Institutional Roll Pattern Detection in Copper Futures

A Computational Framework and Empirical Analysis

CME Copper (HG) Contracts 2008–2024

Abstract

This report presents a computational framework for detecting institutional roll patterns in futures markets through systematic calendar spread analysis. The implementation processes minute-level CME copper futures data (2008–2024) using vectorized algorithms for contract identification, spread computation, and statistical event detection. The framework aggregates 1-minute OHLCV bars into 10 intraday periods per trading day, capturing global trading sessions across US, Asia, and Europe markets. Applying a rolling z-score methodology with a 1.5 standard deviation threshold, the system detected 2,736 roll events across 44,428 intraday periods (6.16% detection rate). The median roll timing occurs 28 days before contract expiry (IQR: 16–43 days), with significant concentration during US afternoon trading sessions (14:00–15:00 CT) showing preference scores exceeding 2.0. The framework achieves computational efficiency through NumPy vectorization, processing approximately 450 million data points in under 3 minutes while maintaining memory usage below 2.5 GB.

Contents

1 Introduction

Institutional investors managing futures positions face the operational necessity of rolling contracts before expiry to maintain market exposure. This rolling activity creates detectable patterns in calendar spreads—the price differential between front-month and next-month contracts. Understanding these patterns provides insights into market microstructure and institutional trading behavior.

This analysis implements a systematic framework for detecting roll patterns in CME copper futures markets. The system processes high-frequency data at minute-level granularity, aggregates it into meaningful trading periods, and applies statistical methods to identify significant spread widening events that indicate institutional rolling activity.

The dataset encompasses 202 copper futures contracts traded on the Chicago Mercantile Exchange from January 2008 through December 2024. Each contract contains approximately 220,000 minute-level observations, totaling over 44 million data points. The analysis reveals that institutional rolling activity follows predictable patterns, with median timing at 28 days before expiry and concentrated activity during specific trading sessions.

2 Data Architecture

2.1 Raw Data Specifications

The analysis processes minute-level futures market data with the following characteristics:

Table 1: Dataset Specifications

Specification	**Value**
Exchange	CME Group (COMEX Division)
Commodity	Copper (HG)
Time Period	January 2008 – December 2024
Contracts Analyzed	202
Total Source Files	13,548 (across 32 commodities)
Data Frequency	1-minute OHLCV
File Format	Parquet (columnar storage)
Timezone	US/Central (Chicago)
Total Data Points	∼450 million
Storage Size	∼5 GB (compressed)

Each minute bar contains:

- **Timestamp**: Minute-precision datetime in US/Central
- **Open**: First trade price in the minute
- **High**: Maximum trade price in the minute
- **Low**: Minimum trade price in the minute
- **Close**: Last trade price in the minute
- **Volume**: Number of contracts traded
- **Open Interest**: Outstanding contracts (when available)

2.2 Data Organization Pipeline

The framework implements a systematic data organization pipeline that structures raw files into a hierarchical commodity-based layout:

Listing 1: Directory Structure After Organization

```
organized_data/
+-- copper/
|   +-- HGF2008.parquet
|   +-- HGG2008.parquet
|   +-- HGH2008.parquet
|   +-- ... (202 contract files)
+-- gold/
+-- silver/
+-- ... (32 commodities total)
```

The organization process:

1. Scans source directories for futures contract files
2. Extracts commodity codes using regex pattern matching
3. Maps 65+ contract symbols to 32 commodity categories
4. Creates commodity-specific folders
5. Moves files maintaining naming conventions
6. Generates `data_inventory.csv` with file metadata

2.3 Metadata Integration

Contract expiry dates are sourced from official CME calendars and stored in a normalized CSV format:

Listing 2: Contract Metadata Structure

```
root,contract,expiry_date,source,source_url
HG,HGF2009,2009-01-28,CME Copper Calendar,https://...
HG,HGG2009,2009-02-25,CME Copper Calendar,https://...
HG,HGH2009,2009-03-27,CME Copper Calendar,https://...
```

This metadata drives the front/next contract identification algorithm, ensuring accurate spread calculations across contract transitions.

3 Methodology – Technical Implementation

3.1 Intraday Period Aggregation

3.1.1 10-Period Structure

The framework aggregates minute-level data into 10 intraday periods that capture distinct trading sessions:

Table 2: Intraday Period Definitions

Period	Time (CT)	Label	Session	Duration
1	09:00–09:59	US Open	US Regular	1 hour
2	10:00–10:59	US Morning	US Regular	1 hour
3	11:00–11:59	US Late Morning	US Regular	1 hour
4	12:00–12:59	US Midday	US Regular	1 hour
5	13:00–13:59	US Early Afternoon	US Regular	1 hour
6	14:00–14:59	US Late Afternoon	US Regular	1 hour
7	15:00–15:59	US Close	US Regular	1 hour
8	16:00–20:59	Late US/After-Hours	Late US	5 hours
9	21:00–02:59	Asia Session	Asia	6 hours
10	03:00–08:59	Europe Session	Europe	6 hours

3.1.2 Aggregation Algorithm

The aggregation process employs vectorized NumPy operations for computational efficiency:

Listing 3: Vectorized Bucket Assignment

```python
def assign_bucket(hour: int) -> int:
    """Map hour (0-23) to bucket ID (1-10)"""
    if 9 <= hour <= 15:
        return hour - 8  # US regular hours
    elif 16 <= hour <= 20:
        return 8  # Late US
    elif hour >= 21 or hour <= 2:
        return 9  # Asia
    elif 3 <= hour <= 8:
        return 10  # Europe

# Vectorized application
hours = df.index.hour
bucket_ids = np.vectorize(assign_bucket)(hours)
```

Aggregation rules preserve OHLCV integrity:

- **Open**: First value in period
- **High**: Maximum value in period
- **Low**: Minimum value in period
- **Close**: Last value in period
- **Volume**: Sum of all volumes

3.1.3 Timestamp Anchoring

Each aggregated period is anchored to its start time to maintain temporal consistency:

- US regular hours: Anchored to hour start (e.g., 09:00, 10:00)
- Asia session: Anchored to 21:00 of previous day
- Europe session: Anchored to 03:00 of current day
- Late US: Anchored to 16:00 of current day

3.2 Panel Assembly and Contract Identification

3.2.1 Multi-Contract Panel Structure

The framework assembles a wide-format panel with MultiIndex columns:

Listing 4: Panel Structure

```
Columns: MultiIndex[(contract, field)]
  - (HGF2009, open)
  - (HGF2009, high)
  - (HGF2009, low)
  - (HGF2009, close)
  - (HGF2009, volume)
  - ... (repeated for 202 contracts)
  - (meta, bucket)
  - (meta, bucket_label)
  - (meta, session)
  - (meta, front_contract)
  - (meta, next_contract)

Index: DatetimeIndex (bucket timestamps)
Shape: (44428, 1015)  # 44K periods x 1015 columns
```

3.2.2 Front/Next Contract Detection

The algorithm identifies front and next contracts using vectorized operations:

Listing 5: Vectorized Contract Identification

```python
def identify_front_next(panel, expiry_map, price_field='close'):
    # Extract price matrix (periods x contracts)
    close_values = close_df.to_numpy(dtype=float)
    available = np.isfinite(close_values)

    # Compute days to expiry for all contracts
    expiry_array = pd.to_datetime(expiry_series).to_numpy()
    date_int = dates.to_numpy(dtype='datetime64[ns]')
    delta = expiry_int.reshape(1, -1) - date_int.reshape(-1, 1)

    # Mask expired and unavailable contracts
    active_mask = available & (delta >= 0)
    delta[~active_mask] = np.inf

    # Find nearest expiry (front) using argmin
    front_idx = delta.argmin(axis=1)

    # Find second nearest (next)
    delta_next = delta.copy()
    delta_next[np.arange(len(delta)), front_idx] = np.inf
    next_idx = delta_next.argmin(axis=1)

    return front_contracts, next_contracts
```

This approach processes all 44,428 periods simultaneously, achieving a 100x speedup over iterative methods.

3.3 Calendar Spread Computation

3.3.1 Spread Calculation

The calendar spread represents the price differential between next and front contracts:

$$S_t = P_{\text{next},t} - P_{\text{front},t} \tag{1}$$

Where:

- S_t = Calendar spread at time t
- $P_{\text{next},t}$ = Close price of next contract
- $P_{\text{front},t}$ = Close price of front contract

The spread change series:

$$\Delta S_t = S_t - S_{t-1} \tag{2}$$

3.3.2 Liquidity Signal

The framework computes a complementary liquidity signal based on volume ratios:

$$L_t = \begin{cases} 1 & \text{if } V_{\text{next},t} \geq \alpha \cdot V_{\text{front},t} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Where $\alpha = 0.8$ (configurable threshold).

3.4 Statistical Event Detection

3.4.1 Z-Score Methodology

The detection algorithm uses a rolling z-score of spread changes:

$$z_t = \frac{\Delta S_t - \mu_w}{\sigma_w} \tag{4}$$

Where:

- μ_w = Rolling mean over window w
- σ_w = Rolling standard deviation over window w
- $w = 20$ periods (approximately 2 trading days)

Detection criteria:

$$\text{Event}_t = \begin{cases} 1 & \text{if } z_t > 1.5 \text{ and } \Delta S_t > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

3.4.2 Cool-down Mechanism

To prevent cascade detections from single large moves, the framework enforces a cool-down period:

Listing 6: Time-Based Cool-down Implementation

```
def apply_cool_down(events, cool_down_hours=3):
    result = pd.Series(False, index=events.index)
    last_event_time = pd.Timestamp.min

    for timestamp in events[events].index:
        hours_since_last = (timestamp - last_event_time).total_seconds
            () / 3600
        if hours_since_last >= cool_down_hours:
            result[timestamp] = True
            last_event_time = timestamp

    return result
```

3.5 Data Quality Filtering

3.5.1 Contract-Level Quality Criteria

The framework evaluates each contract against multiple quality metrics:

Table 3: Data Quality Thresholds

Criterion	Threshold	Rationale
Minimum data points	500	Ensure statistical significance
Coverage percentage	≥25%	Adequate trading day representation
Maximum gap	30 days	Avoid sparse/illiquid periods
Expiry cutoff	≥2015	Focus on recent market structure

3.5.2 Quality Evaluation Algorithm

Listing 7: Contract Quality Assessment

```
def evaluate_contract(df, contract):
    data_points = len(df)
    date_range = (df.index.min(), df.index.max())
    total_days = (date_range[1] - date_range[0]).days + 1

    # Calculate trading day coverage
    expected_trading_days = total_days * 0.7
    coverage = data_points / expected_trading_days * 100

    # Identify gaps
    date_diffs = df.index.to_series().diff()
    gaps = date_diffs[date_diffs > pd.Timedelta(days=30)]

    # Apply criteria
    if data_points < 500:
        return "EXCLUDED", "Insufficient␣data"
    if coverage < 25:
        return "EXCLUDED", "Low␣coverage"
    if len(gaps) > 0:
        return "EXCLUDED", "Large␣gaps"

    return "INCLUDED", None
```

4 Implementation Framework

4.1 Package Architecture

4.1.1 Core Module Structure

The implementation consists of specialized modules with defined responsibilities:

Table 4: Core Module Functions

Module	Responsibility
`ingest.py`	Parquet file loading and contract discovery
`buckets.py`	Intraday period aggregation engine
`panel.py`	Wide-format panel assembly with metadata
`rolls.py`	Front/next identification algorithms
`events.py`	Spread detection and event summarization
`quality.py`	Data filtering and quality assessment
`analysis.py`	Pipeline orchestration and coordination
`config.py`	Settings management and validation

4.1.2 Configuration System

The framework uses YAML-based configuration with hierarchical settings:

Listing 8: Configuration Structure (settings.yaml)

```yaml
products:
  - HG  # Copper commodity code

bucket_config:
  us_regular_hours:
    start: 9
    end: 15
    granularity: "hourly"
  off_peak_sessions:
    late_us: {hours: [16,17,18,19,20], bucket: 8}
    asia: {hours: [21,22,23,0,1,2], bucket: 9}
    europe: {hours: [3,4,5,6,7,8], bucket: 10}

spread:
  method: "zscore"
  window_buckets: 20
  z_threshold: 1.5
  cool_down_hours: 3.0

data_quality:
  filter_enabled: true
  min_data_points: 500
  min_coverage_percent: 25
  max_gap_days: 30
  cutoff_year: 2015
```

4.2 Command-Line Interface

4.2.1 Entry Points

The package provides three primary command-line interfaces:

Table 5: CLI Commands

Command	Function
`futures-roll-organize`	Organize raw data files by commodity
`futures-roll-hourly`	Run intraday period analysis
`futures-roll-daily`	Run daily granularity analysis

4.2.2 Parameter Support

Each CLI supports flexible parameter overrides:

Listing 9: CLI Usage Examples

```
# Organize raw data
futures-roll-organize \
  --source /path/to/raw/files \
  --destination organized_data \
  --inventory data_inventory.csv

# Run hourly analysis with custom settings
futures-roll-hourly \
  --settings config/settings.yaml \
  --root organized_data/copper \
  --metadata metadata/contracts_metadata.csv \
  --output-dir outputs \
  --max-files 10 \
  --log-level DEBUG

# Run daily analysis with quality filtering
futures-roll-daily \
  --settings config/settings.yaml \
  --root organized_data/copper \
  --output-dir outputs
```

5 Empirical Results

5.1 Hourly Analysis – Primary Results

5.1.1 Detection Statistics

The hourly analysis processes the complete dataset at intraday period granularity:

Table 6: Hourly Analysis Summary Statistics

Metric	Value
Total intraday periods analyzed	44,428
Roll events detected	2,736
Detection rate	6.16%
Median days to expiry	28
Mean days to expiry	30.65
Standard deviation	19.32
Interquartile range (IQR)	16–43 days
Range	0–116 days

5.1.2 Intraday Distribution

Roll events exhibit distinct patterns across trading sessions:

Table 7: Roll Events by Intraday Period

Period	Session Label	Events	Share	Rate	Avg Spread	Pref Score
1	09:00 – US Open	383	14.00%	8.74%	0.0119	1.04
2	10:00 – US Morning	284	10.38%	6.49%	0.0110	0.73
3	11:00 – US Late Morning	238	8.70%	5.44%	0.0089	0.70
4	12:00 – US Midday	186	6.80%	4.24%	0.0088	0.60
5	13:00 – US Early Afternoon	132	4.82%	3.05%	0.0096	0.87
6	14:00 – US Late Afternoon	198	7.24%	4.63%	0.0137	**2.18**
7	15:00 – US Close	151	5.48%	3.53%	0.0137	**2.02**
US Regular Hours Total		1,572	57.42%			
8	Late US/After-Hours	296	10.78%	5.63%	0.0157	0.87
9	Asia Session	454	16.67%	10.31%	0.0153	0.47
10	Europe Session	414	15.13%	9.45%	0.0140	0.51
Total		**2,736**	**100%**			

Key Observations:

- US regular hours (periods 1–7) account for 57.4% of all events
- Late afternoon (14:00) and close (15:00) show preference scores > 2.0
- Asia session contributes 16.67% despite overnight timing
- Europe session shows consistent activity (15.13%)

5.1.3 Preference Score Analysis

Preference scores measure the ratio of observed event frequency to expected frequency based on volume share:

$$\text{Preference Score} = \frac{\text{Event Rate}_{\text{period}}}{\text{Volume Share}_{\text{period}}} \tag{6}$$

Scores > 1.0 indicate higher-than-expected roll activity. The 14:00–15:00 window shows scores of 2.18 and 2.02 respectively, indicating institutional preference for late-day rolling.

5.1.4 Transition Matrix

The transition matrix reveals event clustering patterns:

Table 8: Selected Transition Probabilities

From Period	To US Open	To US Late PM	To Asia	To Europe
US Open (1)	0.052	0.031	0.065	0.078
US Late Afternoon (6)	0.076	0.025	0.091	0.086
Asia Session (9)	0.084	0.035	0.099	0.095
Europe Session (10)	0.087	0.039	0.103	0.092

5.2 Daily Analysis – Validation Results

5.2.1 Quality-Filtered Dataset

Applying data quality filters significantly refines the analysis:

Table 9: Daily Analysis with Quality Filtering

Metric	**Value**
Total contracts in dataset	202
Contracts passing quality filters	109
Exclusion rate	46.0%
Trading days analyzed	5,105
Valid spread observations	3,309
Roll events detected	183
Detection rate	3.6%
Median days to expiry	14
Mean days to expiry	24.0
Range	0–352 days

5.2.2 Exclusion Reasons

Contract exclusions result from:

- Pre-2015 expiry: 48 contracts (23.8%)
- Insufficient data points: 31 contracts (15.3%)
- Low coverage: 14 contracts (6.9%)
- Large gaps: 0 contracts (0.0%)

5.2.3 Liquidity Confirmation

Volume-based liquidity signals confirm 64% of spread-based detections, validating the statistical approach with market microstructure evidence.

6 Performance Characteristics

6.1 Computational Efficiency

The framework achieves high performance through vectorization:

Table 10: Performance Metrics

Metric	**Value**
Total data points processed	∼450 million
Processing time (full pipeline)	2.8 minutes
Memory usage (peak)	2.5 GB
Memory usage (average)	1.8 GB
CPU utilization	Single-threaded
Disk I/O	Sequential reads

6.2 Scalability Analysis

Performance scales linearly with data volume:

Table 11: Scalability Characteristics

Dataset Size	Processing Time	Memory Usage
10 contracts	8 seconds	0.3 GB
50 contracts	42 seconds	0.8 GB
100 contracts	85 seconds	1.4 GB
200 contracts	168 seconds	2.5 GB

6.3 Vectorization Benefits

Comparative analysis shows 10–100x speedup from vectorization:

- Front/next identification: 100x faster than iterative approach
- Bucket assignment: 50x faster using NumPy vectorize
- Z-score calculation: 10x faster with rolling window operations
- Panel assembly: 20x faster using DataFrame operations

7 Output Specifications

7.1 Generated Files

The framework produces structured outputs in multiple formats:

Table 12: Output File Structure

Directory	File	Description
`panels/`	`hourly_panel.parquet`	Aggregated OHLCV with metadata
	`hourly_panel.csv`	CSV version for accessibility
	`daily_panel.parquet`	Daily aggregation
`roll_signals/`	`hourly_spread.csv`	Calendar spread series
	`hourly_widening.csv`	Event timestamps
	`liquidity_signal.csv`	Volume-based signals
`analysis/`	`bucket_summary.csv`	Period-level statistics
	`preference_scores.csv`	Event concentration metrics
	`transition_matrix.csv`	Period-to-period transitions
	`daily_widening_summary.csv`	Daily event details
`data_quality/`	`contract_metrics.csv`	Per-contract quality scores
	`quality_summary.json`	Aggregate statistics

7.2 Data Formats

Output formats are optimized for different use cases:

- **Parquet**: Large datasets requiring efficient storage and fast loading
- **CSV**: Human-readable summaries and integration with spreadsheet tools
- **JSON**: Metadata and configuration for programmatic access

8 Testing and Validation

8.1 Test Coverage

The framework includes comprehensive unit tests:

Table 13: Test Suite Coverage

Module	Tests	Coverage
Bucket assignment	12	100%
Aggregation logic	8	95%
Front/next identification	6	100%
Z-score calculation	5	100%
Cool-down mechanism	4	100%
Quality filtering	7	90%
Total	**42**	**96%**

8.2 Validation Checks

Critical invariants are verified:

- **Volume conservation**: Aggregated volume equals sum of minute volumes
- **Price bounds**: High $\geq$ Close $\geq$ Low across all periods
- **Bucket coverage**: All 24 hours map to exactly one bucket
- **Contract continuity**: No gaps in front/next identification
- **Timestamp monotonicity**: Strictly increasing timestamps

8.3 Edge Case Handling

The implementation handles various edge cases:

- Missing data: Forward-fill with configurable limits
- Contract transitions: Smooth handoff at expiry boundaries
- Sparse trading: Minimum period requirements for z-score
- Timezone changes: Daylight saving time adjustments
- Data anomalies: Outlier detection and filtering

9 Conclusions

This analysis successfully implements a comprehensive framework for detecting institutional roll patterns in copper futures markets. The system processes 450 million minute-level data points from 202 contracts spanning 2008–2024, identifying 2,736 statistically significant roll events.

9.1 Key Findings

1. **Timing Pattern**: Institutional rolling begins approximately one month before contract expiry (median 28 days, IQR 16–43 days)
2. **Intraday Concentration**: US afternoon sessions (14:00–15:00 CT) exhibit 2x expected roll activity based on volume shares
3. **Global Distribution**: Roll events occur across all trading sessions, with US regular hours accounting for 57.4%, Asia 16.7%, and Europe 15.1%

4. **Detection Rate**: The z-score methodology identifies events in 6.16% of intraday periods, consistent with expected institutional activity levels
5. **Quality Impact**: Data filtering reduces the contract set by 46% but improves signal quality, with daily analysis showing concentrated activity 14 days before expiry

9.2 Technical Achievements

1. **Computational Efficiency**: Vectorized algorithms process the complete dataset in under 3 minutes with peak memory usage of 2.5 GB
2. **Scalability**: Linear performance scaling enables extension to multiple commodities without architectural changes
3. **Reproducibility**: Configuration-driven pipeline with command-line interfaces ensures consistent results across environments
4. **Robustness**: Comprehensive quality filtering and validation checks maintain data integrity throughout the analysis
5. **Modularity**: Clean separation of concerns across specialized modules facilitates maintenance and enhancement

9.3 Empirical Validation

The detected patterns align with market microstructure expectations:

- Roll timing balances liquidity needs with carrying costs
- Afternoon concentration reflects institutional preference for end-of-day positioning
- Geographic distribution corresponds to global trading center activity
- Volume-based signals confirm 64% of statistical detections

The framework provides a robust foundation for systematic analysis of institutional trading patterns in futures markets, with demonstrated effectiveness on copper futures and extensibility to additional commodities.