Numinor SAM Amplifier Construct Data v1.0 — Methodology

Companion to: `SAM_Amplifier_Construct_Data_v1.0.md` (data dictionary) **Full methodology reference:** *Numinor SAM Amplifier Whitepaper v1.6* (Jan 2026)

The dictionary defines **what** is delivered. This document describes **how** to apply it.

---

1. What this Construct Data is

A pre-engineered **stock-level peer / upstream / downstream relationship graph**, refreshed daily, point-in-time correct. Each row encodes one directed edge from a focal A-share listco to a counterparty A-share listco, with a revenue-share-derived weight.

The product is **factor-model-agnostic**. The buyer applies the graph as an aggregation operator against their own base factor model — the output is the buyer's factors, *amplified* through the SAM-defined network. Numinor does not ship "pre-amplified factor files"; the buyer keeps full control of which factors to amplify.

```
buyer's base factors          ┐
SAM Amplifier 构建数据 ──────┤── aggregation operator ──→  amplified factor variants
                              ┘                                    (peer / upstream /
downstream)
```

---

2. The SAM Network

2.1 Three relationship types

For each focal stock i at trading date t, three sets of related stocks are constructed:

- **Peer set** $\mathcal{P}_{i,t}$ — stocks sharing the focal stock's primary SAM product
- **Upstream set** $\mathcal{U}_{i,t}$ — stocks whose primary SAM product *supplies* the focal stock's primary product (per the SAM supply chain v4 graph)
- **Downstream set** $\mathcal{D}_{i,t}$ — stocks whose primary SAM product *uses* the focal stock's primary product as input

The Construct Data product exposes all three as long-format edges in the schema, distinguished by `relation_type ∈ {"peer", "upstream", "downstream"}`.

2.2 Why COREIN only

ChinaScope's supply chain v4 graph classifies product-to-product relationships into ten categories: `COREIN`, `CAPEQ`, `DISTCH`, `DOWNPR`, `OPCONS`, `OPSUP`, `SVCIN`, `BIZSUP`, `INFRA`, `CAPFIN`. The v1.0 Construct Data product uses **`COREIN` only** for upstream/downstream edges, matching the canonical analysis in WP v1.6.

`COREIN` (Core Input) encodes the most direct economic linkage: a physical input that is consumed or incorporated into the downstream product (e.g., *Copper → Electrical Wire*; *Silicon Wafers → Integrated Circuits*). This directness makes price and demand shocks propagate from upstream to downstream with minimal mediation — the cleanest cross-stock predictability channel for equity-market context.

Per-category and multi-category variants are natural extensions; see WP v1.6 §7.3.

2.3 Primary product assignment (walkup)

Each focal stock is assigned to a **primary SAM product** at each reporting period. The assignment is the product at any hierarchy level for which the stock derives more than **50% of total revenue**, subject to a minimum peer count of **5** (at least 5 other listcos sharing this product as their primary).

When no SAM-level node satisfies both conditions at a finer level, the assignment walks up the 9-level hierarchy to a more aggregated level until both conditions are satisfied. This walkup ensures: (a) the stock is assigned where it has economically meaningful exposure (>50% revenue); (b) sufficient peer density exists for cross-sectional aggregation (≥5 peers); and (c) the assignment level is determined by economic concentration rather than a fixed level.

The primary product walkup is precomputed by Numinor; buyers receive only the final graph derived from it (peers, upstream, downstream edges).

---

3. Graph Construction — Edge Weights

For each `(trade_date, focal_ts)` pair in the construct data, edges are emitted with weights derived as follows. Let rev_j denote stock j's revenue from its primary SAM product at the applicable PIT-eligible source report date.

3.1 Peer edges (relation_type = "peer")

$$\text{peer\_weight}[i \to j] \;=\; \frac{\text{rev}_j}{\sum_{k \in \mathcal{P}_{i,t} \setminus \{i\}} \text{rev}_k}$$

Peer weights for a given focal i at date t **sum to ~1.0** across all peer counterparties. The focal stock is excluded from its own peer set.

3.2 Upstream edges (relation_type = "upstream")

$$\text{upstream\_weight}[i \to j] \;=\; \frac{\text{rev}_j}{\sum_{k \in \mathcal{U}_{i,t}} \text{rev}_k}$$

Where $\mathcal{U}_{i,t}$ is the set of stocks whose primary product appears in `upstream_of[primary[i]]` per the COREIN graph.

3.3 Downstream edges (relation_type = "downstream")

$$\text{downstream\_weight}[i \to j] \;=\; \frac{\text{rev}_j}{\sum_{k \in \mathcal{D}_{i,t}} \text{rev}_k}$$

Where $\mathcal{D}_{i,t}$ is the set of stocks whose primary product appears in `downstream_of[primary[i]]` per the COREIN graph.

3.4 Universe filter applied at edge emission

1. Self-edges (`focal_ts == counterparty_ts`) dropped
2. Both sides must be A-share SH/SZ/KC/CYB listco tickers (`NNNNNN.SH` or `NNNNNN.SZ`); BJ excluded
3. Zero-revenue or near-zero-revenue stocks excluded
4. Edges with weight below `1e-6` floor dropped (numerical noise suppression)
5. Edges whose PIT condition `source_rpt_date + 30 days ≤ trade_date` is violated are not emitted

---

4. Point-in-Time Discipline

4.1 The 30-day lag rule

The v1.0 Construct Data product uses a **30-day publish lag**:

```
source_publish_date + 30 calendar days ≤ trade_date
```

This buffer models realistic vendor delivery (ChinaScope T+1) + institutional buyer ingestion / recompute / deployment lag + a modest conservatism cushion. It matches the convention used in

Numinor's SAM PM Construct Data product, keeping the lag rule consistent across our catalog.

4.2 Relationship to the WP

WP v1.6 used a **60-day** lag for its empirical analysis. The v1.0 Construct Data ships with **30-day** lag, providing fresher data for production use. A buyer running the WP's methodology end-to-end on this construct data will get **structurally similar but slightly different numbers** than the WP — the signal is identical, but ~30 days more recent SAM data is included in the construct's PIT window.

If the buyer wants strict WP-replication: filter the construct data to `source_rpt_date + 60 days ≤ trade_date` in the buyer's pipeline; this is a trivial one-line filter.

4.3 Per-edge source_rpt_date

The shipped `source_rpt_date` column is the **later** of the focal's and counterparty's source report dates — i.e., the edge "exists" only from when both stocks' SAM data was publicly available. This is the conservative interpretation that prevents lookahead at the edge granularity.

4.4 Topology vintage

The SAM **product taxonomy** and the **COREIN supply chain graph** are applied as current-vintage throughout the historical sample (i.e., the latest version of the graph topology applies backward to 2016). The per-stock primary product assignments and per-stock revenues are fully point-in-time. This matches the convention of most production-grade supply-chain network datasets (Bloomberg, FactSet).

---

5. Applying the Graph to a Factor Model

5.1 The aggregation operator

For each base factor f_k and each `(focal_ts, trade_date)` pair, the buyer computes the network-routed factor variant as a revenue-weighted average over the corresponding relationship set:

$$\text{peer}_{k,i,t} = \sum_{j \in \mathcal{P}_{i,t} \setminus \{i\}} \text{peer\_weight}[i \to j] \cdot f_{k,j,t}$$

(and analogously for upstream and downstream variants). For K base factors, the operator produces $3K$ network-routed variants per stock per date: K peer-routed, K upstream-routed, K

downstream-routed. Together with the K base factors, the buyer's full feature set is $4K$ candidate signals per stock per date.

The aggregation is a simple groupby + weighted-mean operation. The reference implementation (MIT-licensed, see §7) does this in ~20 lines:

```python
import pandas as pd

def aggregate(factors_df, graph_df, relation_type, factor_cols):
    """factors_df: index=(trade_date, ts_code), cols=factor_cols
       graph_df: SAM Amplifier Construct Data filtered to relation_type
       Returns: DataFrame with same index/cols, values are network-routed."""
    edges = graph_df[graph_df["relation_type"] == relation_type].merge(
        factors_df.reset_index().rename(columns={"ts_code": "counterparty_ts"}),
        on=["trade_date", "counterparty_ts"], how="left"
    )
    # Weighted sum per (trade_date, focal_ts)
    for c in factor_cols:
        edges[f"_wf_{c}"] = edges["weight"] * edges[c].fillna(0)
    agg = edges.groupby(["trade_date", "focal_ts"]).agg(
        **{f"_wf_{c}": (f"_wf_{c}", "sum") for c in factor_cols},
        _w=("weight", "sum"),
    )
    out = pd.concat(
        [(agg[f"_wf_{c}"] / agg["_w"].where(agg["_w"] > 0)).rename(c) for c in
        factor_cols],
        axis=1,
    )
    out.index.names = ["trade_date", "focal_ts"]
    return out
```

5.2 Cross-sectional orthogonalization (recommended)

The network-routed variants are correlated with their base factors (a stock's peer momentum is positively correlated with the stock's own momentum, etc.). To isolate the *incremental* information in the network-routed variant, cross-sectionally regress the network-routed variant on the base factor at each date and use the residual as the predictive signal:

$$\widetilde{\text{peer}}_{k,i,t} \;=\; \text{peer}_{k,i,t} - \beta^{(p)}_{k,t}\, f_{k,i,t} - \alpha^{(p)}_{k,t}$$

where the regression is run cross-sectionally at each date t across all stocks. This converts each network-routed variant into the part *not explained* by the base factor.

The same orthogonalization can be applied across multiple base factors via cross-sectional OLS on the full base factor set (regress each network-routed variant on all base factors, take residual).

5.3 Combining base + network-routed signals

Several combination approaches are documented in WP v1.6:

- **Equal weighting:** sum base + 3 network residuals (simple baseline).
- **Joint Ridge regression:** train a Ridge model on all $4K$ features against forward returns, in a walk-forward or hold-out design. Used in WP §4-§6 with CV-tuned α.
- **Individual factor allocation:** rolling IC weighting per channel (peer / upstream / downstream).

The construct data is agnostic to the combination scheme. The reference implementation includes worked examples for joint Ridge and equal weighting.

---

6. Validation Summary (from WP v1.6)

Applied to Numinor's illustrative 22-factor Tushare-derived base model on the A-share universe, WP v1.6 reported the following headline numbers in **out-of-sample 2022 → April 2026**:

Metric	Value	Description
ΔICIR	**+0.149**	Joint Ridge with peer+up+down residuals vs base-alone
ΔSharpe	**+0.54 to +0.72**	Across 6 factor-count configurations (1, 3, 6, 9, 12, 22 base factors)
Stratified random subset positivity	**80 / 100**	Of 100 random 12-factor subsets of the 22-factor base, 80 showed positive ΔICIR with SAM amplification
Multi-offset positivity	**100 / 100**	Across 20 alignment offsets × 5 multi-offset robustness slices
Family drop positivity	**8 / 9**	Of 9 factor-family drops (momentum, value, etc.), 8 still showed positive ΔICIR with SAM amplification
Multi-horizon positivity	**5 / 5**	Across 5d, 10d, 20d, 30d, 60d forward returns

Buyer's actual lift depends on the buyer's factor model. Typically, cleaner Barra/MSCI/FactSet base factors deliver larger amplification effects than the Tushare-derived illustrative base used in the WP. The reference implementation lets the buyer measure their own lift in their own factor stack.

---

7. Reference Implementation

MIT-licensed Python reference at:

```
github.com/Numinor-Systems/sam-amplifier-construct-reference
```

Contents:

- `loaders.py` — `load_construct_data(date or range, api_key)` from the Numinor API
- `aggregator.py` — the peer/up/down aggregation operator (§5.1)
- `orthogonalize.py` — cross-sectional residualization (§5.2)
- `examples/` — worked examples for joint Ridge, equal weighting, and the WP's full robustness battery
- `methodology.md` — copy of this document

The reference implementation is **not** how Numinor produces the construct data internally (that's the proprietary `numinor_sam_amp` package). It's how a buyer would *consume* the construct data after purchase. The reference is intentionally minimal — focused on showing the buyer the integration points, not duplicating Numinor's internal pipeline.

---

8. Where the methodology ends and the buyer's quant work begins

The Construct Data product gives the buyer:

- The graph (peer/upstream/downstream relationships, with revenue-share weights)
- PIT-correct, refreshed daily, A-share universe-filtered
- A reference operator implementation

What the buyer must supply:

- Their own factor model (Barra, MSCI, FactSet, CIQ, or proprietary)

- Their backtest infrastructure
- Their portfolio construction logic
- Their risk model

The Construct Data is a "Lego brick" in a quant stack — a high-quality, pre-engineered network feature. It is **not** a turnkey strategy, an alpha signal in isolation, or a buy/sell recommendation. The buyer's modeling work is what converts the network feature into PnL.

---

9. Further Reading

- **Data dictionary:** `SAM_Amplifier_Construct_Data_v1.0.md`
- **Whitepaper:** *Numinor SAM Amplifier Whitepaper v1.6* (Jan 2026) — full empirical study, robustness battery, by-year stability analysis, factor-quality monotonicity tests
- **Executive summary:** *Numinor SAM Amplifier Executive Summary v1.2* — 8-page non-technical overview
- **Pitch deck:** *Numinor SAM Amplifier Pitch Deck v1.6* — slide-format technical + commercial overview
- **Reference implementation:** `github.com/Numinor-Systems/sam-amplifier-construct-reference`
