Resolve 2.0: Agentic Transaction Intelligence & Categorisation Architecture

Status: Implementation Guide

Target System: Replit / FastAPI / Next.js / Supabase

Architecture: Two-Brain (Deterministic Solver + Probabilistic Agent)

1. Executive Critique & Immediate Remediation

Current State Analysis:

The current implementation of Resolve 2.0 is functionally broken regarding its agentic promises. While Ntropy is enriching merchant names, the system falls back to the "Resolve Unfinished Categorisation Tables" (static regex keywords) for categorization. This is brittle and fails on composite merchants (e.g., "Tesco" matches both Groceries and Mobile).

Critical Failure Point:

The "Context Hunter" (Layer 2) is never triggered. The background-sync.ts script fails to retrieve or pass the nylas_grant_id to the enrichment service. Without this token, the system cannot access user emails to find receipts, rendering the "Mindee" OCR layer inaccessible.

Step 1: The Hotfix (Apply Immediately)

Objective: Reconnect the "Eyes" of the Agentic Brain.

Action: Modify background-sync.ts.

1. **Fetch Grant ID:** Before the enrichment call, query supabase.from('user_tokens').select('nylas_grant_id').eq('user_id', userId).
2. **Inject Payload:** Update the payload sent to the Python service to include this ID.

TypeScript
```
// background-sync.ts - CORRECTED PAYLOAD
const payload = {
  transactions: raw_transactions,
  user_id: user.id,
  // CRITICAL FIX: Pass the grant ID so Layer 2 can function
  nylas_grant_id: user_tokens.nylas_grant_id,
  account_id: account.id
};
```

---

2. The New Architecture: Confidence-Gated Cascade

We are moving away from flat categorization to a **waterfall model**. A transaction only flows to the next (more expensive) layer if the current layer's **Confidence Score (C)** is below the **Acceptance Threshold (T_{accept})**.

The Golden Rule: $T_{accept} = 0.90$ (90% Confidence).

The Layer Stack

- **Layer 0: The Math Brain (Deterministic)** - Ghost Pair Protocol (Transfers).
- **Layer 1: The Classifier (Probabilistic)** - Ntropy Enrichment & Label Mapping.
- **Layer 2: The Context Hunter (Forensic)** - Nylas (Email) + Mindee (OCR).
- **Layer 3: The Sherlock Engine (Reasoning)** - Claude 4.5 + Web Search.

---

3. Implementation Specification (Layer by Layer)

Layer 0: The Deterministic Guardrails ("Ghost Pair Protocol")

Trigger: All incoming transactions.

Logic: Internal reconciliation. Identifying transfers between user accounts to prevent them from counting as "Spending" or "Income."

Logic Specification:

1. **Scan Window:** ± 3 days from transaction date.
2. **Match Condition:** Look for Tx_B where $Tx_B.amount = -1 \times Tx_A.amount$.
3. **Action:** If match found:
    - Set Category: transfer
    - Set Confidence: 1.0 (Absolute Truth)
    - **STOP PROCESSING.**

Layer 1: Enrichment & Mapping (Ntropy)

Trigger: Transactions not caught by Layer 0.

Logic: Use Ntropy to normalize the merchant and assign a base label. Crucially, we must discard the static "Categorisation Tables" regex and instead map Ntropy's labels to the Resolve taxonomy.

Confidence Formula (C_1):

$$C_1 = Ntropy\_Score \times Penalty_{ambiguity}$$
Ambiguity Penalties:

- **Merchant "Amazon", "PayPal", "eBay"**: $Penalty = 0.5$ (Forces Layer 2).
- **Label "General Merchandise"**: $Penalty = 0.6$ (Too vague).
- **Label "Utilities"**: $Penalty = 1.0$ (High trust).

Mapping Strategy (Refined based on Research):

- **Ntropy earned income** $\rightarrow$ Resolve income

- **Ntropy government benefits** $\rightarrow$ Resolve benefits
- **Ntropy rent** $\rightarrow$ Resolve rent (Watch out: "Benham & Reeves" might be labeled 'real estate'—requires Layer 3 if ambiguous).
- **Ntropy gas station** $\rightarrow$ Resolve transport (Fuel).

Action:

- If $C_1 \ge 0.90$: Commit Category. **STOP.**
- If $C_1 < 0.90$: **Proceed to Layer 2.**

Layer 2: The Context Hunter (Forensic Evidence)

Trigger: Ambiguous merchant (Amazon) or low confidence score.

Logic: Find "Level 3" data (Line Items) to prove the category.

Workflow:

1. **Nylas Search:** Query user email: from:[Merchant] amount:[Amount] date:.
2. **Mindee OCR:** If email contains PDF/Image, parse with Mindee Receipt API.
3. **Extraction:** Get line items.
    - *Example:* Amazon receipt shows "Pampers". Keyword "Pampers" maps to childcare.
    - *Example:* Amazon receipt shows "Sony Headphones". Maps to shopping -> electronics.

Confidence Update (C_2):

- **Golden Record:** If Receipt Amount == Transaction Amount $\rightarrow C_2 = 0.98$.
- **Partial Match:** Receipt found but amount varies (tips/fees) $\rightarrow C_2 = 0.85$.
- **No Match:** $C_2 = C_1$ (No new info).

Action:

- If $C_2 \ge 0.90$: Commit Category. **STOP.**
- If $C_2 < 0.90$: **Proceed to Layer 3.**

Layer 3: The Sherlock Engine (Neuro-Symbolic Reasoning)

Trigger: Transactions that remain opaque (e.g., "PXT*traviblycom", "HELENA BARRACLOUGH").

Logic: Pure reasoning using Claude 4.5 Sonnet + Web Search Tool.

System Prompt Strategy:

> Role: You are the Chief Financial Investigator.
>
> Task: Categorize a transaction based on fragment data.

Tools: You have access to Google Search.

Input: "PXT*traviblycom", £450.00.

Process:

1. Search the descriptor string.
2. Result: "Travibly is a boutique travel agency."
3. Reasoning: "Merchant is travel agency. Amount is high (£450). Likely a holiday booking."
4. **Output:** Category travel. Confidence 0.85. Reasoning: "Identified merchant as travel agency via web search."

---

4. Development Guide for Replit (Step-by-Step)

Copy and paste the following prompt into your Replit agent to execute this architectural shift.

Prompt for Replit Agent:

We are upgrading the "Resolve" transaction enrichment engine to a "Confidence-Gated Cascade".

Please implement the following changes in server/services/enrichment.py and background-sync.ts.

1. Database Schema Update

Run a migration to add these columns to the transactions table:

- confidence_score (float)
- enrichment_source (string: 'math_brain', 'ntropy', 'context_hunter', 'sherlock')
- reasoning_trace (jsonb) - To store the "Why?" (e.g., "Found receipt for diapers").

2. Fix the Background Sync

In background-sync.ts, modify the syncTransactions function.

- You MUST query the user_tokens table to get the nylas_grant_id.
- Pass this ID in the POST body to the python enrichment service.

3. Implement the Categorization Logic (Python)

Refactor the enrich_transaction function to follow this waterfall:

Layer 0: Ghost Pair (Math Brain)

- Check for internal transfers (same amount, opposite sign, +/- 3 days).
- If found: Category = 'transfer', Confidence = 1.0, Source = 'math_brain'. RETURN.

Layer 1: Ntropy (Classifier)

- Call Ntropy API.

- Map labels to our Resolve UK Categories.
- Calculate confidence = ntropy.confidence.
- APPLY PENALTIES:
    - If merchant is 'Amazon', 'PayPal', 'Tesco' (ambiguous), cap confidence at 0.5.
- If confidence > 0.9: Save & RETURN.

Layer 2: Context Hunter (Nylas/Mindee)

- If nylas_grant_id is present:
    - Search email for receipt using Merchant Name + Amount + Date.
    - If PDF found, send to Mindee.
    - If Line Items found (e.g., "Baby Wipes"), map to Category (e.g., 'family').
    - Set Confidence = 0.98.
    - Update reasoning_trace: "Receipt found: [Item List]".
    - Save & RETURN.

Layer 3: Sherlock (Claude)

- If still uncategorized or low confidence:
    - Construct a prompt for Claude: "Investigate this transaction:. Use Web Search if needed."
    - Output JSON: { category, confidence, reasoning }.
    - Save & RETURN.

Final Fallback

- If all fails, mark as uncategorized with confidence 0.0.

5. Commercial & Strategic Alignment (CPTO Note)

- **Revenue Driver:** By accurately identifying "Subscriptions" (Layer 2) and "Recurring Bills," we power the **Budget Finder** module, which identifies "waste" to pay off debt. This is the "Hook" for the premium tier.
- **Trust:** The "Ghost Pair" protocol (Layer 0) prevents the common PFM error of double-counting credit card payments as spending. This builds the "Math, Not Magic" trust.
- **Efficiency:** By gating Layer 3 (Claude/Search), we reduce API costs. We only pay for expensive LLM/Search tokens on the 5-10% of transactions that actually need it (the "Long Tail"), while cheap deterministic rules handle the 90% routine traffic.