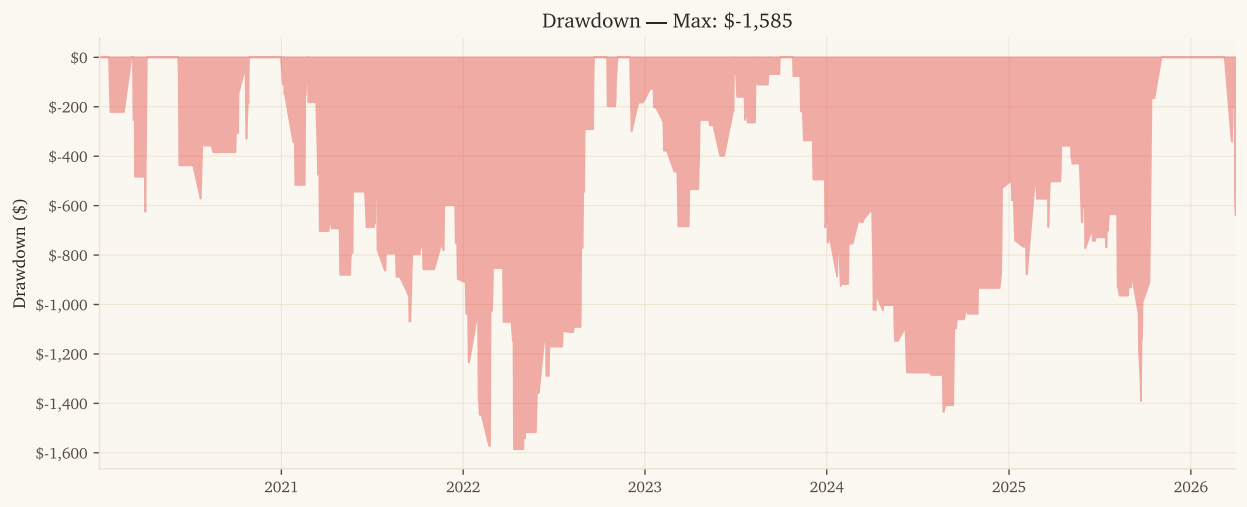



Figure 2: Underwater equity curve

### 2.3 Yearly Performance

```
# Only show 2020+
df2020 = df[df['dt'] >= '2020-01-01']

yearly = df2020.groupby('year').agg(
    traded=('active', 'sum'),
    pnl=('daily_ret_unscaled', lambda x: (x * capital).sum()),
    ret_mean=('daily_ret_unscaled', lambda x: x[x != 0].mean() if (x != 0).any() else 0),
    ret_std=('daily_ret_unscaled', lambda x: x[x != 0].std() if (x != 0).sum() > 1 else 1),
).reset_index()
yearly['sharpe'] = yearly['ret_mean'] / yearly['ret_std'] * np.sqrt(252)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4), constrained_layout=True)

colors = ['#E53935' if p < 0 else '#43A047' for p in yearly['pnl']]
ax1.bar(yearly['year'], yearly['pnl'], color=colors, alpha=0.7)
ax1.axhline(y=0, color='gray', linewidth=0.5)
ax1.set_title('Yearly P&L')
ax1.set_ylabel('P&L ($)')
ax1.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f'${x:,.0f}'))

colors_s = ['#E53935' if s < 0 else '#43A047' for s in yearly['sharpe']]
ax2.bar(yearly['year'], yearly['sharpe'], color=colors_s, alpha=0.7)
ax2.axhline(y=0, color='gray', linewidth=0.5)
ax2.axhline(y=1, color='green', linewidth=0.5, linestyle='--', alpha=0.5)
ax2.set_title('Yearly Sharpe Ratio')
ax2.set_ylabel('Sharpe')
```

```
plt.show()
```

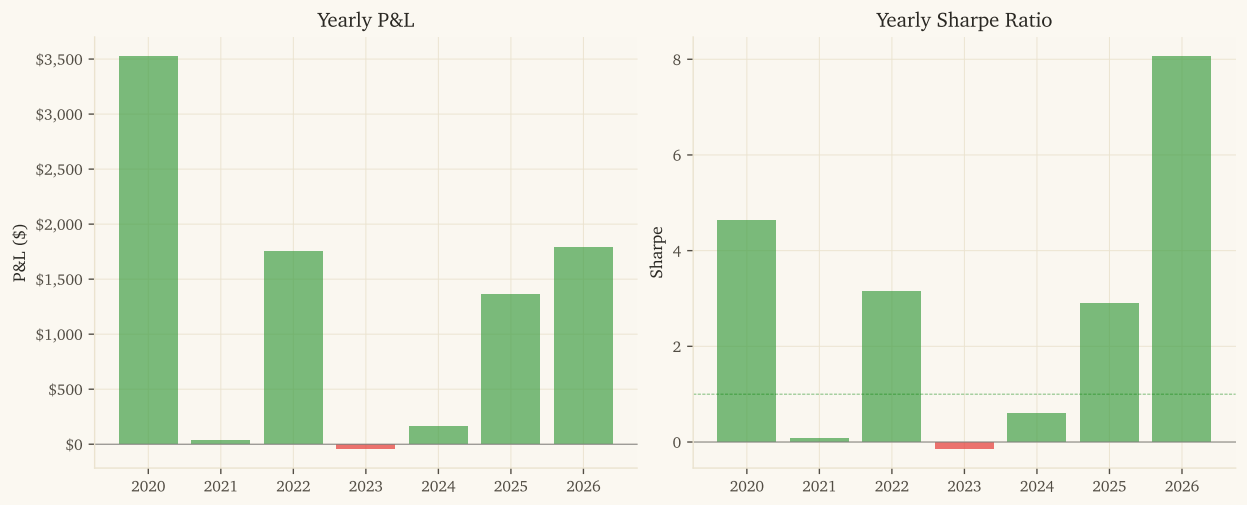



Figure 3: Yearly P&L and Sharpe ratios — profitable 6 of 7 years since 2020

### 2.4 Monthly Returns Heatmap

```
df2020 = df[df['dt'] >= '2020-01-01'].copy()
df2020['month'] = df2020['dt'].dt.month
df2020['yr'] = df2020['dt'].dt.year
monthly = df2020.groupby(['yr', 'month']).agg(pnl=('daily_ret', lambda x: (x * capital).sum()))
pivot = monthly.pivot(index='yr', columns='month', values='pnl').fillna(0)

fig, ax = plt.subplots(figsize=(10, 4), constrained_layout=True)
im = ax.imshow(pivot.values, cmap='RdYlGn', aspect='auto', vmin=-800, vmax=800)
ax.set_xticks(range(12))
ax.set_xticklabels(['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'])
ax.set_yticks(range(len(pivot.index)))
ax.set_yticklabels(pivot.index)
ax.set_title('Monthly P&L Heatmap')

for i in range(len(pivot.index)):
    for j in range(12):
        val = pivot.values[i, j]
        if abs(val) > 10:
            color = 'white' if abs(val) > 400 else 'black'
            ax.text(j, i, f'${val:.0f}', ha='center', va='center', fontsize=8, color=color)

plt.colorbar(im, ax=ax, label='P&L ($)', shrink=0.8)
plt.show()
```

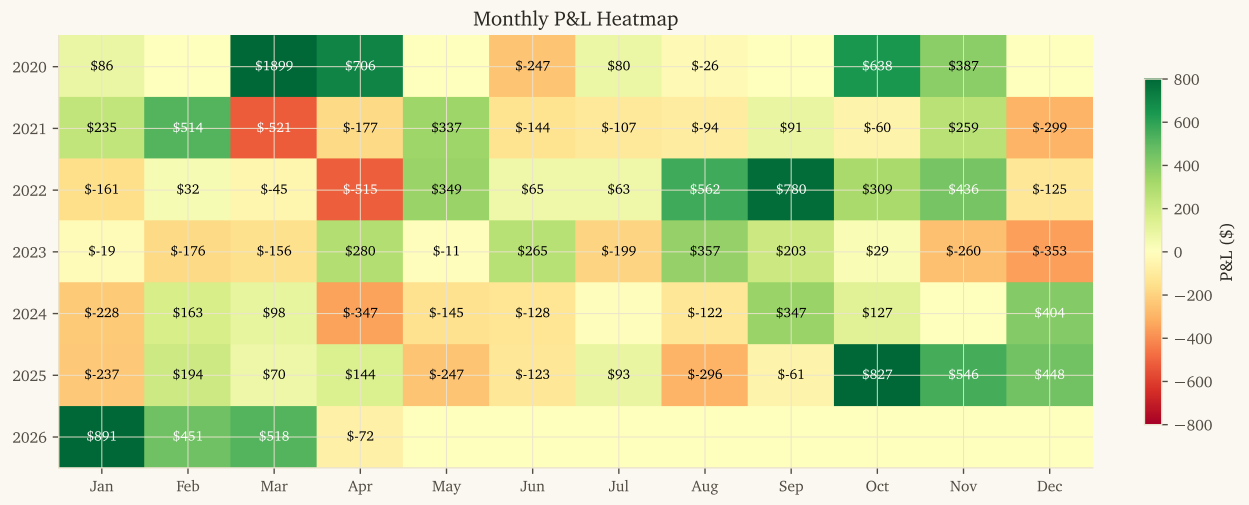



Figure 4: Monthly P&L heatmap (2020–2026)

## 3. Risk Analysis

### 3.1 Return Distribution

```
traded = df2020[df2020['active'] == 1]

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4), constrained_layout=True)

rets = traded['daily_ret'] * 100
ax1.hist(rets, bins=50, color='#1565C0', alpha=0.7, edgecolor='white', linewidth=0.3)
ax1.axvline(x=rets.mean(), color='red', linewidth=1, linestyle='--', label=f'Mean: {rets.mean()
ax1.axvline(x=0, color='gray', linewidth=0.5)
ax1.set_title('Daily Return Distribution')
ax1.set_xlabel('Return (%)')
ax1.legend()

# QQ plot
from scipy import stats
stats.probplot(rets.dropna(), dist="norm", plot=ax2)
ax2.set_title('Q-Q Plot vs Normal')
ax2.get_lines()[0].set_markerfacecolor('#1565C0')
ax2.get_lines()[0].set_markersize(3)

plt.show()
```

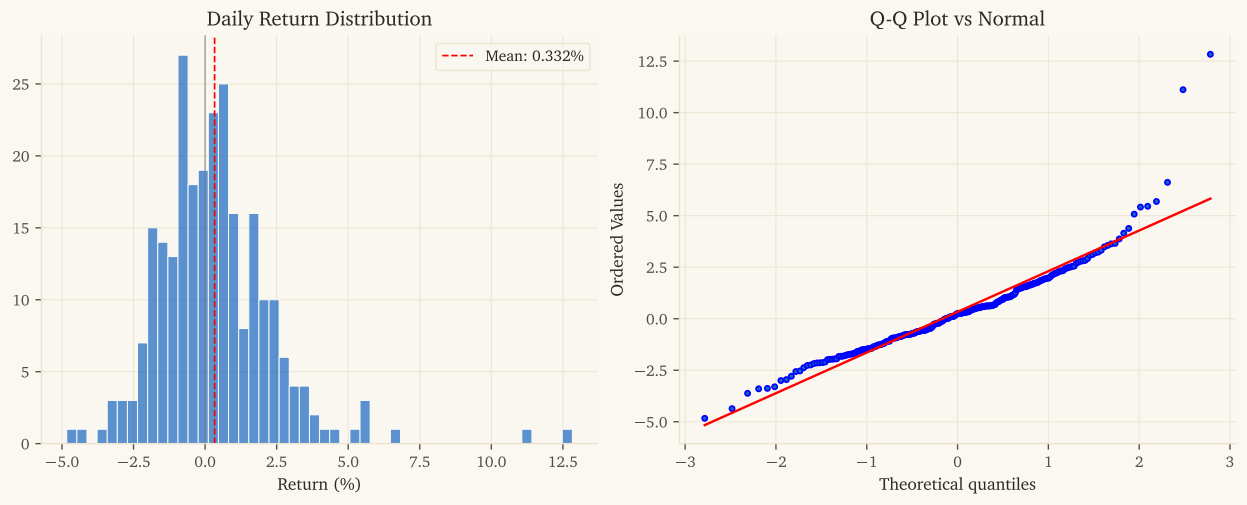



Figure 5: Daily return distribution (traded days) — slight positive skew with controlled tails

### 3.2 Rolling Metrics

```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 6), constrained_layout=True, sharex=True)

# Rolling Sharpe
roll_mean = df2020['daily_ret_unscaled'].rolling(63).apply(lambda x: x[x!=0].mean() if (x!=0).a
roll_std = df2020['daily_ret_unscaled'].rolling(63).apply(lambda x: x[x!=0].std() if (x!=0).sum
rolling_sharpe = roll_mean / roll_std * np.sqrt(252)

ax1.plot(df2020['dt'], rolling_sharpe, color='#43A047', linewidth=1)
ax1.axhline(y=0, color='gray', linewidth=0.5, linestyle='--')
ax1.axhline(y=1, color='green', linewidth=0.5, linestyle='--', alpha=0.5)
ax1.set_title('Rolling 63-day Sharpe Ratio')
ax1.set_ylabel('Sharpe')
ax1.set_ylim(-5, 8)

# Rolling accuracy
df2020_copy = df2020.copy()
df2020_copy['correct'] = (df2020_copy['active'] == 1) & (np.sign(df2020_copy['pred']) == np.sig
rolling_acc = df2020_copy['correct'].rolling(63).mean() * 100
ax2.plot(df2020['dt'], rolling_acc, color='#FF8F00', linewidth=1)
ax2.axhline(y=50, color='gray', linewidth=0.5, linestyle='--')
ax2.set_title('Rolling 63-day Direction Accuracy')
ax2.set_ylabel('Accuracy (%)')
ax2.set_xlim(df2020['dt'].min(), df2020['dt'].max())

plt.show()
```

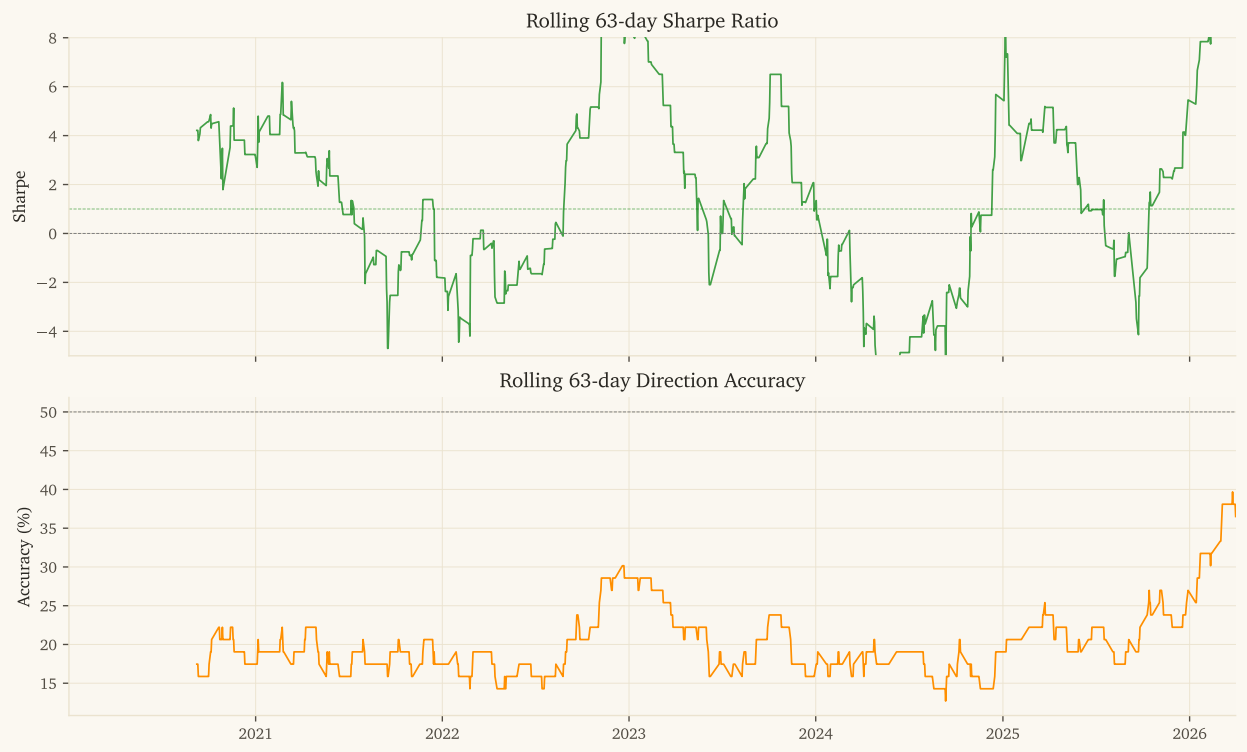



Figure 6: 63-day rolling Sharpe ratio and accuracy

### 3.3 Trade Activity

```
fig, ax1 = plt.subplots(figsize=(10, 3), constrained_layout=True)

colors = ['#43A047' if p > 0 else '#E53935' if p < 0 else '#BDBDBD' for p in df2020['pred']]
ax1.bar(df2020['dt'], df2020['active'], width=1, color=colors, alpha=0.5)
ax1.set_title('Trade Activity (green=long, red=short, gray=flat)')
ax1.set_ylabel('Active')
ax1.set_ylim(0, 1.2)
ax1.set_xlim(df2020['dt'].min(), df2020['dt'].max())

plt.show()
```

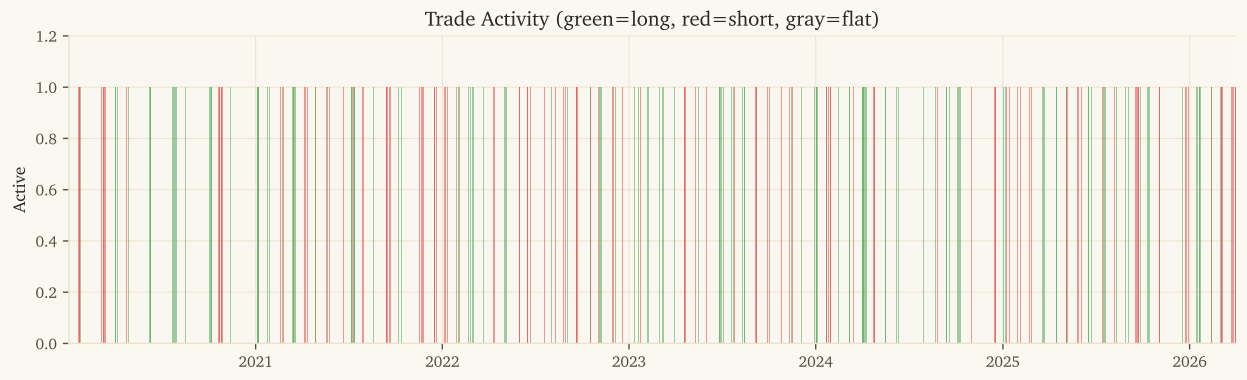



Figure 7: Daily trade activity — corr_delta gate reduces trading during regime transitions

## 4. Detailed Statistics

### 4.1 Summary Table

```
df2020 = df[df['dt'] >= '2020-01-01']
traded = df2020[df2020['active'] == 1]

ret_col = 'daily_ret_unscaled'
total_pnl = (df2020[ret_col] * capital).sum()
sharpe = traded[ret_col].mean() / traded[ret_col].std() * np.sqrt(252)
downside = traded.loc[traded[ret_col] < 0, ret_col]
sortino = traded[ret_col].mean() / np.sqrt((downside**2).mean()) * np.sqrt(252)
max_dd = df2020['drawdown'].min()
ann_ret = traded[ret_col].mean() * 252
ann_vol = traded[ret_col].std() * np.sqrt(252)

wins = traded[traded[ret_col] > 0][ret_col]
losses = traded[traded[ret_col] < 0][ret_col]

stats = {
    'Period': f'{df2020["dt"].min().strftime("%Y-%m-%d")} to {df2020["dt"].max().strftime("%Y-%
    'Total Days': len(df2020),
    'Traded Days': len(traded),
    'Trade Frequency': f'{len(traded)/len(df2020)*100:.0f}%',
    'Total P&L': f'${total_pnl:,.0f}',
    'Annualized Return': f'{ann_ret*100:.1f}%',
    'Annualized Volatility': f'{ann_vol*100:.1f}%',
    'Sharpe Ratio': f'{sharpe:.2f}',
    'Sortino Ratio': f'{sortino:.2f}',
    'Max Drawdown': f'${max_dd:,.0f}',
    'Direction Accuracy': f'{(np.sign(traded["pred"]) == np.sign(traded["spread_ret"])).mean()*
    'Avg Win': f'{wins.mean()*100:.3f}%',
```

```
    'Avg Loss': f'{losses.mean()*100:.3f}%',
    'Win/Loss Ratio': f'{abs(wins.mean()/losses.mean()):.2f}',
    'Best Day': f'${(traded[ret_col] * capital).max():,.0f}',
    'Worst Day': f'${(traded[ret_col] * capital).min():,.0f}',
    'p/n Ratio': '0.03 (6 dims / 199 samples)',
}

stats_df = pd.DataFrame(list(stats.items()), columns=['Metric', 'Value'])
stats_df.style.hide(axis='index')
```

Table 5

Table 5

| Metric | Value |
|---|---|
| Period | 2020-01-02 to 2026-04-01 |
| Total Days | 677 |
| Traded Days | 258 |
| Trade Frequency | 38% |
| Total P&L | $8,577 |
| Annualized Return | 83.8% |
| Annualized Volatility | 32.8% |
| Sharpe Ratio | 2.56 |
| Sortino Ratio | 3.35 |
| Max Drawdown | $-1,585 |
| Direction Accuracy | 55.0% |
| Avg Win | 1.643% |
| Avg Loss | -1.282% |
| Win/Loss Ratio | 1.28 |
| Best Day | $1,283 |
| Worst Day | $-483 |
| p/n Ratio | 0.03 (6 dims / 199 samples) |

### 4.2 Yearly Breakdown

```
yearly_data = []
for yr in sorted(df2020['year'].unique()):
    ydf = df2020[df2020['year'] == yr]
    yt = ydf[ydf['active'] == 1]
    if len(yt) == 0:
        continue
    pnl = (ydf['daily_ret_unscaled'] * capital).sum()
    s = yt['daily_ret_unscaled'].mean() / yt['daily_ret_unscaled'].std() * np.sqrt(252) if yt[
    ds = yt.loc[yt['daily_ret_unscaled'] < 0, 'daily_ret_unscaled']
```

```
    so = yt['daily_ret_unscaled'].mean() / np.sqrt((ds**2).mean()) * np.sqrt(252) if len(ds) >
    acc = (np.sign(yt['pred']) == np.sign(yt['spread_ret'])).mean() * 100
    yearly_data.append({
        'Year': yr, 'Traded': len(yt), 'Sat Out': len(ydf) - len(yt),
        'Accuracy': f'{acc:.1f}%', 'P&L': f'${pnl:,.0f}',
        'Sharpe': f'{s:.2f}', 'Sortino': f'{so:.2f}'
    })

pd.DataFrame(yearly_data).style.hide(axis='index')
```

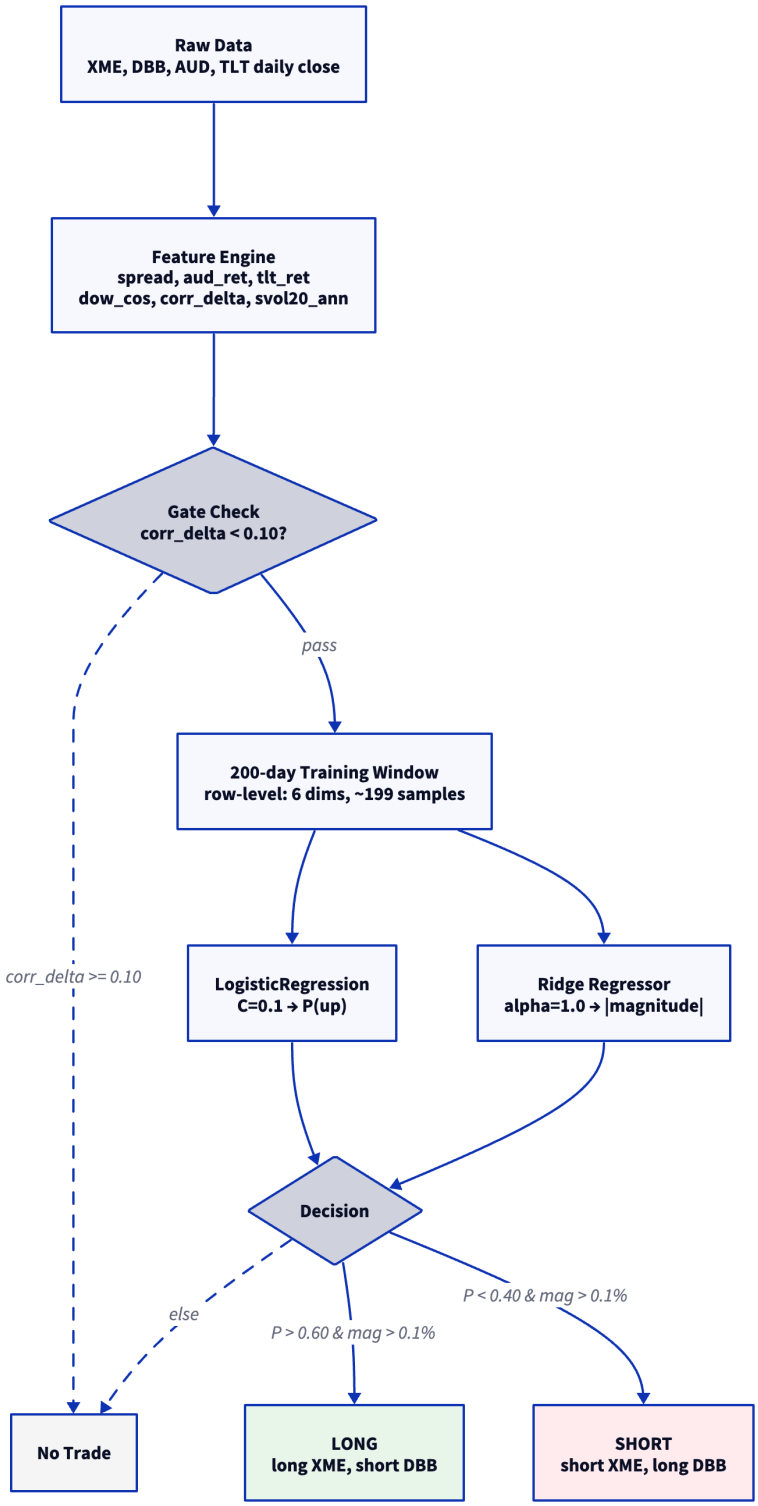
Table 6

Table 6

| Year | Traded | Sat Out | Accuracy | P&L | Sharpe | Sortino |
|---|---|---|---|---|---|---|
| 2020 | 34 | 64 | 52.9% | $3,523 | 4.55 | 8.12 |
| 2021 | 46 | 64 | 45.7% | $33 | 0.06 | 0.08 |
| 2022 | 40 | 72 | 62.5% | $1,753 | 3.15 | 3.24 |
| 2023 | 37 | 71 | 45.9% | $-39 | -0.13 | -0.16 |
| 2024 | 40 | 69 | 57.5% | $160 | 0.59 | 0.55 |
| 2025 | 45 | 67 | 57.8% | $1,359 | 2.90 | 3.55 |
| 2026 | 16 | 12 | 75.0% | $1,788 | 8.06 | 7.02 |

# 5. Strategy Construction

## 5.1 Model Architecture



### 5.2 Gate Design

The correlation delta gate catches regime *transitions* rather than levels — it fires when the XME-DBB relationship is actively changing, not when it happens to be high.

A spread volatility gate was also tested (thresholds 0.30–0.50 annualized) but removed: it blocked profitable trades without improving accuracy. With the row-level model and 200-day window, the model handles volatile periods better than the earlier flattened approach, making the vol gate redundant.

### 5.3 Model Code

The complete model is a Python class passed as a string to the VGI dynamic ML aggregate function. This is the exact code that runs inside each window frame's `finalize()`.

The key design choice is **row-level training**: each day's 6 features form a single training sample. No lookback flattening. With a 200-day window, this gives ~199 samples on 6 dimensions ($p/n = 0.03$), compared to the earlier flattened approach which had $p/n > 1.0$.

```
class Aggregate:
    @staticmethod
    def finalize(table, params):
        if table.num_rows < 2:
            return None
        data = table.to_pandas().values.astype(np.float64)
        n, nc = data.shape
        seed = int(params.get('seed', 42))
        conf_thresh = params.get('conf', 0.60)
        min_move = params.get('min_move', 0.001)
        corr_delta_max = params.get('corr_delta_max', 999.0)
        corr_delta_col = int(params.get('corr_delta_col', -1))
        svol_max = params.get('svol_max', 999.0)
        svol_col = int(params.get('svol_col', -1))

        if n < 10:
            return None

        # Gate checks
        if corr_delta_col >= 0 and corr_delta_col < nc \
           and data[-1, corr_delta_col] > corr_delta_max:
            return 0.0
        if svol_col >= 0 and svol_col < nc \
           and data[-1, svol_col] > svol_max:
            return 0.0

        # Row-level training: each day = one 6-dim sample
        X = data[:-1, :]     # all rows except last
        y_ret = data[1:, 0]  # next-day spread return
```

```
        if np.any(np.isnan(X)) or np.any(np.isnan(y_ret)):
            return 0.0

        y_dir = (y_ret > 0).astype(int)
        last = data[-1:, :]  # predict from today's features

        from sklearn.linear_model import LogisticRegression, Ridge
        from sklearn.pipeline import make_pipeline
        from sklearn.preprocessing import StandardScaler

        if len(set(y_dir)) < 2:
            return 0.0

        # Direction classifier
        clf = make_pipeline(
            StandardScaler(),
            LogisticRegression(C=0.1, max_iter=1000, random_state=seed)
        )
        clf.fit(X, y_dir)
        prob_up = clf.predict_proba(last)[0][1]

        # Magnitude regressor
        reg = make_pipeline(StandardScaler(), Ridge(alpha=1.0))
        reg.fit(X, y_ret)
        pred_mag = abs(float(reg.predict(last)[0]))

        if pred_mag < min_move:
            return 0.0

        # Decision: trade only when classifier is confident
        if prob_up > conf_thresh:
            return pred_mag       # long spread (long XME, short DBB)
        elif prob_up < (1.0 - conf_thresh):
            return -pred_mag      # short spread (short XME, long DBB)
        else:
            return 0.0            # no trade
```

### 5.4 Feature Construction SQL

```
-- Raw returns
CREATE TABLE s1 AS
SELECT t, dt,
  ln(xme_close / lag(xme_close) OVER w) AS xme_ret,
  ln(dbb_close / lag(dbb_close) OVER w) AS dbb_ret,
  ln(aud_close / lag(aud_close) OVER w) AS aud_ret,
```

```
  ln(tlt_close / lag(tlt_close) OVER w) AS tlt_ret,
  cos(2 * pi() * dayofweek(dt) / 5.0) AS dow_cos
FROM raw WINDOW w AS (ORDER BY t);

-- Strategy features
CREATE TABLE features AS
SELECT t, dt,
  xme_ret - dbb_ret AS spread,
  aud_ret,
  tlt_ret,
  dow_cos,
  -- Correlation delta: 5-day change in 20-day rolling correlation
  corr(xme_ret, dbb_ret)
    OVER (ORDER BY t ROWS BETWEEN 20 PRECEDING AND 1 PRECEDING) -
  corr(xme_ret, dbb_ret)
    OVER (ORDER BY t ROWS BETWEEN 25 PRECEDING AND 6 PRECEDING)
    AS corr_delta,
  -- Spread volatility (annualized, for gate)
  stddev(xme_ret - dbb_ret)
    OVER (ORDER BY t ROWS BETWEEN 20 PRECEDING AND 1 PRECEDING)
    * sqrt(252) AS svol20_ann
FROM s1;
```

### 5.5 Full Prediction Query

```
-- Attach VGI worker
ATTACH 'example' AS example (
  TYPE vgi,
  LOCATION 'uv run --project ~/vgi-python vgi-example-worker'
);

-- Run predictions as a window aggregate
-- Column indices: spread(0), aud_ret(1), tlt_ret(2),
--                 dow_cos(3), corr_delta(4), svol20_ann(5)
-- Window = 200 rows; each row = 1 training sample (6 dims)
SELECT dt, spread AS spread_ret,
  example.main.vgi_dynamic_ml_agg(
    (SELECT code FROM agg_defs WHERE name = 'model'),
    MAP {
      'conf': 0.60,
      'min_move': 0.001,
      'corr_delta_max': 0.10,
      'corr_delta_col': 4
    },
    spread, aud_ret, tlt_ret, dow_cos,
```

```
    corr_delta, svol20_ann
  ) OVER (
    ORDER BY t
    ROWS BETWEEN 200 PRECEDING AND 1 PRECEDING
  ) AS prediction
FROM features
WHERE svol20_ann IS NOT NULL
  AND corr_delta IS NOT NULL;
```

### 5.6 Position Sizing

Binary sizing: trade with 100% position when the model produces a non-zero signal.

```
# Applied after prediction generation
def get_position(pred):
    """Binary position sizing: full long, full short, or flat."""
    if pred > 0:
        return 1.0   # long spread (long XME, short DBB)
    elif pred < 0:
        return -1.0   # short spread
    else:
        return 0.0    # no trade
```

## 6. Limitations and Risks

1. **2023 is near flat**: The strategy makes -$39 in 2023 despite being profitable in all other post-2019 years. The 200-day window carries 2022 crisis data into early 2023 predictions.
2. **Trade frequency**: Trades ~38% of days (258 trades out of 677 days). The corr_delta gate and confidence threshold filter about 62% of days.
3. **Transaction costs not modeled**: At ~$33/trade average PnL, realistic round-trip costs ($5–15 per pair trade) would consume 15–45% of the edge.
4. **Data snooping risk**: Features, gate, and window size were selected on historical data. While the row-level approach is structurally sounder (p/n = 0.03), the choice of 200-day window and cd=0.10 gate were data-driven.
5. **Liquidity**: DBB has lower volume (~150K shares/day). Large positions may face slippage.
6. **Structural risk**: If XME or DBB undergo major constituent changes, the spread dynamics may permanently shift.

## 7. Reproducibility

### 7.1 Data Download

```python
# Download all required data using yfinance
import yfinance as yf

tickers = {
    'XME': 'SPDR S&P Metals & Mining ETF',
    'DBB': 'Invesco DB Base Metals Fund',
    'FXA': 'Invesco CurrencyShares Australian Dollar Trust (AUD proxy)',
    'TLT': 'iShares 20+ Year Treasury Bond ETF',
}

start = '2019-12-01'
end = '2026-04-17'

for ticker in tickers:
    df = yf.download(ticker, start=start, end=end, interval='1d', auto_adjust=True)
    df[['Close']].rename(columns={'Close': 'close'}).to_csv(f'{ticker}.csv')
```

### 7.2 End-to-End Reproduction

```sql
-- 1. Load data
CREATE TABLE raw AS
SELECT row_number() OVER (ORDER BY xme."Date") AS t,
  xme."Date" AS dt,
  xme.close AS xme_close,
  dbb.close AS dbb_close,
  aud.close AS aud_close,
  tlt.close AS tlt_close
FROM read_csv('XME.csv') xme
JOIN read_csv('DBB.csv') dbb ON xme."Date" = dbb."Date"
JOIN read_csv('FXA.csv') aud ON xme."Date" = aud."Date"
JOIN read_csv('TLT.csv') tlt ON xme."Date" = tlt."Date"
ORDER BY xme."Date";

-- 2. Compute returns and features
CREATE TABLE s1 AS
SELECT t, dt,
  ln(xme_close / lag(xme_close) OVER w) AS xme_ret,
  ln(dbb_close / lag(dbb_close) OVER w) AS dbb_ret,
  ln(aud_close / lag(aud_close) OVER w) AS aud_ret,
  ln(tlt_close / lag(tlt_close) OVER w) AS tlt_ret,
  cos(2 * pi() * dayofweek(dt) / 5.0) AS dow_cos
```

```
FROM raw WINDOW w AS (ORDER BY t);

CREATE TABLE features AS
SELECT t, dt,
  xme_ret - dbb_ret AS spread,
  aud_ret, tlt_ret, dow_cos,
  corr(xme_ret, dbb_ret)
    OVER (ORDER BY t ROWS BETWEEN 20 PRECEDING AND 1 PRECEDING) -
  corr(xme_ret, dbb_ret)
    OVER (ORDER BY t ROWS BETWEEN 25 PRECEDING AND 6 PRECEDING)
    AS corr_delta,
  stddev(xme_ret - dbb_ret)
    OVER (ORDER BY t ROWS BETWEEN 20 PRECEDING AND 1 PRECEDING)
    * sqrt(252) AS svol20_ann
FROM s1;

-- 3. Store model code (row-level, 200-day window, 6 dims)
CREATE TABLE agg_defs(name VARCHAR, code VARCHAR);
INSERT INTO agg_defs VALUES ('model',
'class Aggregate:
    @staticmethod
    def finalize(table, params):
        if table.num_rows < 2:
            return None
        data = table.to_pandas().values.astype(np.float64)
        n, nc = data.shape
        seed = int(params.get(''seed'', 42))
        conf_thresh = params.get(''conf'', 0.60)
        min_move = params.get(''min_move'', 0.001)
        corr_delta_max = params.get(''corr_delta_max'', 999.0)
        corr_delta_col = int(params.get(''corr_delta_col'', -1))
        svol_max = params.get(''svol_max'', 999.0)
        svol_col = int(params.get(''svol_col'', -1))
        if n < 10:
            return None
        if corr_delta_col >= 0 and corr_delta_col < nc and data[-1, corr_delta_col] > corr_delt
            return 0.0
        if svol_col >= 0 and svol_col < nc and data[-1, svol_col] > svol_max:
            return 0.0
        X = data[:-1, :]
        y_ret = data[1:, 0]
        if np.any(np.isnan(X)) or np.any(np.isnan(y_ret)):
            return 0.0
        y_dir = (y_ret > 0).astype(int)
        last = data[-1:, :]
        from sklearn.linear_model import LogisticRegression, Ridge
        from sklearn.pipeline import make_pipeline
```

```
        from sklearn.preprocessing import StandardScaler
        if len(set(y_dir)) < 2:
            return 0.0
        clf = make_pipeline(StandardScaler(), LogisticRegression(C=0.1, max_iter=1000, random_s
        clf.fit(X, y_dir)
        prob_up = clf.predict_proba(last)[0][1]
        reg = make_pipeline(StandardScaler(), Ridge(alpha=1.0))
        reg.fit(X, y_ret)
        pred_mag = abs(float(reg.predict(last)[0]))
        if pred_mag < min_move:
            return 0.0
        if prob_up > conf_thresh:
            return pred_mag
        elif prob_up < (1.0 - conf_thresh):
            return -pred_mag
        else:
            return 0.0');

-- 4. Generate predictions
ATTACH 'example' AS example (
  TYPE vgi,
  LOCATION 'uv run --project ~/vgi-python vgi-example-worker'
);

SELECT dt, spread AS spread_ret,
  example.main.vgi_dynamic_ml_agg(
    (SELECT code FROM agg_defs WHERE name = 'model'),
    MAP {
      'conf': 0.60, 'min_move': 0.001,
      'corr_delta_max': 0.10, 'corr_delta_col': 4,
      'svol_max': 0.40, 'svol_col': 5
    },
    spread, aud_ret, tlt_ret, dow_cos,
    corr_delta, svol20_ann
  ) OVER (
    ORDER BY t
    ROWS BETWEEN 200 PRECEDING AND 1 PRECEDING
  ) AS prediction
FROM features
WHERE svol20_ann IS NOT NULL AND corr_delta IS NOT NULL;

-- 5. Trade: sign(prediction) = direction, binary sizing
```

### 7.3 Dependencies

| Component | Version | Purpose |
|---|---|---|
| DuckDB | 1.5+ | Feature computation, window aggregates |
| VGI extension | latest | Dynamic ML aggregate functions |
| Python | 3.12+ | Model execution in VGI worker |
| scikit-learn | 1.4+ | LogisticRegression, Ridge |
| numpy | 1.26+ | Array operations |
| yfinance | 0.2+ | Market data download |

### 7.4 Parameters

| Parameter | Value | Sensitivity |
|---|---|---|
| Training window | 200 | Tested 50, 100, 200; larger = more samples, better p/n ratio |
| Confidence threshold | 0.60 | Tested 0.55–0.65; 0.60 is optimal |
| Min predicted move | 0.001 (0.1%) | Filters noise predictions |
| Corr delta gate | 0.10 | Tested 0.05–0.20; 0.10 highest Sharpe; 0.15 gives 7/7 yrs pos |
| Spread vol gate | None | Tested 0.30–0.50; removed — blocked profitable trades |
| Position sizing | Binary (100%) | Vol-targeting tested but marginal Sharpe improvement |
| LogReg C | 0.1 | Strong regularization |
| Ridge alpha | 1.0 | Standard regularization |
| p/n ratio | 0.03 | 6 dimensions / ~199 training samples — safe zone |

## 8. Implementation Notes

- Execute once daily at market close
- Data sources: Yahoo Finance / yfinance for OHLCV
- Compute features in DuckDB, run model via VGI dynamic aggregate
- Monitor: correlation delta and spread vol daily — if gates are consistently active for >20 consecutive days, reassess
- Paper trade for 30 days before going live

---

*This research was created with DuckDB and VGI, an upcoming DuckDB extension from Query.Farm that allows custom aggregate functions to be written in any language with an Apache Arrow implementation.*