**AI Agent Architecture Document**

**1. Components of the Agent**

1. **Data Ingestion (Gmail Fetcher)**
   * Uses Gmail API to fetch the latest transaction emails.
   * Extracts Date, RawText, Merchant, Amount, and Transaction\_ID.
2. **Parser & Preprocessor**
   * Applies regex to identify amounts (Rs. 123.45) and merchants (from VPA or names).
   * Normalizes raw transaction text for downstream classification.
3. **Categorizer**
   * **Fine-tuned DistilBERT model**: predicts transaction category from the email raw text.
   * Confidence threshold applied; if uncertain → defaults to "Transaction".
   * This fulfills the requirement of integrating a fine-tuned model.
4. **Google Sheets Logger**
   * Uses Google Sheets API to log structured transactions into a sheet (Date, Merchant, Amount, RawText, Category, Transaction\_ID).
   * Skips duplicates using Transaction\_ID.

**2. Interaction Flow**

1. **Step 1:** Gmail API fetches latest 100 transaction emails.
2. **Step 2:** Parser extracts structured fields.
3. **Step 3:** Raw text is passed to fine-tuned DistilBERT.
4. **Step 4:** Category predicted (e.g., *Food, Travel, Online Shopping*).
   * If confidence < 0.6 → fallback = "Transaction".
5. **Step 5:** Data logged into Google Sheets with proper headers.



**Components**

**1. Gmail Authentication & Fetcher**

* **Responsibility: Obtain OAuth2 credentials, list message IDs, fetch message snippets or full payloads.**
* **Why: Direct ingestion from user mailbox provides real-world, continuously updating transaction data.**
* **Key libraries: google-auth-oauthlib, google-api-python-client.**

**2. Parser & Preprocessor**

* **Responsibility: Convert raw email text into structured fields:**
  + **Date (from internalDate or parsed from text)**
  + **Amount (regex for Rs., INR, variations with 1–2 decimals)**
  + **Merchant (VPA parsing to VPA ... or name extraction)**
  + **Transaction\_ID (UPI reference / ref no.)**
  + **RawText (cleaned snippet)**
* **Preprocessing: lowercasing, trimming, removing excessive whitespace, truncation for long texts.**
* **Why: Emails are noisy — robust regex + cleaning ensures consistent model inputs.**

**3. Merchant Dictionary (Rule-based overrides)**

* **Responsibility: Maintain a JSON/dict mapping of known merchant names → categories (e.g., GAUTAM BHUYAN: Stationery).**
* **When used: First check — if merchant is present, assign category immediately.**
* **Why: Local merchants and ambiguous VPAs often lack context; dictionary ensures high-precision classification for recurring merchants.**

**4. Fine-tuned DistilBERT Classifier**

* **Responsibility: Classify RawText into one of the expense categories when merchant dictionary does not match.**
* **Model choice & reasoning:**
  + **DistilBERT is chosen for its balance between performance and inference speed (lighter than BERT).**
  + **Fine-tuning allows specialization on domain-specific transaction text.**
  + **Produces probability/confidence scores enabling fallbacks and uncertainty handling.**
* **Integration: Load tokenizer + model from a local folder in src/models/bert\_finetuned/. Use softmax over logits to obtain class probabilities.**

**5. Postprocessing & Decision Logic**

* **Responsibility: Combine dictionary output and model output:**
  + **If dictionary provides mapping → use it.**
  + **Else use model prediction.**
  + **If model confidence < threshold → mark as Uncategorized or fallback label Transaction.**
  + **Save predicted label and confidence to final row.**
* **Why: Avoid blocking the pipeline with prompts; maintain deterministic logging.**

**6. Google Sheets Logger**

* **Responsibility: Append rows to a Google Sheet with headers: Date, Merchant, Amount, RawText, Category, Transaction\_ID. Skip duplicates based on Transaction\_ID.**
* **Why: Simple, shareable storage for demo and dashboards. Easy to screenshot for submission.**

**7. Monitoring & Logging**

* **Responsibility: Log model probabilities, parsing errors, API responses, and optionally push training metrics to wandb.**
* **Why: Debugging and reproducibility; critical during evaluation and fine-tuning.**

**8. Optional User Feedback Loop (future)**

* **Responsibility: Allow user to correct categories in the sheet or via a prompt; persist corrections to dictionary and/or a retraining queue.**
* **Why: Improves performance over time, especially for local merchants.**

**(See docs/architecture\_detailed.png for component-level flow.)**

**Interaction Flow (Step-by-step)**

1. **User grants access via OAuth2 consent screen. The agent stores token.pickle locally (secure).**
2. **Gmail Fetcher queries Gmail with q='transaction' and retrieves the latest N messages (N = 100 for demo).**
3. **Parser extracts snippet or payload, runs regex extractions:**
   * **Amount regex supports Rs., INR, decimals.**
   * **Merchant extraction focuses on to VPA <vpa> <NAME> on <date> patterns; falls back to VPA id or heuristics.**
4. **Merchant dictionary check: If merchant (uppercased & normalized) in map → assign category.**
5. **If no dictionary match: Preprocess RawText (clean\_text) and feed to DistilBERT tokenizer → model → softmax → predicted label + confidence.**
6. **Decision logic: If confidence < threshold (tunable) → assign Uncategorized or Transaction. Otherwise use predicted label.**
7. **Google Sheets writer batches rows and appends them (with dedupe by Transaction\_ID).**
8. **Monitoring logs probabilities and errors. User can review the sheet and optionally annotate corrections.**
9. **(Optional) Periodic retraining or incremental fine-tuning with corrected labels.**

**Models Used & Rationale**

**Baseline: TF-IDF + Naive Bayes**

* **Why included: Quick baseline, extremely fast, interpretable, and performed well on synthetic data (~94%).**
* **Where used: Development & quick sanity checks; fallback if transformer resources unavailable.**

**Fine-tuned DistilBERT**

* **Why chosen:**
  + **Robust to noisy, unstructured email snippets.**
  + **Can learn context (e.g., "paid to Domino's" → Food) where keyword rules miss nuances.**
  + **Lightweight relative to full BERT (faster inference on CPU/GPU).**
* **Fine-tuning target: Classify RawText → predefined category labels.**
* **Parameter-efficient options: Consider LoRA / PEFT in future to reduce model size and training cost.**

**Decision Points & Trade-offs**

* **Dictionary vs Model: Dictionary yields perfect precision for known merchants; model covers unseen/unstructured cases. Combination reduces user interruptions.**
* **Thresholding: High threshold avoids false positives but increases Uncategorized rate. For demo, we set a lower threshold so model outputs are visible; in production, tune threshold based on validation set.**
* **Storage: Google Sheets chosen for simplicity and demo screenshots. For production, use a proper DB + dashboarding tool.**
* **Model storage on GitHub: Large model weights (>100MB) must not be pushed; use Git LFS or external hosting (Hugging Face / Google Drive).**

**Failure Modes & Mitigations**

1. **Model outputs same label for most rows (class bias):**
   * **Mitigation: balance dataset, class weights, augment minority classes, merge similar labels.**
2. **Merchant names ambiguous or missing:**
   * **Mitigation: dictionary override + ask user for corrections later (feedback loop).**
3. **Parsing errors for unusual email formats:**
   * **Mitigation: log failures, expand regex, include full payload parsing when needed.**
4. **Large model files block repo push:**
   * **Mitigation: .gitignore large weights, host externally, or use Git LFS.**

**Appendix: File & Folder Mapping**

* **src/fetch\_emails.py — Gmail fetch + parsing + categorization**
* **src/update\_sheets.py — Google Sheets logging**
* **src/pipeline.py — Orchestrator**
* **src/merchant\_dictionary.py — Merchant → Category JSON**
* **src/models/bert\_finetuned/ — (local model checkpoint; not in GitHub)**
* **data/transactions.csv — training dataset**
* **docs/architecture\_highlevel.png — high-level flowchart**
* **docs/architecture\_detailed.png — detailed flowchart**

**Models Used**

1. **Baseline: Naive Bayes (TF-IDF)**
   * Lightweight model trained on synthetic + real dataset.
   * Used for quick prototyping and to establish baseline metrics (~94% accuracy on synthetic dataset).
2. **Final: Fine-Tuned DistilBERT**
   * Transformer model fine-tuned on ~950 transactions (synthetic + real).
   * Handles noisy Gmail text better than Naive Bayes.
   * Provides confidence scores.
   * Integrated into the agent pipeline for live categorization.

**Reasons for Design Choices**

* **Gmail API:** Direct, automated data ingestion → no manual CSV uploads.
* **Regex Parser:** Ensures extraction of amounts, merchants, and transaction IDs even from unstructured raw text.
* **DistilBERT (fine-tuned):**
  + More robust to real-world noisy, unstructured text.
  + Lightweight compared to full BERT → faster inference.
* **Google Sheets:**
  + Easy to monitor logged transactions.
  + Enables real-time dashboarding (can connect to Power BI, Looker, etc.).
* **Fallback Label ("Transaction"):** Keeps pipeline non-blocking and ensures every row gets a category, even if uncertain.

**Future Extensions**

* **Merchant Dictionary Memory:** Save user-defined categories for specific local merchants (canteen, stationery shop).
* **Continuous Learning:** Append user-corrected rows into dataset → retrain DistilBERT periodically.
* **Visualization:** Build Power BI dashboard on top of Google Sheet for expense trends.
* **Multi-Channel Ingestion:** Extend from Gmail to SMS or bank statements.