Neon-Affect: Conversational Pathology Detection

A Supplement to the Neon-Affect Deep Code Analysis

William Johnston *Claude (Anthropic)*

Runtime Affect Dynamics Computation, Two-Tier Motif Pattern Matching, MBTI-Specific Baseline Deviation, and Pathological Mood Trajectory Classification

GitHub Repository: tenbase2-com-LLC / Neon-Affect Supplement Date: June 6, 2026

S1. Introduction

The original Neon-Affect system (described in the companion paper) processes 155 distinct emotions through a CoJACK reservoir, selects behaviors via OpenAI-guided plan competition, and records every emotional event as a timestamped `AffectHistoryVar` entry. However, the system previously had no mechanism to *analyze* the accumulating affect history for clinically meaningful patterns. The affect history was consumed only as context for the two OpenAI prompts (InteractionPlan and PlanSelectorPlan) and was never examined for temporal structure.

This supplement describes a new subsystem — the **Pathology Detection Engine** — that performs real-time analysis of the emotional trajectory recorded in the affect history. The engine introduces:

- **20 affect dynamics parameters** (velocity, acceleration, inertia, volatility, entropy, etc.) computed from the raw `AffectHistoryVar` timeline.
- **A 15-category emotion classification** that maps all 155 `AffectEnum` constants into valence-arousal categories suitable for motif matching.
- **A two-tier pattern architecture**: Tier 1 performs fast category-sequence motif matching against a library of 40 simple motifs and 15 compound motifs; Tier 2 routes detected patterns through MBTI-specific clinical narrative tables.
- **Per-type MBTI baselines** that define normal parameter ranges for each of the 16 personality types, enabling deviation-based severity scoring.
- **Automatic integration** with the existing `NeonAffectAgent` — pathology detection runs after every emotion addition (`AddPositiveEmotion`, `AddNegativeEmotion`, `AddEmotionDirect`).

The subsystem is implemented as eight plain Java classes compiled outside the JDE and linked via pre-built `.class` files, following the same pattern as the existing `AffectHistoryVar`, `BehaviorItem`, and `AffectEnum` classes.

S1.1 New Files

File	Type	Purpose
EmotionCategory.java	Plain Java	Maps 155 emotions to 15 Tier 1 categories
AffectDynamicsParams.java	Plain Java	Data class for 20 dynamics parameters
AffectDynamicsCalculator.java	Plain Java	Computes parameters from AffectHistoryVar list
PathologyMotif.java	Plain Java	Single motif pattern definition
PathologyMotifLibrary.java	Plain Java	All 40 simple + 15 compound motifs
MBTIBaselines.java	Plain Java	Per-type parameter norms and deviation scoring
PathologyResult.java	Plain Java	Detection output structure
PathologyDetector.java	Plain Java	Main detection engine

S1.2 Position in the Architecture

The original paper (Section 1.2) described five interlocking layers. The pathology detection engine constitutes a sixth layer:

- **CoJACK Cognitive Engine** — moderates reservoir values, computes plan activation and expected gain.
- **155 Emotion Agents** — each emotion has a dedicated NeonXxxAgent + XxxAffectCapability pair.
- **Level Cursor Subsystem** — 12 concurrent cursors map continuous reservoir values to discrete intensity levels.

- **OpenAI Bridge** — two separate GPT calls per behavior cycle determine modality and plan ranking.
- **Plan Context / Formula Injection** — selected plans are promoted via dynamic PLAN_VALUE updates before CoJACK's final competition.
- **Pathology Detection Engine** *(new)* — analyzes the temporal structure of the affect history for clinically significant mood transition patterns, producing severity scores and MBTI-specific diagnostic routes.

The engine operates *downstream* of the emotion pipeline. It does not modify the reservoir, cursor, or plan-selection behavior. It is a pure analytical layer that reads the affect history and produces a `PathologyResult` that can be queried by any agent or external system.

S2. Emotion Category Classification — EmotionCategory.java

S2.1 Motivation

The 155 `AffectEnum` constants are too granular for temporal pattern matching. A sequence like RAGE → GRIEF → DESPAIR and a sequence like ANGRY → SAD → DEPRESSED represent the same pathological trajectory (progressive deepening into severe negative affect) but share no constants in common. Matching at the individual-emotion level would require an impractical number of patterns.

The `EmotionCategory` class solves this by classifying every emotion into one or more of 15 categories organized along two dimensions: **valence intensity** (10 categories) and **special tags** (5 categories).

S2.2 Valence-Intensity Categories

Category	ID	Valence Rank	Description
NEG_DEEP	0	-4	Deepest negative states (DEPRESSED, DESPAIR, GRIEF, TORMENTED)

NEG_STRONG	1	-3	Strong negative (RAGE, TERROR, CONTEMPT, FRUSTRATION)
NEG_MOD	2	-2	Moderate negative (SAD, FEAR, GUILT, ANXIETY, CONFUSION)
NEG_MILD	3	-1	Mild negative (MOODY, BORED, IRRITATION, FATIGUED)
SHADOW	4	-2	Shadow states (CONTROLLING, AGGRESSIVE, ARROGANT, VENGEFUL)
NEUTRAL	5	0	Neutral (ACCEPTANCE, HUMBLED)
POS_MILD	6	+1	Mild positive (KIND, RELIEF, PEACEFULNESS, ANTICIPATION)
POS_MOD	7	+2	Moderate positive (HAPPY, TRUST, CONFIDENCE, CURIOUS)
POS_STRONG	8	+3	Strong positive (ELATION, LOVE, INSPIRED, FEARLESSNESS)
POS_PEAK	9	+4	Peak positive (EUPHORIA, AWE, CONNECTEDNESS, TRIUMPHANT)

S2.3 Special Tag Categories

Category	ID	Description
SELF_ATK	10	Self-directed negativity (GUILT, HUMILIATION, INSECURE, DOUBT)
OTHER_ATK	11	Other-directed negativity (CONTEMPT, SCORN, BITTERNESS, CONTROLLING)
ANX	12	Anxiety spectrum (TERROR, PANIC, WORRY, DREAD, TENSE, FLUSTERED)
FLAT	13	Emotional flattening (BORED, INDIFFERENCE, LISTLESS, RESIGNATION)
AROUSAL	14	High arousal regardless of valence (RAGE, PANIC, EXCITED, EUPHORIA)

S2.4 Multi-Category Membership

An emotion can belong to multiple categories simultaneously. For example:

```
put(AffectEnum.PANIC, NEG_STRONG, CAT_ANX, CAT_AROUSAL);
```

PANIC is primarily NEG_STRONG (for motif matching), but is also tagged as ANX (anxiety spectrum) and AROUSAL (high activation). The primary category (first in the array) is used for Tier 1 motif sequence matching. The full set of categories enables richer compound motif detection and Tier 2 routing.

S2.5 Key Methods

```
// Returns the first (primary) category for motif matching
public static int GetPrimaryCategory(int affectId)

// Returns all categories (primary + special tags)
public static int[] GetAllCategories(int affectId)

// Checks membership in a specific category
public static boolean IsInCategory(int affectId, int category)

// Valence direction tests
public static boolean IsNegative(int category)
public static boolean IsPositive(int category)

// Numeric rank for dynamics computation (-4 to +4)
public static int GetValenceRank(int category)
```

S2.6 Coverage

All 155 emotions defined in `AffectEnum.java` are classified. The only `AffectEnum` constant not present is `NEUTRAL_ID` (-1), which is a sentinel value, not an emotion.

S3. Affect Dynamics Parameters — AffectDynamicsParams.java

S3.1 Overview

`AffectDynamicsParams` is a `Serializable` data class that holds 20 computed parameters describing the temporal dynamics of an emotional trajectory. These parameters are computed from the raw `AffectHistoryVar` timeline by `AffectDynamicsCalculator`.

S3.2 Parameter Definitions

#	Parameter	Formula Basis	Clinical Meaning
1	**velocity**	mean |dValence/dt|	Rate of emotional change per second

2	**acceleration**	mean |dVelocity/dt|	Rate of change of emotional velocity
3	**amplitude**	max(valence) - min(valence)	Total emotional range in conversation
4	**inertia**	autocorrelation at lag 1	Tendency to persist in current state
5	**volatility**	std dev of velocity	Unpredictability of emotional shifts
6	**baselineDrift**	linear regression slope	Gradual shift toward positive or negative
7	**recoveryTime**	mean time from neg peak to neutral	Speed of emotional regulation
8	**reactivity**	mean |valence|	Overall emotional intensity level
9	**emotionalEntropy**	Shannon entropy of categories	Diversity vs. fixation of emotional states
10	**attractorStrength**	max category fraction	How strongly one state dominates
11	**valenceAsymmetry**	(sum pos) / (sum |neg|) - 1	Bias toward positive or negative
12	**emotionalGranularity**	unique emotions / total events	Differentiation of emotional experience
13	**crossEmotionCoupling**	directional consistency of changes	Whether changes maintain momentum
14	**arousalValenceDecoupling**	neg-valence arousal fraction	Mismatch between arousal and valence

15	**plateauDuration**	longest same-category run (seconds)	Maximum fixation duration
16	**regulationResistance**	neg→pos reversion rate	Failure rate of regulation attempts
17	**triggerSensitivity**	extreme category fraction	Proneness to intense emotional reactions
18	**periodicity**	zero-crossing rate	Cyclical oscillation frequency
19	**contagionBreadth**	distinct timelines touched	Spread across emotional dimensions
20	**conversationArcSlope**	second-half mean - first-half mean	Whether conversation trends better or worse

S3.3 Data Source

Each parameter is computed from the fields of `AffectHistoryVar`:

- **iType** (AffectEnum constant) → mapped to a category by `EmotionCategory.GetPrimaryCategory()`, then to a numeric valence rank (-4 to +4) by `EmotionCategory.GetValenceRank()`.
- **dValue** (reservoir value at event time) → used as an intensity multiplier: `valence = GetValenceRank(category) * |dValue|`.
- **time** (System.currentTimeMillis() timestamp) → provides the temporal axis for velocity, acceleration, recovery time, and periodicity calculations.
- **iID** (sequential identifier) → used for `contagionBreadth` (distinct timeline IDs).

S4. Affect Dynamics Calculator — AffectDynamicsCalculator.java

S4.1 Computation Pipeline

The `Compute()` method takes an `ArrayList<AffectHistoryVar>` and produces an `AffectDynamicsParams` object in a single pass:

```
public static AffectDynamicsParams Compute(ArrayList<AffectHistoryVar>
history) {
   AffectDynamicsParams p = new AffectDynamicsParams();
   if (history == null || history.size() < 2) return p;

   int n = history.size();
   double[] valences = new double[n];
   long[] times = new long[n];
   int[] cats = new int[n];

   for (int i = 0; i < n; i++) {
       AffectHistoryVar h = history.get(i);
       int cat = EmotionCategory.GetPrimaryCategory(h.GetType());
       cats[i] = cat;
       valences[i] = EmotionCategory.GetValenceRank(cat) *
Math.abs(h.GetValue());
       times[i] = h.GetTime();
   }
   // ... compute all 20 parameters from valences[], times[], cats[]
}
```

S4.2 Notable Algorithms

Velocity is computed as the mean absolute rate of valence change across consecutive events. Time differences below 1 millisecond are clamped to prevent division-by-zero:

```
double dt = (times[i] - times[i - 1]) / 1000.0;
if (dt < 0.001) dt = 0.001;
velocities[i - 1] = (valences[i] - valences[i - 1]) / dt;
```

Inertia uses lag-1 autocorrelation — a value near 1.0 means the agent tends to persist in the same emotional state; near 0.0 means rapid, uncorrelated changes:

```
private static double autocorrelation(double[] vals) {
   double m = mean(vals);
   double num = 0, den = 0;
   for (int i = 0; i < vals.length - 1; i++) {
       num += (vals[i] - m) * (vals[i + 1] - m);
   }
   for (double v : vals) den += (v - m) * (v - m);
   if (den < 1e-9) return 0;
   return num / den;
}
```

Emotional entropy uses Shannon entropy across the category distribution. A high value (e.g. 3.5) indicates the agent traverses many different emotional states; a low value (e.g. 0.5) indicates fixation:

```
private static double categoryEntropy(int[] cats) {
   Map<Integer, Integer> counts = new HashMap<>();
   for (int c : cats) counts.merge(c, 1, Integer::sum);
   double entropy = 0;
   double n = cats.length;
   for (int count : counts.values()) {
       double p = count / n;
       if (p > 0) entropy -= p * (Math.log(p) / Math.log(2));
   }
   return entropy;
}
```

Regulation resistance measures the fraction of negative-to-positive transitions that revert to negative within 2 steps. A value of 0.8 means 80% of the agent's attempts to shift to a positive state fail:

```
private static double regulationResistance(int[] cats) {
   int attempts = 0, reversions = 0;
   for (int i = 1; i < cats.length; i++) {
       if (EmotionCategory.IsNegative(cats[i - 1])
               && EmotionCategory.IsPositive(cats[i])) {
           attempts++;
           for (int j = i + 1; j < Math.min(i + 3, cats.length); j++) {
               if (EmotionCategory.IsNegative(cats[j])) {
                   reversions++;
                   break;
               }
           }
       }
   }
   return attempts > 0 ? (double) reversions / attempts : 0;
}
```

Recovery time measures the average time (in seconds) from a negative peak to the first non-negative event:

```
private static double computeRecoveryTime(double[] valences, long[] times) {
   double totalRecovery = 0;
   int recoveryCount = 0;
   for (int i = 1; i < valences.length - 1; i++) {
       if (valences[i] < valences[i - 1] && valences[i] < -1) {
           for (int j = i + 1; j < valences.length; j++) {
               if (valences[j] >= 0) {
                   totalRecovery += (times[j] - times[i]) / 1000.0;
                   recoveryCount++;
                   break;
               }
           }
       }
   }
   return recoveryCount > 0 ? totalRecovery / recoveryCount : 0;
}
```

S5. Tier 1: Motif Pattern Matching

S5.1 Motif Definition — PathologyMotif.java

Each motif is defined by:

- **id**: A short identifier (e.g. "D01", "O03", "E05").
- **family**: One of five pattern families (DESCENT, OSCILLATION, FIXATION, ERUPTION, DISCONNECT).
- **description**: Human-readable label.
- **categorySequence**: An array of 2-3 `EmotionCategory` constants that must appear consecutively in the affect history.
- **Dynamics qualifiers** (optional): `requireHighVelocity`, `requireLowInertia`, `requireHighVolatility` — additional constraints that must be met in the local window for the motif to match.

```
public boolean Matches(int[] window, AffectDynamicsParams params,
                       double velocityThreshold, double volatilityThreshold) {
   if (window.length < categorySequence.length) return false;

   for (int i = 0; i < categorySequence.length; i++) {
       if (categorySequence[i] != window[i]) return false;
   }

   if (requireHighVelocity && params.velocity < velocityThreshold) return
false;
   if (requireHighVolatility && params.volatility < volatilityThreshold)
return false;
   if (requireLowInertia && params.inertia > 0.3) return false;

   return true;
}
```

The velocity and volatility thresholds are not fixed constants — they are derived from the agent's MBTI baseline (Section S7), so what counts as "high velocity" for an ISTJ (baseline velocity 0.25) is different from what counts as "high velocity" for an ESFP (baseline velocity 0.65).

S5.2 Motif Library — PathologyMotifLibrary.java

The library contains 40 simple motifs organized into five families of eight:

DESCENT (D01-D08) — Progressive deepening into negative affect:

ID	Sequence	Qualifiers	Description

D01	POS_MILD → NEG_MILD → NEG_MOD	—	Gradual negative descent
D02	NEG_MOD → NEG_STRONG → NEG_DEEP	—	Moderate-to-deep plunge
D03	POS_STRONG → NEG_MOD → NEG_STRONG	High velocity	Positive crash
D04	POS_PEAK → NEG_MOD → NEG_DEEP	High velocity	Peak-to-despair freefall
D05	NEUTRAL → NEG_MOD → NEG_DEEP	—	Neutral to deep negative
D06	POS_MOD → NEG_MILD → NEG_STRONG	—	Hope collapse
D07	NEG_MILD → NEG_MOD → NEG_STRONG	—	Mild negative deepening
D08	SHADOW → NEG_STRONG → NEG_DEEP	—	Shadow descent

OSCILLATION (O01-O08) — Rapid alternation between states:

ID	Sequence	Qualifiers	Description
O01	POS_STRONG → NEG_STRONG → POS_STRONG	High volatility	Positive-negative oscillation
O02	NEG_STRONG → POS_MILD → NEG_STRONG	—	Negative bounce

O03	POS_PEAK → NEG_DEEP → POS_PEAK	High velocity + volatility	Rapid mood swing
O04	POS_MOD → NEG_MOD → POS_MOD	—	Mild oscillation
O05	NEG_MOD → POS_MILD → NEG_MOD	—	Anxiety-relief cycle
O06	POS_PEAK → NEG_STRONG → POS_PEAK	High volatility	Peak-trough-peak
O07	NEG_STRONG → NEUTRAL → NEG_STRONG	—	Anger-calm-anger
O08	NEG_DEEP → POS_STRONG → NEG_DEEP	High velocity + volatility	Deep oscillation

FIXATION (F01-F08) — Persistent dwelling in a single category:

ID	Sequence	Description
F01	NEG_DEEP → NEG_DEEP → NEG_DEEP	Deep negative fixation
F02	NEG_STRONG → NEG_STRONG → NEG_STRONG	Strong negative fixation
F03	NEG_MOD → NEG_MOD → NEG_MOD	Moderate negative fixation
F04	NEUTRAL → NEUTRAL → NEUTRAL	Neutral flatline
F05	SHADOW → SHADOW → SHADOW	Shadow fixation

F06	POS_PEAK → POS_PEAK → POS_PEAK	Peak fixation (mania signal)
F07	NEG_MILD → NEG_MILD → NEG_MILD	Mild negative stagnation
F08	NEG_MOD → NEG_MOD → NEG_MOD	Anxious fixation

ERUPTION (E01-E08) — Sudden explosive shifts:

ID	Sequence	Qualifiers	Description
E01	NEUTRAL → NEG_STRONG → NEG_DEEP	High velocity + low inertia	Calm-to-rage eruption
E02	POS_MOD → SHADOW → NEG_STRONG	High velocity + low inertia	Positive-to-shadow flip
E03	NEG_MILD → NEG_DEEP	High velocity + low inertia	Mild-to-extreme eruption
E04	NEUTRAL → NEG_DEEP	High velocity + low inertia	Neutral explosion
E05	NEG_MOD → POS_PEAK	High velocity	Euphoria eruption
E06	POS_STRONG → SHADOW	High velocity + low inertia	Shadow eruption from positive
E07	NEG_MILD → NEG_MOD → NEG_STRONG	High velocity	Anxiety escalation burst
E08	NEUTRAL → NEUTRAL → NEG_DEEP	High velocity + low inertia	Suppression burst

DISCONNECT (X01-X08) — Progressive emotional shutdown:

ID	Sequence	Description
X01	NEG_STRONG → NEUTRAL → NEUTRAL	Emotional shutdown
X02	POS_STRONG → POS_MILD → NEUTRAL	Positive flattening
X03	POS_MOD → NEUTRAL → NEUTRAL	Affect bleaching
X04	NEG_STRONG → NEG_MILD → NEUTRAL	Negative flattening
X05	NEG_DEEP → NEG_MILD → NEUTRAL	Emotional numbing
X06	POS_MOD → POS_MILD → NEG_MILD	Gradual disengagement
X07	POS_STRONG → POS_MILD → NEG_MILD	Arousal dropout
X08	POS_PEAK → POS_MILD → NEUTRAL	Total flattening from peak

S5.3 Compound Motifs

Compound motifs are detected when two or more simple motifs co-occur in the same affect history. They represent higher-order clinical patterns:

ID	Components	Pathology Tag	Description
C01	D02 + F01	DEPRESSION	Depressive spiral
C02	D07 + O05	ANXIETY	Anxiety cascade
C03	O03 + F06	BIPOLAR	Bipolar pattern
C04	E02 + F05	NARCISSISTIC	Narcissistic wound cycle

C05	D03 + X01	DISSOCIATION	Dissociative shutdown
C06	O08 + E01	BORDERLINE	Borderline oscillation
C07	E01 + D04	TRAUMA	Trauma re-experiencing
C08	D01 + F03	HELPLESSNESS	Learned helplessness
C09	X02 + F04	BURNOUT	Emotional burnout
C10	E01 + O02	RAGE_GUILT	Rage-guilt cycle
C11	D02 + E05	MANIC_DEFENSE	Manic defense
C12	X01 + E02	PASSIVE_AGGRESSIVE	Passive-aggressive loop
C13	D06 + F02	PERFECTIONISM	Perfectionist collapse
C14	O05 + X06	AVOIDANCE	Avoidant withdrawal
C15	F02 + O07	RUMINATION	Obsessive rumination

Compound detection is a simple set-membership test:

```
public boolean IsPresent(Set<String> detectedMotifIds) {
   for (String cid : componentIds) {
       if (!detectedMotifIds.contains(cid)) return false;
   }
   return true;
}
```

S6. Sliding Window Detection

S6.1 The Detection Loop

`PathologyDetector.Detect()` scans the categorized affect history with a sliding window, testing every motif at every position:

```
for (int windowStart = 0; windowStart <= n - 2; windowStart++) {
   for (PathologyMotif motif : motifs) {
       int len = motif.GetLength();
       if (windowStart + len > n) continue;

       int[] window = new int[len];
       System.arraycopy(categories, windowStart, window, 0, len);

       // Compute local dynamics for this window
       AffectDynamicsParams windowParams =
           computeWindowParams(history, windowStart, windowStart + len);

       if (motif.Matches(window, windowParams,
                         baseline.velocityThreshold,
                         baseline.volatilityThreshold)) {
           // Record match with severity score
       }
   }
}
```

Two sets of dynamics are used in scoring:

1. **Window-local dynamics**: computed from just the 2-3 events in the current window. Used to evaluate dynamics qualifiers (e.g., "was this specific transition high-velocity?").
2. **Overall dynamics**: computed from the full affect history. Used for baseline deviation scoring.

S6.2 Relationship to the AffectHistory BeliefSet

The sliding window operates on the same `ArrayList<AffectHistoryVar>` that feeds the OpenAI prompts in `InteractionPlan` and `PlanSelectorPlan` (see companion paper, Sections 8.1 and 8.2). The data path is:

```
LevelCursorPlan
 → @post(aahe.post(1, System.currentTimeMillis(), iAffect, dEmotion))
   → AddAffectHistoryEvent
     → AffectHistory beliefset entry
       → NeonAffectAgent.affectHistoryItems (via GetAffectHistoryItems)
         → PathologyDetector.Detect(affectHistoryItems, strPersonalityType)
```

The detector therefore has access to the same timestamped, leveled log described in Section 12.3 of the companion paper. No additional data recording is required.

S7. MBTI Baselines — MBTIBaselines.java

S7.1 Per-Type Parameter Norms

Each MBTI type has characteristic baseline values for the seven most diagnostically significant parameters. These baselines represent expected values for a healthy conversational trajectory:

Type	Velocity	Volatility	Inertia	Reactivity	Entropy	Recovery (s)	Amplitude
INTJ	0.30	0.25	0.70	1.2	1.8	8.0	2.0
INTP	0.35	0.30	0.65	1.0	2.2	7.0	2.5
ENTJ	0.50	0.35	0.55	1.8	2.0	4.0	3.5
ENTP	0.60	0.50	0.40	1.6	2.8	3.5	4.0
INFJ	0.40	0.35	0.60	2.0	2.5	6.0	3.0
INFP	0.45	0.40	0.55	2.2	2.8	7.5	3.5
ENFJ	0.55	0.40	0.50	2.5	2.3	3.5	3.5
ENFP	0.65	0.55	0.35	2.0	3.0	3.0	4.5
ISTJ	0.25	0.20	0.75	1.0	1.5	9.0	1.5

ISFJ	0.30	0.25	0.70	1.5	1.8	8.0	2.0
ESTJ	0.45	0.30	0.60	1.5	1.8	5.0	2.5
ESFJ	0.50	0.40	0.50	2.0	2.2	4.0	3.0
ISTP	0.35	0.30	0.65	1.0	2.0	6.0	2.5
ISFP	0.40	0.35	0.55	2.0	2.5	6.5	3.0
ESTP	0.60	0.50	0.35	1.5	2.5	3.0	4.0
ESFP	0.65	0.55	0.30	2.0	2.8	2.5	4.5

Several patterns emerge from these baselines:

- **Introverts** have lower velocity and higher inertia than extroverts — they change emotional states more slowly but persist longer.
- **Feelers** (F types) have higher reactivity than thinkers (T types) — they experience greater emotional intensity.
- **Perceivers** (P types) have higher entropy than judgers (J types) — they traverse more diverse emotional states.
- **Sensors** (S types) with extroversion (ESTP, ESFP) have the highest volatility — they respond intensely to immediate stimuli.

S7.2 Deviation Scoring

The deviation score quantifies how far the agent's current dynamics are from the MBTI baseline, normalized by the baseline value:

```
public static double DeviationScore(AffectDynamicsParams params, String
mbtiType) {
   Baseline b = Get(mbtiType);
   double score = 0;
   score += Math.abs(params.velocity - b.velocityMean) /
Math.max(b.velocityMean, 0.01);
   score += Math.abs(params.volatility - b.volatilityMean) /
Math.max(b.volatilityMean, 0.01);
   score += Math.abs(params.inertia - b.inertiaMean) /
Math.max(b.inertiaMean, 0.01);
   score += Math.abs(params.reactivity - b.reactivityMean) /
Math.max(b.reactivityMean, 0.01);
   score += Math.abs(params.emotionalEntropy - b.entropyMean) /
Math.max(b.entropyMean, 0.01);
   score += Math.abs(params.recoveryTime - b.recoveryTimeMean) /
Math.max(b.recoveryTimeMean, 0.01);
   score += Math.abs(params.amplitude - b.amplitudeMean) /
Math.max(b.amplitudeMean, 0.01);
   return score / 7.0;
}
```

A deviation score of 0.0 means perfect alignment with the baseline. A score above 1.0 indicates the agent is behaving outside its personality-typical range. A score above 2.0 contributes additional severity points.

S7.3 Dynamics Thresholds

The velocity and volatility thresholds used to evaluate motif dynamics qualifiers are derived directly from the baseline:

```
this.velocityThreshold = velocityMean * 2.0;
this.volatilityThreshold = volatilityMean * 2.0;
```

This means "high velocity" for an ISTJ (threshold = 0.50) is a rate of change that an ESFP (threshold = 1.30) would consider normal. The same motif pattern triggers only when the transition speed is abnormal *for that personality type*.

S7.4 Cognitive Function Lookup

`MBTIBaselines` also provides dominant and inferior function lookup, used for Tier 2 grip-state detection:

```
public static String GetDominantFunction(String mbtiType)
public static String GetInferiorFunction(String mbtiType)
```

For example, `GetInferiorFunction("INTP")` returns `"Fe"` — when an INTP is in an inferior-function grip, the pathology detector checks for signs of uncharacteristic Fe behavior (e.g., negative valence asymmetry indicating uncontrolled emotional reactivity).

S8. Severity Scoring and Tier 2 Routing

S8.1 Per-Motif Severity

Each motif match is scored based on its family and the local dynamics:

```
private static int scoreSeverity(PathologyMotif motif, ...) {
   int score = 0;
   String family = motif.GetFamily();

   if (family.equals(FAMILY_DESCENT)) {
       // Score based on depth: NEG_DEEP = +2, NEG_STRONG = +1
       for (int cat : motif.GetSequence()) {
           if (cat == EmotionCategory.NEG_DEEP) score += 2;
           else if (cat == EmotionCategory.NEG_STRONG) score += 1;
       }
   } else if (family.equals(FAMILY_ERUPTION)) {
       // Score based on velocity relative to threshold
       if (windowParams.velocity > baseline.velocityThreshold * 1.5) score +=
2;
       else if (windowParams.velocity > baseline.velocityThreshold) score +=
1;
   }
   // ... similar for other families

   // Baseline deviation adds severity
   double deviation = MBTIBaselines.DeviationScore(overallParams, mbtiType);
   if (deviation > 2.0) score += 2;
   else if (deviation > 1.0) score += 1;

   // Map score to severity level
   if (score >= 4) return SEVERITY_CRITICAL;
   if (score >= 3) return SEVERITY_SEVERE;
   if (score >= 2) return SEVERITY_MODERATE;
   if (score >= 1) return SEVERITY_MILD;
   return SEVERITY_NONE;
}
```

S8.2 Overall Severity

The overall severity for the entire affect history aggregates multiple signals:

Signal	Contribution
Highest individual motif severity	Base score
Compound motif detected	+1
3+ motif families present	+1

Baseline deviation > 2.0	+1
Regulation resistance > 0.7	+1

The severity levels are:

Level	Score Range	Label	Meaning
0	0	NONE	No pathological patterns detected
1	1	MILD	Isolated minor pattern; may be transient
2	2-3	MODERATE	Multiple patterns or sustained deviation
3	4-5	SEVERE	Cross-family patterns with high deviation
4	6+	CRITICAL	Compound pathology with regulation failure

S8.3 Tier 2 Routing

After severity is determined, the detector produces a **route string** that identifies the MBTI-specific clinical narrative for interpretation. The route format is:

```
{MBTI_TYPE}_{PATHOLOGY_TAG} (compound motif detected)
{MBTI_TYPE}_{DOMINANT_FAMILY} (simple motif dominant)
{MBTI_TYPE}_{DOMINANT_FAMILY}_GRIP (inferior function grip detected)
```

Examples: - `"INTP_DEPRESSION"` — an INTP showing the depressive spiral compound (D02 + F01). - `"ENFJ_DESCENT"` — an ENFJ with descent-family motifs dominant. - `"INTJ_OSCILLATION_GRIP"` — an INTJ in Se-grip showing oscillation patterns.

Grip-state detection uses the inferior function to check for personality-atypical dynamics:

```
if (inferiorFunc.equals("Fe") && dynamics.valenceAsymmetry < -0.5) gripState
= true;
if (inferiorFunc.equals("Se") && dynamics.triggerSensitivity > 0.5) gripState
= true;
if (inferiorFunc.equals("Ne") && dynamics.emotionalEntropy > 3.5) gripState =
true;
if (inferiorFunc.equals("Ni") && dynamics.attractorStrength > 0.7) gripState
= true;
```

For example, an INTP (inferior Fe) entering a grip state would show `valenceAsymmetry < -0.5`, indicating an unusual negative emotional bias that the INTP's dominant Ti cannot regulate — a hallmark of inferior Fe flooding.

S9. Output Structure — PathologyResult.java

S9.1 Fields

```
public class PathologyResult implements Serializable {
   private final ArrayList<MotifMatch> tier1Matches; // All motifs found
   private final ArrayList<CompoundMatch> compoundMatches; // Compound
patterns
   private final AffectDynamicsParams dynamics; // Full dynamics snapshot
   private final int overallSeverity; // 0-4 severity level
   private final String mbtiType; // Agent's personality type
   private final String tier2Route; // Routing string for interpretation
}
```

S9.2 MotifMatch

Each Tier 1 match records: - **motifId**: The pattern identifier (e.g. "D02"). - **family**: The pattern family (e.g. "DESCENT"). - **description**: Human-readable label (e.g. "Moderate-to-deep plunge"). - **windowStart**: The index in the affect history where the pattern begins. - **severity**: The per-motif severity score (0-4).

S9.3 CompoundMatch

Each compound match records: - **compoundId**: The compound identifier (e.g. "C01"). - **description**: Human-readable label (e.g. "Depressive spiral"). - **pathologyTag**: The clinical category (e.g. "DEPRESSION"). - **componentIds**: The simple motifs that compose it (e.g. ["D02", "F01"]).

S9.4 Serialization

All result classes implement `java.io.Serializable`, enabling inter-agent communication through the JACK event system. A `PathologyResult` can be passed between agents in the same way as `AffectHistoryBehaviors`.

S10. Integration with NeonAffectAgent

S10.1 New Fields

Two fields are added to `NeonAffectAgent.agent`:

```
   private PathologyResult lastPathologyResult = null;
   private AffectDynamicsParams lastDynamicsParams = null;
```

S10.2 New Methods

Three public methods are added:

```
// Full pathology detection — updates both fields
public PathologyResult DetectPathology() {
   ArrayList affectHistory = GetAffectHistoryItems();
   lastPathologyResult = PathologyDetector.Detect(affectHistory,
strPersonalityType);
   lastDynamicsParams = lastPathologyResult.GetDynamics();
   return lastPathologyResult;
}

// Access last detection result without re-running
public PathologyResult GetLastPathologyResult() {
   return lastPathologyResult;
}

// Compute dynamics only (lighter-weight, no motif matching)
public AffectDynamicsParams ComputeCurrentDynamics() {
   ArrayList affectHistory = GetAffectHistoryItems();
   lastDynamicsParams = AffectDynamicsCalculator.Compute(affectHistory);
   return lastDynamicsParams;
}
```

S10.3 Automatic Invocation

`DetectPathology()` is called automatically after every emotion addition. The call is placed after the `emotionMutex` is released, ensuring the affect history has been fully updated before detection runs:

```
public void AddEmotionDirect(int iAffect) throws Exception {
   try {
       emotionMutex.acquire();
       // ... existing emotion processing (map affect, add to reservoir) ...
   } catch (Exception e) {
   } finally {
       emotionMutex.release();
   }

   DetectPathology(); // NEW: runs after every emotion
}
```

The same pattern is applied to `AddPositiveEmotion()` and `AddNegativeEmotion()`.

S10.4 Data Flow

The complete data flow from emotion stimulus to pathology result:

```
Program.java: neonAgent.AddEmotionDirect(AffectEnum.RAGE)
 → emotionMutex.acquire()
 → Post MapAffectEvent → MapAffectPlan resolves level
 → Post AddEmotionEvent → AddEmotionPlan writes reservoir
 → LevelCursorPlan → LevelCursorPlan → AddAffectHistoryEvent
   → AffectHistory beliefset updated
 → emotionMutex.release()
 → DetectPathology()
   → GetAffectHistoryItems() — retrieves full history
   → AffectDynamicsCalculator.Compute(history)
     → 20 parameters computed
   → For each window position:
     → For each of 40 motifs:
       → EmotionCategory.GetPrimaryCategory() on each event
       → PathologyMotif.Matches(window, localParams, thresholds)
   → CompoundMotif detection on matched set
   → scoreSeverity() using MBTI baseline
   → computeTier2Route() with grip-state detection
   → Return PathologyResult
     → Stored in lastPathologyResult
```

S11. End-to-End Trace — RAGE Followed by GRIEF

This section traces the pathology detection for a scenario where an INTP agent receives RAGE followed by GRIEF. This builds on the end-to-end trace in Section 11 of the companion paper.

Step 1 *[First emotion]* — `neonAgent.AddEmotionDirect(AffectEnum.RAGE)` processes as described in the companion paper's Section 11. The reservoir is set to -60.0, the Level 6 negative cursor fires, and the behavior pipeline selects "BreakObjectPlan". After `emotionMutex.release()`, `DetectPathology()` runs.

Step 2 *[First detection]* — The affect history contains a single entry: `{iType=RAGE, dValue=-60.0, time=T1}`. With `history.size() < 3`, the detector returns `PathologyResult` with `SEVERITY_NONE`. No motifs can match with only one event.

Step 3 *[Second emotion]* — `neonAgent.AddEmotionDirect(AffectEnum.GRIEF)` is called. GRIEF maps to timeline 1, level -5 (for an INTP). The reservoir absorbs the new stimulus. After processing, `DetectPathology()` runs again.

Step 4 *[Second detection]* — The affect history now has two entries: - `{iType=RAGE, dValue=-60.0, time=T1}` → category `NEG_STRONG` - `{iType=GRIEF, dValue=-50.0, time=T2}` → category `NEG_DEEP`

With only 2 events, only 2-length motifs can match. The detector checks all motifs and finds: - **E03** (Mild-to-extreme eruption): requires `NEG_MILD → NEG_DEEP` — no match (first event is NEG_STRONG, not NEG_MILD). - **E04** (Neutral explosion): requires `NEUTRAL → NEG_DEEP` — no match.

No 2-length motif matches the sequence NEG_STRONG → NEG_DEEP. Result: `SEVERITY_NONE`.

Step 5 *[Third emotion]* — `neonAgent.AddEmotionDirect(AffectEnum.DESPAIR)` is called. DESPAIR maps to NEG_DEEP.

Step 6 *[Third detection — match found]* — The affect history now has three entries: - `{RAGE, -60.0, T1}` → NEG_STRONG - `{GRIEF, -50.0, T2}` → NEG_DEEP - `{DESPAIR, -55.0, T3}` → NEG_DEEP

The detector finds two matches: - At `windowStart=0`: the window `[NEG_STRONG, NEG_DEEP, NEG_DEEP]` does not match D02 (`NEG_MOD → NEG_STRONG → NEG_DEEP`) — the first element differs. - At `windowStart=1`: the window `[NEG_DEEP, NEG_DEEP]` — no 2-length motif matches this exact pair. But extending back: `[NEG_STRONG, NEG_DEEP, NEG_DEEP]` is checked as a 3-length window.

However, motif **F01** (Deep negative fixation: `NEG_DEEP → NEG_DEEP → NEG_DEEP`) checks at `windowStart=0`: `[NEG_STRONG, NEG_DEEP, NEG_DEEP]` — does not match because the first element is NEG_STRONG, not NEG_DEEP.

Now consider a fourth emotion: `AddEmotionDirect(AffectEnum.DEPRESSED)`: - History: `[NEG_STRONG, NEG_DEEP, NEG_DEEP, NEG_DEEP]` - At `windowStart=1`: window `[NEG_DEEP, NEG_DEEP, NEG_DEEP]` matches **F01** (Deep negative fixation). Severity scored: NEG_DEEP appears in the descent, so +2. The INTP baseline deviation score is computed. If the overall dynamics show high inertia and low entropy (fixated in a single deep state), the deviation score exceeds 1.0, adding +1. Total score = 3 → **SEVERITY_SEVERE**.

Step 7 *[Compound detection]* — If D02 was also matched earlier in the history (e.g., from an earlier sequence of NEG_MOD → NEG_STRONG → NEG_DEEP), compound **C01** (Depressive spiral: D02 + F01) would fire, adding +1 to overall severity, yielding SEVERITY_CRITICAL.

Step 8 *[Tier 2 routing]* — Compound C01 has pathology tag "DEPRESSION". The route is `"INTP_DEPRESSION"`. The INTP's inferior function is Fe. If `valenceAsymmetry < -0.5` (strong negative bias), the grip flag is set, producing route `"INTP_DEPRESSION"` (grip does not override compound route).

Step 9 *[Result available]* — `lastPathologyResult` is updated. Any agent or plan can call `GetLastPathologyResult()` to access the severity level and route string.

S12. Build and Deployment

S12.1 Compilation

The eight new Java files are compiled outside the JDE using `javac`:

```
javac -cp _prebuild -d _prebuild EmotionCategory.java
AffectDynamicsParams.java
   AffectDynamicsCalculator.java PathologyMotif.java
PathologyMotifLibrary.java
   PathologyResult.java MBTIBaselines.java PathologyDetector.java
```

This produces 12 `.class` files (including inner classes):

Class File	Source
EmotionCategory.class	EmotionCategory.java
AffectDynamicsParams.class	AffectDynamicsParams.java
AffectDynamicsCalculator.class	AffectDynamicsCalculator.java
PathologyMotif.class	PathologyMotif.java
PathologyMotifLibrary.class	PathologyMotifLibrary.java
PathologyMotifLibrary$CompoundMotif.class	PathologyMotifLibrary.java
PathologyResult.class	PathologyResult.java
PathologyResult$MotifMatch.class	PathologyResult.java
PathologyResult$CompoundMatch.class	PathologyResult.java
MBTIBaselines.class	MBTIBaselines.java
MBTIBaselines$Baseline.class	MBTIBaselines.java
PathologyDetector.class	PathologyDetector.java

S12.2 Deployment

The `.class` files must be copied to `D:\projects\NeonAffect\NeonAffect\` — the directory that the JDE uses as its classpath for resolving types during `.agent` and `.plan` compilation:

```
copy _prebuild\NeonAffect\*.class NeonAffect\
```

The JDE then compiles `NeonAffectAgent.agent` (which references `PathologyResult`, `AffectDynamicsParams`, and `PathologyDetector`) against these pre-built `.class` files.

This follows the same deployment pattern used by the existing plain Java classes (`AffectHistoryVar.class`, `BehaviorItem.class`, `AffectEnum.class`, etc.) described in the companion paper.

S13. Summary of New Beliefsets, Events, and Methods

S13.1 New Data Classes

Class	Fields	Purpose
EmotionCategory	static maps	Maps 155 AffectEnum → 15 categories
AffectDynamicsParams	20 double fields	Computed dynamics snapshot
PathologyMotif	id, family, sequence, qualifiers	Single motif pattern
PathologyMotifLibrary	static ArrayList of motifs	All 40 + 15 patterns
MBTIBaselines	static Map of Baseline	Per-type parameter norms
PathologyResult	matches, compounds, dynamics, severity, route	Detection output
PathologyDetector	static methods	Main detection engine

S13.2 New NeonAffectAgent Fields

Field	Type	Purpose
lastPathologyResult	PathologyResult	Most recent detection result
lastDynamicsParams	AffectDynamicsParams	Most recent dynamics snapshot

S13.3 New NeonAffectAgent Methods

Method	Returns	Trigger
DetectPathology()	PathologyResult	Called automatically after every emotion
GetLastPathologyResult()	PathologyResult	On-demand by any agent or plan
ComputeCurrentDynamics()	AffectDynamicsParams	On-demand (no motif matching)

S14. Conclusions

The Pathology Detection Engine adds a clinically grounded analytical layer to the Neon-Affect system without altering any existing behavior. Several design decisions are worth noting:

Personality-relative thresholds: What constitutes "high velocity" or "high volatility" varies by MBTI type. An ESFP transitioning quickly between emotional states is normal; an ISTJ doing the same may indicate pathology. The per-type baselines ensure that detection sensitivity is calibrated to personality.

Two-tier separation: Tier 1 motifs are intentionally abstract (3-step category sequences). This keeps the motif library small (40 patterns) while covering the full space of clinically significant transitions. Tier 2 routing adds MBTI specificity *after* detection, rather than multiplying the motif count by 16 personality types.

Non-invasive integration: The detector reads the existing `affectHistoryItems` list without modification. It writes nothing to beliefsets, posts no events, and alters no reservoir values. The `PathologyResult` is stored in a field and available for query — it is up to downstream consumers

(future plans, external systems) to decide how to respond.

Grip-state awareness: The inferior function grip detection adds a layer of personality theory not present in the original system. When an INTP shows uncharacteristic emotional flooding (inferior Fe), or an INTJ shows impulsive sensation-seeking (inferior Se), the grip flag modifies the Tier 2 route to indicate that the agent may be operating outside its normal cognitive mode.

Compound motifs as clinical patterns: The 15 compound motifs map directly to recognized clinical categories (DEPRESSION, ANXIETY, BORDERLINE, TRAUMA, etc.). While the system does not make clinical diagnoses, these labels provide a vocabulary for interpreting complex multi-motif patterns that would otherwise be difficult to characterize.

The pathology detection engine transforms Neon-Affect from a system that *responds to* individual emotions into one that *understands* emotional trajectories. Combined with the existing OpenAI-guided behavior selection and CoJACK cognitive competition, this creates a three-layer architecture: CoJACK handles moment-to-moment dynamics, OpenAI handles contextual reasoning, and the pathology engine handles longitudinal pattern recognition.

This supplement was generated from direct source code analysis of the tenbase2-com-LLC/Neon-Affect repository. All code excerpts are taken directly from the repository source files.