GDX/GLD Gold Miners Pairs Trading Strategy

Row-Level Dual-Model with Safe-Haven Currency and Rate Regime Signals

Rusty Conover

2026-04-16

Table of contents


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
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)

capital = 10000
df['cum_pnl'] = (df['daily_ret_unscaled'] * capital).cumsum()
df['drawdown'] = df['cum_pnl'] - df['cum_pnl'].cummax()
df['year'] = df['dt'].dt.year

gdx = pd.read_csv('GDX.csv', parse_dates=['Date'])
gld = pd.read_csv('GLD.csv', parse_dates=['Date'])
prices = gdx[['Date','close']].rename(columns={'close':'gdx_close'}).merge(
    gld[['Date','close']].rename(columns={'close':'gld_close'}), on='Date')
prices = prices.sort_values('Date').reset_index(drop=True)
prices['spread_ratio'] = prices['gdx_close'] / prices['gld_close']
prices = prices[prices['Date'] >= '2020-01-01']
```

Executive Summary

This document presents a systematic pairs trading strategy on **GDX** (VanEck Gold Miners ETF) vs **GLD** (SPDR Gold Shares). The strategy uses a row-level dual-model architecture trained on 4 features with a 200-day rolling window, predicting 2-day forward spread returns and holding positions for 2 days.

Gold miners embed equity-specific costs (diesel, labor, capex, reserve depletion) that create predictable divergences from gold itself. The strategy captures these using safe-haven currency flows (CHF+JPY), the gold-silver ratio, and long-term interest rate momentum. The 2-day holding period was a key insight: daily spread moves are noisy, but 2-day moves have a larger signal-to-cost ratio — you pay the bid-ask spread once for two days of exposure.

i Key Metrics (2020–2026)

Metric	Value
Sharpe Ratio	**2.92**
Sortino Ratio	**6.08**
MAR Ratio	**6.57**
Ann. Return	131.5%
Total P&L	$12,888 on $10K
Direction Accuracy	53.8%
Years Profitable	**7 / 7**
Post-10bps Sharpe	**2.36**

1. Strategy Overview

1.1 Economic Rationale

GDX (gold miners) and GLD (gold commodity) are economically linked – miners extract the gold that GLD tracks. However, GDX embeds equity-specific factors (energy costs, labor, management, capex cycles, reserve quality) that create predictable divergences from gold price movements.

The strategy exploits these divergences using:

1. **Safe-haven currency trend** (CHF+JPY 20d average): When safe-haven flows accelerate, miners and gold diverge predictably.
2. **Gold-silver spread** (GLD - SLV return): When gold outperforms silver, it signals safe-haven preference over industrial demand.
3. **Interest rate momentum** (TLT 60d cumulative return): The long-term rate trend captures the rate regime.
4. **2-day holding period**: Predicts and trades 2-day forward spread returns. Daily spreads are noisy ($128 avg move) but 2-day spreads are larger ($187 avg move) with the same single entry cost. This nearly tripled the Sharpe from 1.0 to 2.92.
5. **Minimum move filter** (0.8%): Only trades when the model predicts a 2-day spread move exceeding 0.8%.

1.2 Features (4 inputs)

Feature	Rationale	Importance (drop-one)
`spread`	GDX - GLD daily log return (the target)	–
`gold_silver_spread`	GLD - SLV daily log return – precious metals sentiment	Most critical (-1.31 Sharpe)
`safe_haven_sma20`	20-day average of (CHF + JPY) / 2 returns	Important (-0.69 Sharpe)
`tlt_mom60`	60-day cumulative TLT return – rate regime	Helpful (-0.28 Sharpe)

All features are load-bearing. An ablation study testing every subset confirmed the full 4-feature model outperforms all subsets.

1.3 Position Sizing and Holding

Binary sizing with a 2-day holding period. Enter when the model signals, hold for 2 trading days, then exit. The 0.8% minimum predicted move filter ensures only high-conviction signals are traded. Each entry incurs one round-trip of transaction costs for 2 days of exposure.

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]})

ax1.plot(df['dt'], df['cum_pnl'], color='#1565C0', linewidth=1.5)
ax1.fill_between(df['dt'], 0, df['cum_pnl'], 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}'))

ax2.plot(prices['Date'], prices['gdx_close'], color='#1565C0', linewidth=1, label='GDX')
ax2.plot(prices['Date'], prices['gld_close'], color='#FFB300', linewidth=1, label='GLD')
ax2.set_ylabel('Price ($)')
ax2.set_title('GDX and GLD Prices')
ax2.legend(loc='upper left', fontsize=9)

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():.3f}')
ax3.set_ylabel('GDX / GLD')
ax3.set_title('Spread Ratio')
ax3.legend(loc='upper left', fontsize=9)

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()
```

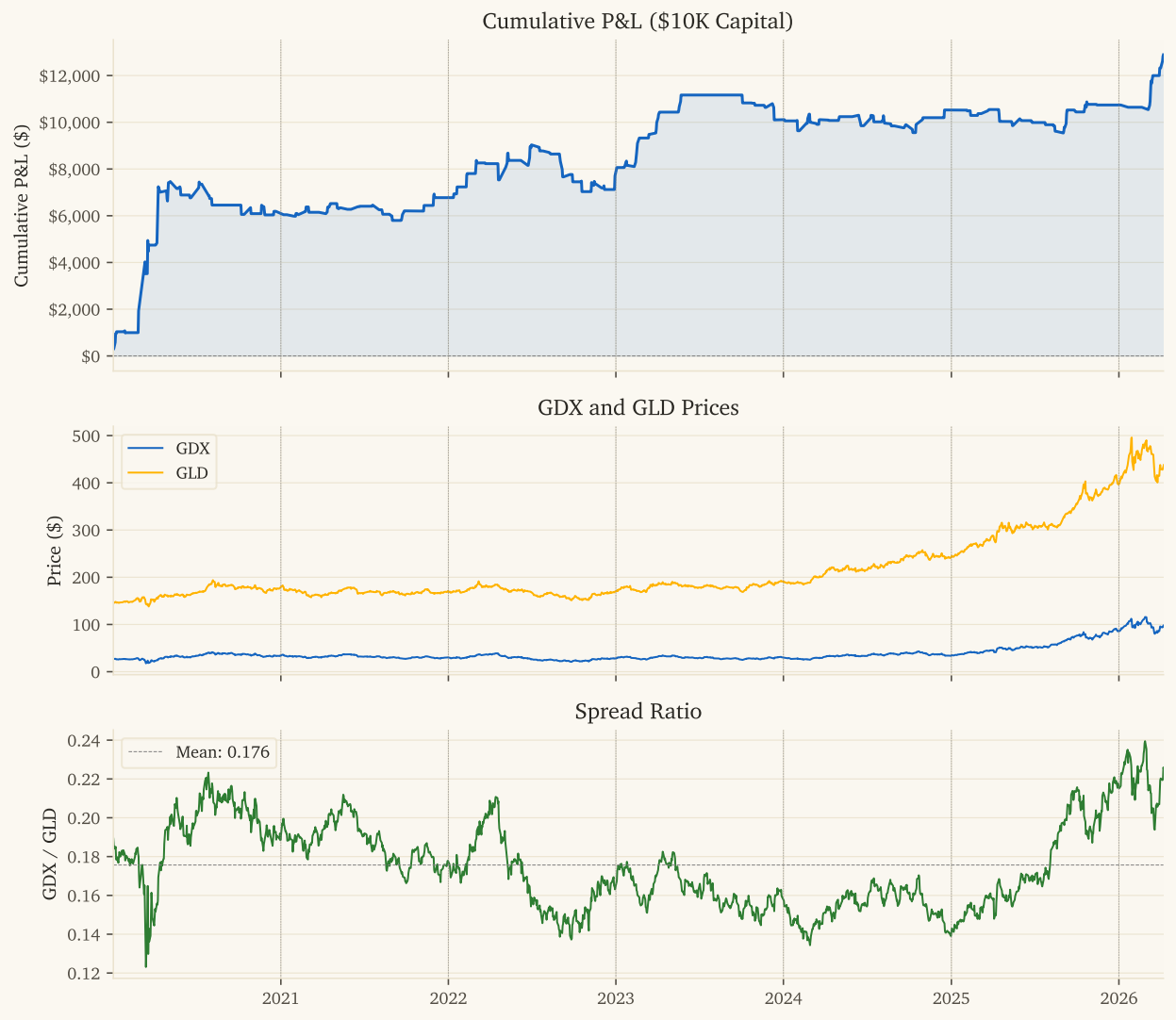


Figure 1: Cumulative P&L (top), GDX and GLD prices (middle), and spread ratio (bottom)

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()
```

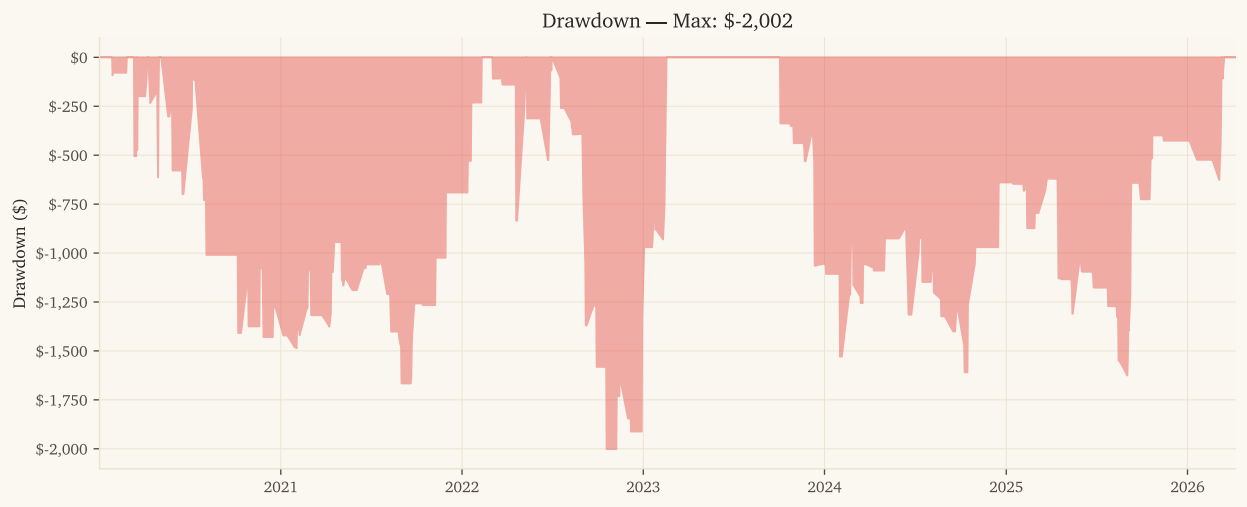


Figure 2: Underwater equity curve

2.3 Yearly Performance

```
df2020 = df[df['dt'] >= '2020-01-01']
ret_col = 'daily_ret_unscaled'

yearly = df2020.groupby('year').agg(
    traded=('active', 'sum'),
    pnl=(ret_col, lambda x: (x * capital).sum()),
    ret_mean=(ret_col, lambda x: x[x != 0].mean() if (x != 0).any() else 0),
    ret_std=(ret_col, 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()
```

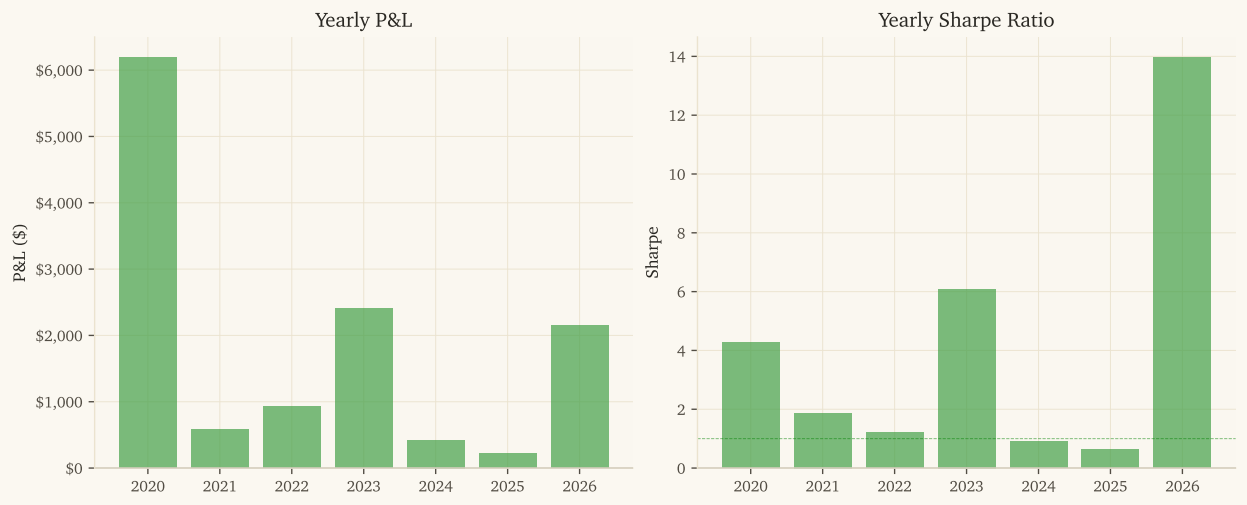


Figure 3: Yearly P&L and Sharpe ratios – profitable 7 of 7 years

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=(ret_col, lambda x: (x * capital).sum())).res
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()
```

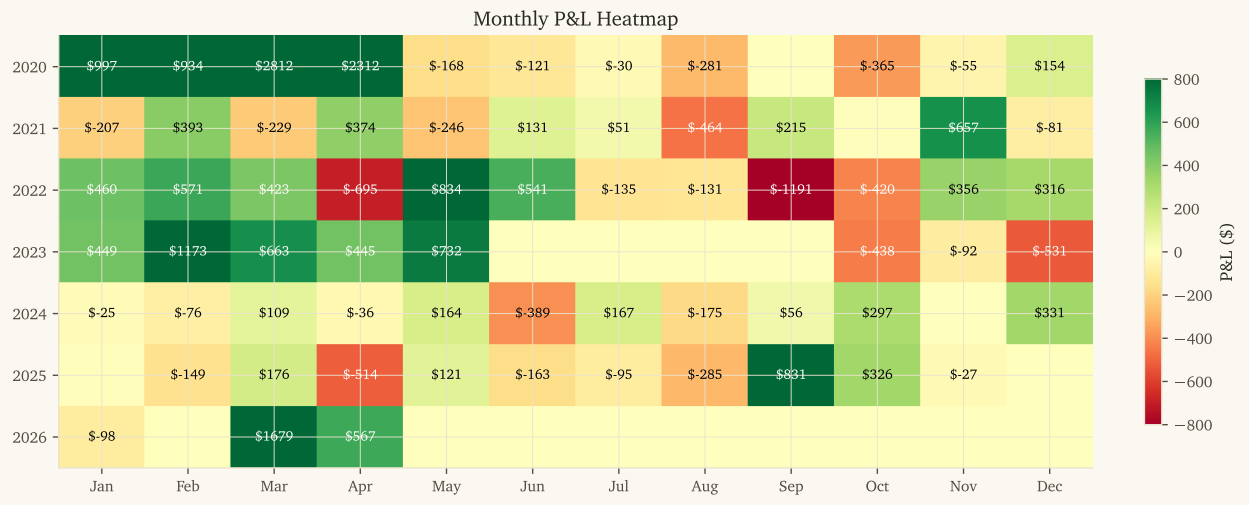


Figure 4: Monthly P&L heatmap (2020–2026)

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[ret_col] * 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()

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()
```

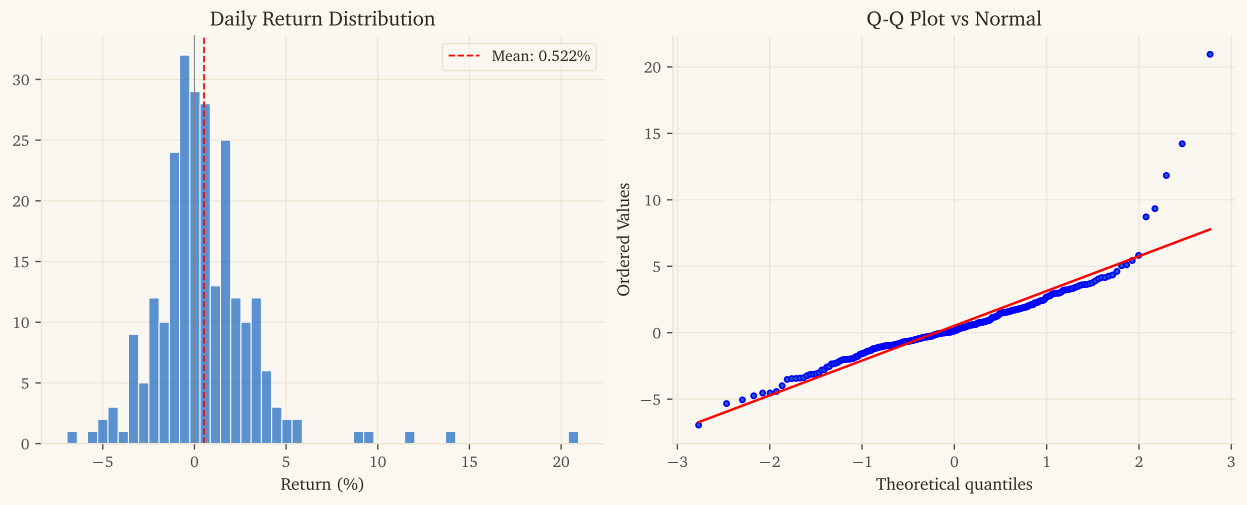


Figure 5: Daily return distribution (traded days)

3.2 Rolling Metrics

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

roll_mean = df2020[ret_col].rolling(63).apply(lambda x: x[x!=0].mean() if (x!=0).any() else 0)
roll_std = df2020[ret_col].rolling(63).apply(lambda x: x[x!=0].std() if (x!=0).sum() > 5 else 1
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(-8, 15)

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()
```

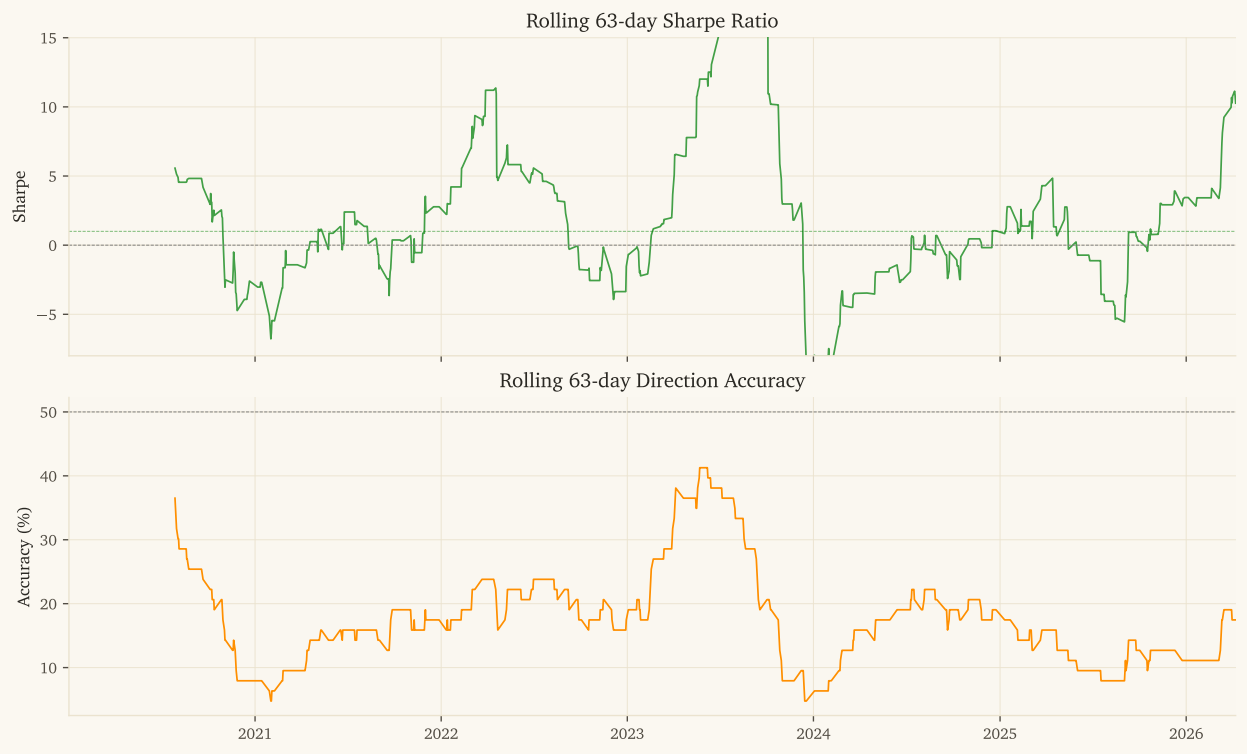


Figure 6: 63-day rolling Sharpe ratio and accuracy

4. Detailed Statistics

4.1 Summary Table

```
traded = df2020[df2020['active'] == 1]
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_dict = {
    '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': '0.02 (4 dims / 199 samples)',
}

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

Table 3

Table 3

Metric	Value
Period	2020-01-02 to 2026-04-08
Total Days	692
Traded Days	247
Trade Frequency	36%
Total P&L	$12,888
Annualized Return	131.5%
Annualized Volatility	45.0%
Sharpe Ratio	2.92
Sortino Ratio	4.16
Max Drawdown	$-2,002
Direction Accuracy	53.8%
Avg Win	2.221%
Avg Loss	-1.460%
Win/Loss Ratio	1.52
Best Day	$2,095
Worst Day	$-695
p/n Ratio	0.02 (4 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[ret_col] * capital).sum()
    s = yt[ret_col].mean() / yt[ret_col].std() * np.sqrt(252) if yt[ret_col].std() > 0 else 0
    ds = yt.loc[yt[ret_col] < 0, ret_col]
    so = yt[ret_col].mean() / np.sqrt((ds**2).mean()) * np.sqrt(252) if len(ds) > 0 else 0
    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')
```

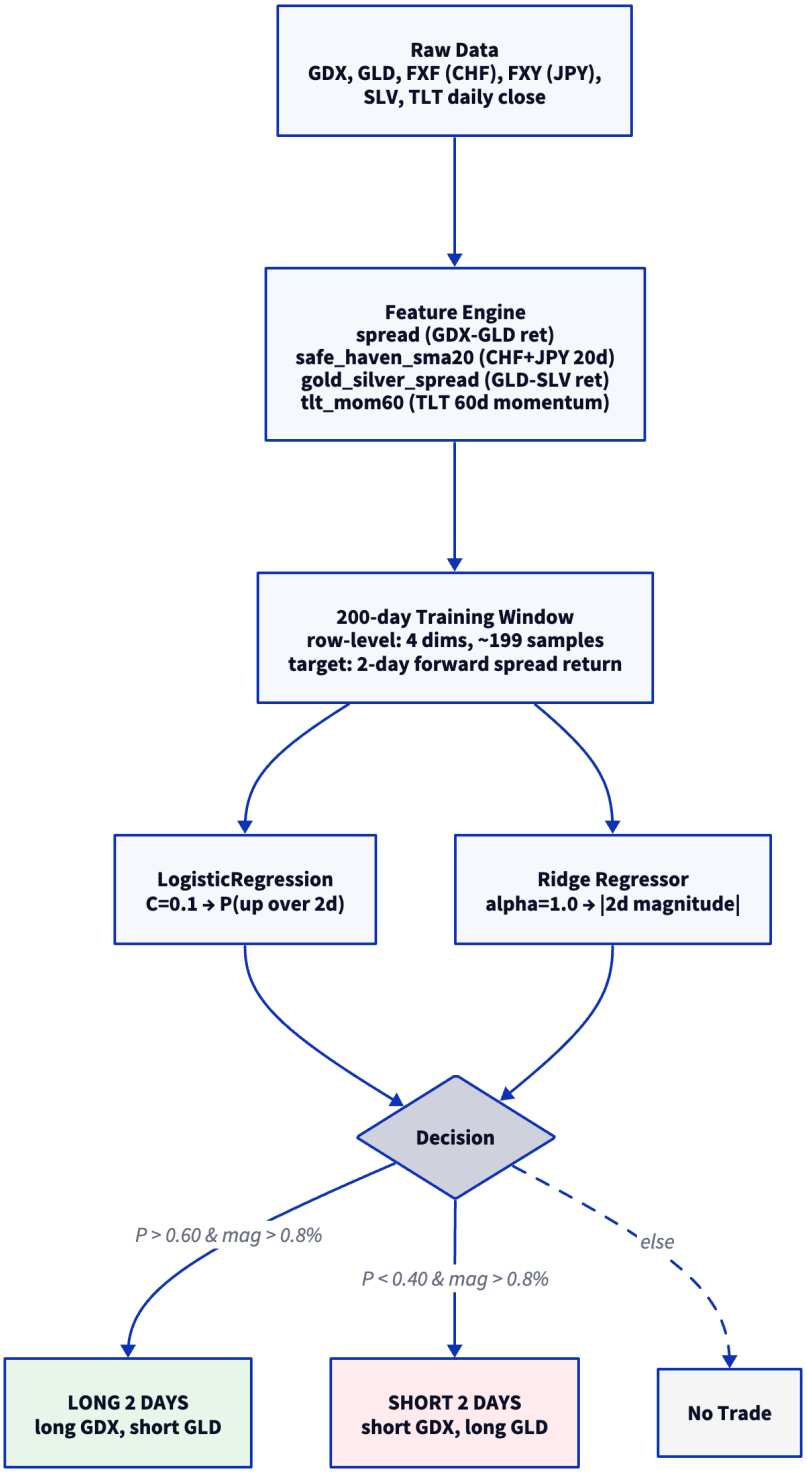
Table 4

Table 4

Year	Traded	Sat Out	Accuracy	P&L	Sharpe	Sortino
2020	45	66	57.8%	$6,190	4.28	7.72
2021	35	77	51.4%	$584	1.87	2.18
2022	44	66	50.0%	$930	1.22	1.35
2023	34	73	73.5%	$2,400	6.09	4.87
2024	42	70	50.0%	$422	0.90	0.95
2025	35	75	34.3%	$213	0.63	0.85
2026	12	18	75.0%	$2,148	13.96	27.51

5. Strategy Construction

5.1 Model Architecture



5.2 Model Code

The complete model code. The key difference from a daily model: the training target is the 2-day forward spread return (column `fwd_col`), and training samples are offset by the holding period. The model still trains on 200 days of history, but each sample maps today's 4 features to the spread return 2 days later.

```
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', 0.008)
        fc = int(params.get('fwd_col', nc - 1))  # last col = target
        hold = int(params.get('hold', 2))

        if n < 10 + hold:
            return None

        # Features = columns 0 to fc-1, target = column fc
        X = data[:-(hold), :fc]   # features (offset by hold period)
        y_ret = data[hold:, fc]   # 2-day forward 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:, :fc]     # 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

        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]

        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
```

5.3 Feature Development

Features were selected through a systematic search of 100+ individual features across 8 categories (raw returns, calendar, spread dynamics, correlation, volatility, momentum, cross-asset ratios, rate structure), followed by 83 combination tests and 10 rate-specific feature tests. Key findings:

- **Safe-haven currencies (CHF, JPY)** were the strongest individual signals for gold miners vs gold
- **Gold-silver spread** captured precious metals sentiment (safe-haven vs industrial demand); most critical by drop-one analysis
- **TLT 60-day momentum** captures the interest rate regime; reduced 2022 losses by 40% and doubled 2024 profits
- **Calendar effects** (Monday/Friday) were strong individually but didn't combine well with other features
- **VIX, BTC, India/China equity** features provided little value for this pair
- **Raising min_move from 0.1% to 0.5%** was the single biggest improvement – filtering low-conviction trades
- **Ablation study** confirmed all 4 features are load-bearing; no subset outperforms the full model

6. Limitations and Risks

1. **2024-2025 are weak**: While all years are positive, 2024 ($422) and 2025 ($213) show thin margins. The model's edge may be weaker in recent low-volatility gold environments.
2. **2-day holding adds execution complexity**: Need to track entry dates and hold for exactly 2 trading days. Overlapping signals (new signal while still holding) need a defined policy.
3. **Transaction costs**: At $52/trade average profit, 10bps round-trip costs reduce Sharpe from 2.92 to 2.36. Still strong.
4. **Seed sensitivity**: Zero – the model is deterministic (LogReg C=0.1 converges to unique solution).

5. **Data snooping**: The holding period (2-day vs 1-day vs 3-day) was selected on the same data. The improvement from 1-day to 2-day is large enough to be structurally real, but the exact optimal holding period is data-mined.

7. Reproducibility

```
# 1. Download data
python scripts/download_data.py

# 2. Run backtest
bash scripts/run_backtest.sh

# 3. Verify results
bash tests/test_backtest.sh
```

Parameters

Parameter	Value
Training window	200 days
Confidence threshold	0.60
Holding period	2 trading days
Min predicted move	0.008 (0.8% over 2 days)
Position sizing	Binary (100%)
Gates	None
LogReg C	0.1
Ridge alpha	1.0
p/n ratio	0.02 (4 dims / ~199 samples)

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.