SOT Internal Extensions

Validator-Side Feature Engineering and Pattern Discovery

Date: 2025-10-25
Status: Architecture Clarification
Context: Proposals 4 & 5 are SOT enhancements done by VALIDATORS, not miners

---

Critical Clarification

Proposals 4 & 5 are NOT miner responsibilities. They are internal SOT enhancements performed by the validator/subnet to provide richer data to miners.

```
┌─────────────────────────────────────────────────────────────┐
│                    VALIDATOR / SOT OWNER                    │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  Base SOT Generation:                                       │
│  ├─ analyzers_alerts                                        │
│  ├─ analyzers_alert_clusters                               │
│  ├─ analyzers_features (140+ base features)                │
│  └─ core_money_flows                                        │
│                                                             │
│  PROPOSAL 4 - SOT Feature Engineering (Validator-side):    │
│  ├─ Graph embeddings (Node2Vec, DeepWalk)                  │
│  ├─ Temporal derivatives across windows                    │
│  ├─ Domain-specific AML signals                            │
│  └─ Added as columns to analyzers_features                 │
│                                                             │
│  PROPOSAL 5 - SOT Pattern Discovery (Validator-side):      │
│  ├─ Unsupervised anomaly detection                         │
│  ├─ Novel graph motif mining                               │
│  ├─ Structural pattern variations                          │
│  └─ New alerts added to analyzers_alerts                   │
│                                                             │
└─────────────────────────────────────────────────────────────┘
                  │
                  │
                  │ Enriched SOT Data
                  ▼
         ┌──────────────────┐
         │  MINERS (Consume) │
         ├──────────────────┤
         │ Use enriched SOT │
         │ to do ONLY:      │
         │ 1. Alert Scoring │
         │ 2. Prioritization│
         │ 3. Cluster Assess│
         └──────────────────┘
```

Proposal 4: Feature Engineering (SOT Internal)

What Validators Add to SOT

Enhanced features computed during daily SOT generation and added to `analyzers_features`.

Implementation in Daily Pipeline

```python
class FeatureBuilderTask:
    """
    Enhanced feature building in daily pipeline
    """

    def run(self, processing_date, window_days):
        # Step 1: Build base features (existing)
        base_features = self.build_base_features(processing_date, window_days)

        # PROPOSAL 4.1: Add graph embeddings
        graph_embeddings = self.compute_graph_embeddings(
            money_flows=self.get_money_flows(window_days),
            method='node2vec',
            dimensions=64
        )

        # PROPOSAL 4.2: Add temporal derivatives
        temporal_features = self.compute_temporal_derivatives(
            features_7d=self.get_features(7),
            features_30d=self.get_features(30),
            features_90d=self.get_features(90)
        )

        # PROPOSAL 4.3: Add AML-specific signals
        aml_signals = self.compute_aml_signals(
            base_features=base_features,
            money_flows=self.get_money_flows(window_days)
        )

        # Merge all features
        enhanced_features = self.merge_features(
            base_features,
            graph_embeddings,
            temporal_features,
            aml_signals
        )

        # Store in SOT
        self.feature_repository.save(enhanced_features)
```

Schema Extensions

```sql
-- Extend analyzers_features with Proposal 4 columns
ALTER TABLE analyzers_features ADD COLUMN IF NOT EXISTS
    -- Graph embeddings
    graph_embedding_64d Array(Float32),
    embedding_quality Float32,

    -- Temporal derivatives (window-over-window changes)
    volume_7d_vs_30d_ratio Float32,
    volume_30d_vs_90d_ratio Float32,
    degree_acceleration_7d Float32,
    behavior_shift_score Float32,

    -- AML-specific signals
    structuring_indicator Float32,
    rapid_movement_score Float32,
    mixer_proximity_hops UInt8,
    exchange_affinity_ratio Float32,
    layering_score Float32,
    smurfing_score Float32;
```

Feature Categories (Validator Computes)

```python
class SOTFeatureEngineer:
    """
    Advanced feature engineering for SOT
    """

    def compute_graph_embeddings(self, money_flows, dimensions=64):
        """
        Node2Vec embeddings for graph representation
        """
        from node2vec import Node2Vec

        G = self.build_networkx_graph(money_flows)
        node2vec = Node2Vec(
            G,
            dimensions=dimensions,
            walk_length=10,
            num_walks=20,
            workers=4,
            seed=42  # Deterministic
        )
        model = node2vec.fit(window=5, min_count=1)

        embeddings = {}
        for address in G.nodes():
            embeddings[address] = model.wv[address]

        return embeddings
    def compute_temporal_derivatives(self, features_7d, features_30d,
features_90d):
        """
        Cross-window feature comparisons
        """
        derivatives = {}

        for address in features_7d.keys():
            derivatives[address] = {
                # Volume trends
                'volume_7d_vs_30d': features_7d[address]['total_volume_usd'] /
                                  (features_30d[address]['total_volume_usd'] +
1e-9),
                'volume_30d_vs_90d': features_30d[address]['total_volume_usd'] /
                                   (features_90d[address]['total_volume_usd'] +
1e-9),

                # Degree acceleration
                'degree_acceleration': (features_7d[address]['degree_total'] -
                                      features_30d[address]['degree_total']) /
23,

                # Behavioral shift
                'behavior_shift': self.cosine_distance(
                    features_7d[address]['hourly_activity'],
                    features_30d[address]['hourly_activity']
                )
            }

        return derivatives

    def compute_aml_signals(self, base_features, money_flows):
        """
        Domain-specific AML detection signals
        """
        signals = {}

        for address, features in base_features.items():
            signals[address] = {
                # Structuring: transactions just below reporting thresholds
                'structuring_indicator': self.detect_structuring(
                    address, money_flows, threshold=10000
                ),

                # Rapid movement: volume per active hour
                'rapid_movement_score': features['total_volume_usd'] /
                                      max(1, features['activity_span_days'] *
24),

                # Mixer proximity: minimum hops to known mixer
                'mixer_proximity_hops': self.compute_min_hops_to_mixers(
                    address, money_flows
                ),
                # Exchange affinity: ratio of volume to exchanges
                'exchange_affinity': self.compute_exchange_affinity(
                    address, money_flows
                ),

                # Layering: rapid in-out patterns
                'layering_score': self.detect_layering(address, money_flows),

                # Smurfing: many small transactions from multiple sources
                'smurfing_score': self.detect_smurfing(address, money_flows)
            }

        return signals
```

Proposal 5: Pattern Discovery (SOT Internal)

What Validators Add to SOT

Novel alert types discovered through unsupervised learning and added to `analyzers_alerts`.

Implementation in Typology Detection

```
class EnhancedTypologyDetector:
    """
    Extends existing typology detector with unsupervised discovery
    """

    def detect_typologies(self, processing_date, window_days):
        # Step 1: Run existing rule-based typologies
        rule_based_alerts = self.run_rule_based_detection(
            processing_date, window_days
        )

        # PROPOSAL 5.1: Unsupervised anomaly detection
        anomaly_alerts = self.detect_novel_anomalies(
            processing_date, window_days
        )

        # PROPOSAL 5.2: Graph motif mining
        motif_alerts = self.discover_graph_patterns(
            processing_date, window_days
        )

        # PROPOSAL 5.3: Temporal pattern discovery
        temporal_alerts = self.discover_temporal_patterns(
            processing_date, window_days
        )

        # Merge all alert types
        all_alerts = self.merge_alerts(
            rule_based_alerts,
            anomaly_alerts,
            motif_alerts,
            temporal_alerts
        )

        # Store in SOT
        self.alerts_repository.save(all_alerts)
```

Unsupervised Detection Methods

```
class UnsupervisedPatternDiscovery:
    """
    Discover novel patterns not covered by rules
    """

    def detect_novel_anomalies(self, features, money_flows):
        """
        Ensemble of unsupervised detectors
        """
        # Method 1: Isolation Forest (already in features as anomaly scores)
        # Use existing behavioral_anomaly_score, graph_anomaly_score

        # Method 2: DBSCAN for density-based clusters
        outliers = self.dbscan_outliers(features)

        # Method 3: Autoencoder for reconstruction errors
        reconstruction_anomalies = self.autoencoder_detection(features)

        # Combine detectors
        novel_anomalies = self.ensemble_vote(
            outliers,
            reconstruction_anomalies,
            threshold=2  # Must be flagged by 2/3 detectors
        )

        return self.create_alerts(novel_anomalies,
typology_type='unsupervised_anomaly')

    def discover_graph_patterns(self, money_flows):
        """
        Mine novel graph motifs
        """
        G = self.build_graph(money_flows)
        patterns = []

        # Pattern 1: Circular flows (money returns to source)
        circular = self.find_circular_flows(G, max_hops=6)
        patterns.extend(circular)

        # Pattern 2: Hub-spoke patterns (1-to-many-to-1)
        hub_spoke = self.find_hub_spoke_patterns(G)
        patterns.extend(hub_spoke)

        # Pattern 3: Temporal bursts (many txs in narrow window)
        bursts = self.find_temporal_bursts(money_flows)
        patterns.extend(bursts)

        return self.create_alerts(patterns, typology_type='graph_motif')

    def find_circular_flows(self, G, max_hops=6):
        """
        Detect cycles that may indicate layering
        """
        cycles = []

        for node in G.nodes():
            # Find paths from node back to itself
            paths = self.find_cycles_from_node(G, node, max_hops)

            for path in paths:
                # Filter for suspicious characteristics
                if self.is_suspicious_cycle(path, G):
                    cycles.append({
                        'addresses': path,
                        'pattern_type': 'circular_flow',
                        'total_volume': sum(G[a][b]['volume'] for a,b in
zip(path[:-1], path[1:])),
                        'hops': len(path) - 1,
                        'confidence': self.compute_cycle_confidence(path, G)
                    })

        return cycles

    def is_suspicious_cycle(self, path, G):
        """
        Determine if cycle is likely illicit
        """
        # Check: Similar amounts (potential structuring)
        amounts = [G[a][b]['amount'] for a,b in zip(path[:-1], path[1:])]
        amount_variance = np.std(amounts) / (np.mean(amounts) + 1e-9)

        # Check: Temporal proximity (coordinated)
        timestamps = [G[a][b]['timestamp'] for a,b in zip(path[:-1], path[1:])]
        time_span = max(timestamps) - min(timestamps)

        # Check: Unusual for these addresses
        participants_history = [self.get_address_history(addr) for addr in path]

        return (
            amount_variance < 0.2  # Similar amounts
            and time_span < 3600 * 24  # Within 24 hours
            and all(h['first_cycle'] for h in participants_history)  # New
behavior
        )
```

Pattern Types Added to SOT

```python
# New typology types added to analyzers_alerts
NEW_TYPOLOGY_TYPES = {
    'unsupervised_anomaly': 'Detected by ensemble anomaly detectors',
    'circular_flow': 'Money returns to source via intermediaries',
    'hub_spoke_pattern': 'Centralized distribution then collection',
    'temporal_burst': 'Unusual concentration of activity',
    'value_laddering': 'Sequential transactions with increasing amounts',
    'coordinated_smurfing': 'Many sources, similar amounts, tight timing',
    'bridge_hopping': 'Rapid movement across multiple chains/protocols'
}
```

Benefits to Miners

Richer Feature Set (Proposal 4)

Miners receive **180+ features** instead of 140:

- Base features (140)
- Graph embeddings (64D = 64 features)
- Temporal derivatives (5)
- AML signals (6)

```python
# Miner can now use richer features
miner_model = LightGBM(
    features=[
        # Base features
        'total_volume_usd',
        'degree_total',
        'behavioral_anomaly_score',

        # NEW: Graph embeddings (from Proposal 4)
        'graph_embedding_64d[0]',  # 64 dimensions
        'graph_embedding_64d[1]',
        # ...

        # NEW: Temporal features (from Proposal 4)
        'volume_7d_vs_30d_ratio',
        'degree_acceleration_7d',

        # NEW: AML signals (from Proposal 4)
        'structuring_indicator',
        'mixer_proximity_hops',
        'layering_score'
    ]
)
```

More Alert Types (Proposal 5)

Miners score **more diverse alerts**:

- Rule-based alerts: ~60% of daily alerts
- Unsupervised anomalies: ~20%
- Graph motifs: ~15%
- Temporal patterns: ~5%

This diversity makes the scoring task more challenging and valuable.

Implementation Timeline

Phase 1: Feature Engineering (Proposal 4)

Week 1-2: Graph Embeddings

```
# Add to daily pipeline
packages/jobs/tasks/build_features_task.py
├─ compute_graph_embeddings()
└─ Add columns to analyzers_features
```

Week 3-4: Temporal Derivatives

```
# Cross-window feature computation
├─ Load features from 7d, 30d, 90d windows
├─ Compute ratios and differences
└─ Add columns to analyzers_features
```

Week 5-6: AML Signals

```
# Domain-specific feature engineering
├─ Structuring detection
├─ Mixer proximity
├─ Layering/smurfing scores
└─ Add columns to analyzers_features
```

Phase 2: Pattern Discovery (Proposal 5)

Week 7-8: Unsupervised Anomalies

```
# Add to typology detector
packages/analyzers/typologies/typology_detector.py
```

```
├─ DBSCAN outlier detection
├─ Autoencoder anomalies
└─ Add alerts to analyzers_alerts
```

Week 9-10: Graph Motifs

```
# Graph pattern mining
├─ Circular flow detection
├─ Hub-spoke patterns
└─ Add alerts to analyzers_alerts
```

Week 11-12: Temporal Patterns

```
# Time-based pattern discovery
├─ Burst detection
├─ Coordinated activity
└─ Add alerts to analyzers_alerts
```

Validation of SOT Extensions

How to Validate Enhanced Features Work

```python
# A/B test: baseline vs enhanced features
def validate_feature_engineering():
    """
    Compare miner performance with base vs enhanced features
    """
    # Group 1: Miners using only base features (140)
    baseline_scores = get_miner_scores(feature_set='base')

    # Group 2: Miners using enhanced features (180+)
    enhanced_scores = get_miner_scores(feature_set='enhanced')

    # Compare at T+τ
    improvement = compute_performance_gain(
        baseline_scores,
        enhanced_scores,
        metric='brier_score'
    )

    assert improvement > 0, "Enhanced features should improve predictions"
```

How to Validate Pattern Discovery Works

```python
# Measure discovery quality at T+τ
def validate_pattern_discovery():
    """
    Confirm discovered patterns have predictive value
    """
    # Compare confirmation rates
    rule_based_confirmation = get_confirmation_rate(
        alert_type='rule_based'
    )

    discovered_confirmation = get_confirmation_rate(
        alert_type='unsupervised_anomaly'
    )

    # Discovered patterns should have comparable or better rates
    assert discovered_confirmation >= rule_based_confirmation * 0.8, \
        "Discovered patterns should be meaningful"
```

Conclusion

Proposals 4 & 5 are SOT internal enhancements that:

1. **Enrich the data** provided to miners (more features, more alert types)
2. **Done by validators** as part of daily SOT generation
3. **Improve miner performance** by giving them better input data
4. **Continuously evolve** the AML detection capabilities

Miners do NOT do Proposals 4 & 5. Miners only consume the enriched SOT data to perform Proposals 1, 2, 3 (scoring, prioritization, cluster assessment).