CSC490 Assignment A2 - Pipelines, Data and Infrastructure

Chris Jiang (Student ID: 1008109574) | Yihe Li (Student ID: 1009802331)
Qianwen Wang (Student ID: 1009985358) | Seonghyun Ban (Student ID: 1005127738)

February 21, 2026

1 Aspirational Datasets

1.1 Project Context

The core challenge identified in our project is that small business founders are repeatedly required to make complex accounting decisions that demand specialized knowledge, such as determining appropriate accounts, handling depreciation, managing accruals, and ensuring ledger correctness. Our system aims to remove this cognitive burden by automatically generating correct bookkeeping entries, asking for clarification only when uncertainty is high.

To achieve this, the system must rely on rich, structured, and semantically grounded datasets that capture real-world accounting behavior, transaction semantics, and expert decision-making patterns.

1.2 Aspirational Dataset Categories

We identify the following aspirational datasets as most critical to the success of the project.

1.2.1 Expert-Labeled Transaction to Journal Entry Dataset

An ideal dataset would consist of real-world small business transactions paired with expert-validated journal entries. Each record would include the raw transaction description, amount, date, vendor information, and the corresponding double-entry bookkeeping records created by professional accountants.

This dataset would directly support the core posting engine by enabling supervised learning of transaction-to-account mappings and debit/credit structures.

Desired Schema:

```
transaction_id         : UUID
date                   : DATE
raw_description        : VARCHAR(512)    -- bank statement text
normalized_description: VARCHAR(256)
amount                 : DECIMAL(12,2)
currency               : CHAR(3)         -- e.g. CAD, USD
vendor_name            : VARCHAR(256)
industry_tag           : VARCHAR(64)     -- e.g. 'e-commerce', 'consulting'
journal_lines          : ARRAY[{
    account_number : VARCHAR(10),
    account_name   : VARCHAR(128),
    debit_amount   : DECIMAL(12,2),
    credit_amount  : DECIMAL(12,2)
}]
```

```
confidence_label       : ENUM('high', 'medium', 'low')
annotator_id           : UUID            -- the accountant who labeled it
notes                  : TEXT            -- explanation of posting rationale
```

Ideal scale: 1M+ transactions across 20+ industries (retail, e-commerce, consulting, food service, freelancing, construction, SaaS, etc.).

1.2.2 Receipt and Invoice Corpus with Ground Truth Line Items

This dataset would contain receipts and invoices across a wide range of vendors, industries, and jurisdictions, with ground truth annotations for vendor name, line items, taxes, totals, payment methods, and purchase categories.

Such a dataset would enable high-accuracy document understanding and improve downstream posting decisions, especially for distinguishing between expenses, assets, and mixed transactions.

Desired Schema:

```
document_id      : UUID
image_path       : VARCHAR(512)
document_type    : ENUM('receipt', 'invoice', 'credit_note')
vendor_name      : VARCHAR(256)
vendor_address   : TEXT
date             : DATE
line_items       : ARRAY[{
    description  : VARCHAR(256),
    quantity     : DECIMAL(8,2),
    unit_price   : DECIMAL(12,2),
    amount       : DECIMAL(12,2),
    tax_amount   : DECIMAL(12,2),
    tax_type     : VARCHAR(16)     -- HST, GST, PST, VAT
}]
subtotal         : DECIMAL(12,2)
tax_total        : DECIMAL(12,2)
grand_total      : DECIMAL(12,2)
payment_method   : ENUM('cash', 'debit', 'credit', 'etransfer')
currency         : CHAR(3)
ocr_text         : TEXT            -- raw OCR output for training
```

Ideal scale: 50K+ annotated receipt/invoice images with Canadian (HST/GST/PST) and US tax breakdowns.

1.2.3 Chart of Accounts and Accounting Policy Dataset

An aspirational dataset would include standardized and industry-specific Charts of Accounts (CoA), along with associated accounting rules, constraints, and best practices. This includes mappings between transaction types and allowable account combinations, as well as rules governing depreciation, accruals, and tax treatment.

This dataset would support hard validation rules to ensure that generated journal entries are always legally and structurally valid.

Desired Schema:

```
-- Primary table: chart_of_accounts
coa_template_id : UUID
industry         : VARCHAR(64)
jurisdiction     : VARCHAR(32)     -- 'CA-ON', 'US-CA', etc.
account_number   : VARCHAR(10)
```

```
account_name     : VARCHAR(128)
account_type     : ENUM('Asset', 'Liability', 'Equity', 'Revenue', 'Expense')
sub_type         : VARCHAR(64)     -- e.g. 'Current Asset', 'Long-Term Liability'
normal_balance   : ENUM('Debit', 'Credit')
parent_account   : VARCHAR(10)     -- nullable, for hierarchy
is_header        : BOOLEAN         -- is this a grouping account?
tax_relevance    : VARCHAR(64)     -- 'CCA Class 50', 'deductible', etc.
description      : TEXT

-- Companion table: accounting_rules
rule_id          : UUID
rule_type        : ENUM('depreciation', 'accrual', 'tax_split', 'constraint')
cca_class        : INT             -- nullable, for Canadian depreciation
rate             : DECIMAL(5,4)    -- e.g. 0.30 for 30%
method           : ENUM('declining_balance', 'straight_line')
half_year_rule   : BOOLEAN
conditions       : TEXT            -- human-readable conditions
```

Ideal scale: 20+ industry templates with complete CRA CCA classes (1–56) and IRS MACRS schedules.

1.2.4 Historical Reconciliation and Correction Dataset

This dataset would capture historical reconciliation events, including mismatches between bank statements and ledger entries, user corrections, and accountant adjustments. Each correction would be linked to the original automated decision and the final resolved entry.

Access to such data would enable learning from mistakes, improving confidence estimation, and supporting active learning workflows.

Desired Schema:

```
correction_id           : UUID
transaction_id          : UUID   -- FK to original transaction
original_journal_lines  : JSONB  -- the AI-generated entry
corrected_journal_lines: JSONB  -- the human-corrected entry
correction_type         : ENUM('account_change', 'amount_fix', 'split_change',
                               'duplicate_removal', 'missing_entry', 'reversal')
corrected_by            : ENUM('user', 'accountant', 'system_reconciliation')
correction_reason       : TEXT
original_confidence     : DECIMAL(3,2)  -- model confidence at prediction
timestamp               : TIMESTAMP
reconciliation_source   : ENUM('bank_statement', 'manual_review', 'audit')
```

Ideal scale: 10K+ correction events across diverse users and business types, enabling active learning.

1.2.5 Business Context and Vendor Intelligence Dataset

An ideal dataset would capture metadata about businesses (industry, size, region) and vendors (subscription services, utilities, marketplaces, payroll providers). This information would allow the system to recognize recurring patterns and vendor-specific semantics, reducing ambiguity and the need for user clarification over time.

Desired Schema:

```
vendor_canonical_name   : VARCHAR(256)
aliases                 : ARRAY[VARCHAR(128)]
```

```
    -- bank statement variations, e.g. ['AMZN MKTP CA*', 'AMAZON.CA', 'AMZ*']
mcc_code               : CHAR(4)
mcc_description        : VARCHAR(256)
default_expense_account: VARCHAR(10)
common_split_pattern   : JSONB    -- e.g. {"expense": 0.87, "hst": 0.13}
is_subscription        : BOOLEAN
typical_frequency      : ENUM('one-time', 'weekly', 'monthly', 'annual')
typical_amount_range   : NUMRANGE -- e.g. [9.99, 49.99]
industry_category      : VARCHAR(64)
```

Ideal scale: 100K+ vendor entries covering major merchants, SaaS providers, utility companies, and payment processors across Canada and the US.

1.3 Desired Data Schemas

Across all aspirational datasets, we would ideally have access to the following schema elements:

- Unique transaction or document identifiers
- Raw textual descriptions and normalized representations
- Monetary amounts with currency and tax breakdowns
- Vendor and counterparty identifiers
- Timestamped events (transaction date, posting date, correction date)
- Ground truth journal entries with debit and credit accounts
- Confidence or uncertainty annotations (where applicable)
- Provenance metadata linking decisions to sources or rules

Together, these aspirational datasets would enable an end-to-end accounting automation system that is accurate, explainable, and capable of improving over time while maintaining strict correctness guarantees.

2 Reality Check

While Part One outlined ideal datasets, many of them are not directly accessible in practice due to privacy constraints, proprietary ownership, or the fact that such datasets do not exist publicly. In this section, we identify datasets that are realistically available, along with data that we will generate ourselves to bridge critical gaps.

2.1 Available and Feasible Datasets

Table 1: **Reality-check dataset inventory.** Public datasets support categorization and document understanding, while synthetic and human-labeled data address privacy and availability gaps for journal entry supervision.

Relevant Item	Description	Commentary
Transaction Categorization Dataset		

Continued on next page

Relevant Item	Description	Commentary
HuggingFace: mitulshah/transaction-categorization	Public dataset containing bank transaction descriptions paired with high-level spending categories. Includes merchant names, free-text descriptions, and category labels.	Supports the baseline transaction classification task and vendor normalization. This dataset is useful for training and evaluating the first-stage classifier before journal entry generation. Does not include debit/credit structure, so it cannot directly supervise posting.
CORD Receipt Dataset Clova AI OCR Dataset	Receipt images with annotated OCR text and labeled semantic fields such as vendor name, date, total amount, and tax.	Directly supports receipt and invoice understanding. Improves extraction quality for downstream accounting decisions, especially when transaction text alone is ambiguous.
SROIE / ICDAR Receipt Dataset Kaggle: SROIE Dataset	Scanned receipt images with OCR annotations and key information labels (company, date, address, total).	Provides additional layout diversity and noise conditions for testing robustness of receipt extraction models.
Multi-Layout Invoice Document Dataset (MIDD) MIDD Dataset Paper and Links	Annotated invoice documents with structured labels across multiple layouts and vendors.	Complements receipt datasets by covering invoices and accounts-payable–style documents. Useful for generalizing document schemas beyond simple receipts.
Merchant Category Code (MCC) Reference Lists Visa MCC Manual	Reference documentation mapping merchant category codes to merchant types and industries.	Provides weakly supervised signals for vendor intelligence and constraining likely account mappings (e.g., software subscriptions vs. meals vs. equipment).
Chart of Accounts Examples GnuCash Sample Chart of Accounts	Open-source examples of standard Charts of Accounts used in small business accounting.	Used to define canonical account schemas and enforce hard validation rules (e.g., legal account types and hierarchical structure).

Continued on next page

Relevant Item	Description	Commentary
Synthetic Transaction → Journal Entry Dataset (Generated) Internal dataset generated by team	Synthetic bank transactions paired with ground-truth journal entry templates, including debit/credit splits, taxes, and fees.	Necessary because real transaction-to-journal-entry datasets are not publicly available. Allows supervised training and controlled evaluation of the posting engine under known constraints.
Human-Labeled Clarification Dataset (Generated) Internal dataset generated by team	Small set of ambiguous transactions labeled with the correct clarification question and final resolved journal entry.	Directly evaluates the "Detect → Ask → Post" workflow. Used to assess whether the system asks minimal, appropriate questions only when needed.
CRA Capital Cost Allowance (CCA) Class Listings	canada.ca/.../cca/classes.html	Official CRA documentation of all CCA classes (1–56) with rates, asset types, half-year rule, and AIIP provisions. Structured as web pages with tables. Essential for the tax rules engine. We will scrape and structure into machine-readable JSON. Covers depreciation rules for equipment, vehicles, buildings, software, etc. that Autobook must apply automatically.

3 Data-processing pipelines

3.1 Pipeline Overview

Our system processes financial inputs into validated double-entry journal entries using a staged, idempotent pipeline implemented inside a single FastAPI backend service. The pipeline is executed synchronously at the application layer (with background execution for long-running stages), and all pipeline state is persisted in PostgreSQL for observability, retries, and UI polling.

Core stages.

1. **Ingestion:** accept bank CSV uploads and receipt/invoice files; persist immutable raw artifacts.
2. **Normalization:** canonicalize heterogeneous transaction schemas into a single internal schema.
3. **Document understanding:** extract structured fields from receipts/invoices (OCR + layout parsing).
4. **Posting inference:** map transaction (+ optional document context) → proposed journal entry.
5. **Validation + gating:** enforce hard accounting rules; auto-post only above confidence threshold.
6. **Persistence:** store raw/processed artifacts in an S3-backed data lake and write ledger views to Postgres.

Execution model. The pipeline is triggered via a direct API call on upload (request returns a `job_id` immediately), and actual processing is scheduled using FastAPI `BackgroundTasks`. For batch use cases (reprocessing, reconciliation), the same pipeline entrypoint is invoked by a scheduled runner (nightly) using the same code path.The service is exposed externally only through the load balancer, while all internal pipeline execution and database access occur within the private network.

3.2 Technologies Used (Implemented)

- **API + orchestration:** FastAPI routes drive ingestion and pipeline execution (`BackgroundTasks`).
- **Compute:** single container deployed to ECS Fargate; all stages run as functions inside one service.
- **Networking and access control:** the backend is deployed behind an HTTPS Application Load Balancer within a VPC. Application containers accept traffic only from the load balancer, and the PostgreSQL database is placed in private subnets, reachable only by the backend service.
- **Object store / data lake:** Amazon S3 (one bucket per environment) with logical prefixes: `raw/` for immutable inputs and `processed/` for intermediate artifacts.
- **Structured storage:** PostgreSQL (RDS) for application state, pipeline job tracking, clarification tasks, and ledger views.
- **Document extraction:** deterministic OCR/layout extraction (baseline) producing structured document objects.

- **Posting inference:** LLM-based inference via Amazon Bedrock; document structure is attached as context.

3.3 Data Ingestion

3.3.1 Inputs

- **Bank transactions:** CSV exports (Stripe, major banks).
- **Documents:** receipt/invoice uploads (JPG, PNG, PDF).

3.3.2 Ingestion Process

Uploads follow a single contract: `POST /api/upload` sends file bytes to the FastAPI backend, which writes the artifact to S3 server-side. This avoids presigned URLs and S3 CORS complexity. Each upload results in a new pipeline job row in Postgres and returns a `job_id` for status tracking and UI polling.

Raw artifact persistence (immutable). All uploaded files are stored *unchanged* in:

- `s3://{bucket}/raw/{user_id}/{upload_id}/...`

Metadata (content type, timestamp, user id, source system) is stored in Postgres and linked to the pipeline job.

Implementation mapping.

- API entrypoints: `src/backend/routes/upload.py`, `src/backend/routes/pipeline.py`
- Storage adapters: `src/common/io/s3.py`, `src/common/io/local_fs.py`
- IDs/config/logging: `src/common/ids.py`, `src/common/config.py`, `src/common/logging.py`

3.4 Canonical Schemas

This section defines the key internal schemas required for a maintainable pipeline: (1) normalized transactions, (2) parsed documents, (3) journal proposals, and (4) pipeline job state.

3.4.1 Canonical Transaction Schema

Each ingested transaction is normalized into:

- `transaction_id` (UUID)
- `raw_description` (string)
- `normalized_description` (string)
- `amount` (decimal)
- `currency` (ISO-4217 string)
- `transaction_date` (date)
- `counterparty` (string, optional)
- `source_system` (string)

Implementation mapping.

- Schema definitions: `src/common/schemas/transactions.py`
- Normalization job: `src/jobs/normalize_transactions/job.py`

3.4.2 Structured Document Schema (Receipt/Invoice)

Document understanding produces a structured object:

- `document_id` (UUID)
- `document_type` (`receipt`|`invoice`|`other`)
- `vendor_name` (string, optional)
- `document_date` (date, optional)
- `subtotal` (decimal, optional)
- `tax_total` (decimal, optional)
- `grand_total` (decimal, optional)
- `currency` (string, optional)
- `line_items` (list, optional: description, qty, unit price, line total, tax)
- `ocr_text` (string, optional for debugging/tracing)
- `artifact_refs`: S3 keys for `raw` input and `processed` extraction output

Implementation mapping.

- Schema definitions: `src/common/schemas/documents.py`
- Parsing job: `src/jobs/parse_documents/job.py`

3.4.3 Journal Proposal Schema (Intermediate Representation)

Posting inference produces a proposal (never directly writes the ledger):

- `proposal_id` (UUID)
- `transaction_id` (UUID)
- `document_id` (UUID, optional)
- `journal_lines`: list of {account_code, account_name, debit, credit, memo(optional)}
- `splits`: optional structured splits (tax, fees, mixed purchases)
- `confidence` (float in [0,1])
- `rationale`: short text justification (stored for audit/debug)

Implementation mapping.

- Schema definitions: `src/common/schemas/journal.py`
- Inference service: `src/services_ml/posting_inference/app.py`
- Optional enrichment: `src/services_ml/feature_enrichment/app.py`

3.4.4 Pipeline Job State Schema (Postgres)

All pipeline runs are tracked in Postgres to support retries, UI polling, and post-mortems. The `pipeline_jobs` table minimally contains:

- `job_id` (UUID, primary key)
- `user_id` (string/UUID)
- `created_at`, `updated_at` (timestamps)
- `status`: `queued`|`running`|`succeeded`|`failed`
- `stage`: `ingest`|`normalize`|`parse`|`infer`|`validate`|`commit`
- `input_refs`: references to S3 raw objects and/or transaction batch ids
- `output_refs`: references to S3 processed artifacts and proposal ids
- `error_code`, `error_message` (nullable)

Implementation mapping.

- Job tracking service: `src/backend/services/pipeline_state.py`
- Pipeline endpoints: `src/backend/routes/pipeline.py`

3.5 Normalization Stage

Bank exports vary widely in column names and formatting. The normalization job:

1. parses CSV using source-specific adapters,
2. aligns fields into the canonical schema,
3. normalizes merchant strings (trim, casing, token cleanup),
4. emits normalized records and stores them as `processed/transactions/...` artifacts in S3,
5. writes transaction rows (or batch metadata) into Postgres.

Idempotency. Normalization is keyed by `(upload_id, source_system)`. Re-runs overwrite only the derived `processed/` objects and do not mutate the raw artifact.

3.6 Document Understanding Stage

Receipt/invoice understanding produces structured fields used for posting and validation.

3.6.1 Steps

1. OCR/layout extraction to obtain a deterministic text+structure baseline.
2. Field extraction (vendor/date/totals/taxes), then optional line item detection.
3. Emit a structured document object and persist it under `processed/documents/...`.

Linking documents to transactions. Documents may be linked to one or more normalized transactions by heuristics (date/amount/vendor) and stored as foreign-key references in Postgres for downstream inference.

3.7 Posting Inference Stage

Posting inference maps a normalized transaction, optionally enriched with structured document context, to a proposed journal entry. The system performs inference via an LLM call (Bedrock) using a constrained prompt that requests a structured JSON output matching the `Journal Proposal Schema`.

3.7.1 Outputs

- debit/credit accounts (candidates resolved to the selected Chart of Accounts),
- explicit split structure (tax/fees),
- global confidence score,
- short rationale for audit/debug.

Intermediate-only. No ledger writes occur in this stage. The proposal is stored and passed to validation.

3.8 Validation and Confidence Gating

Validation enforces deterministic constraints before committing any entry:

3.8.1 Hard Validation Rules

- Debits must equal credits.
- Accounts must exist and be legal under the selected Chart of Accounts.
- Tax splits must be present when applicable (e.g., GST/HST patterns), and totals must reconcile.

3.8.2 Confidence Gate and Clarification Workflow

If `confidence` $\geq \tau$, the proposal is eligible for auto-post. Otherwise, a clarification task is created.

Clarification storage. Clarifications are stored in a Postgres table (not a queue), making them queryable and joinable with pipeline jobs and ledger entries. The UI polls `GET /api/clarifications` and resolves tasks via `POST /api/clarify`.

Implementation mapping.

- Validator job: `src/jobs/validate_ledger/job.py`
- Clarification endpoints: `src/backend/routes/clarify.py`

3.9 Persistence and Storage Design

We persist both immutable artifacts (for auditability) and structured views (for application queries).

3.9.1 S3 Data Lake Layout

- `raw/`: immutable uploaded inputs (CSVs, images, PDFs).
- `processed/`: derived artifacts such as normalized transactions, parsed documents, model proposals.

Artifacts are written with provenance metadata (job id, stage, timestamps, version hash) in either object metadata or sidecar JSON. Lifecycle rules can expire older `processed/` artifacts while retaining `raw/`.

3.9.2 Postgres Application Storage

Postgres stores:

- pipeline job tracking (`pipeline_jobs`),
- clarification tasks (`clarifications`),
- ledger tables / journal entry views for the app UI (`journal_entries`, `journal_lines`),
- audit trail linking back to S3 raw and processed artifacts.

3.10 When Pipelines Run and Use Cases

3.10.1 Event-driven (primary)

Triggered on upload:

- `POST /api/upload` → create `job_id` → schedule BackgroundTasks pipeline run
- UI polls `/api/pipeline/{job_id}` for status

3.10.2 Batch (secondary)

Nightly runs for:

- reprocessing with improved parsers/prompts,
- reconciliation and consistency checks,
- backfills after schema changes.

3.11 Pipeline Architecture Diagram

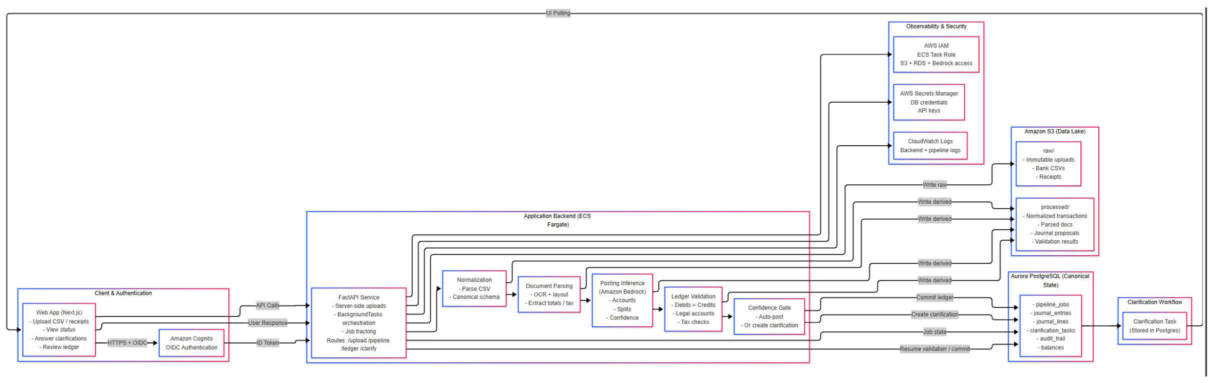

Figure 1: End-to-end data-processing pipeline for the AI Accountant system.The FastAPI back-end runs behind an HTTPS load balancer, persists artifacts in S3, and stores canonical state in a private PostgreSQL database.

3.12 Initial Implementation Status

The current implementation delivers an end-to-end runnable pipeline with:

- server-side uploads into S3 with Postgres job tracking,
- transaction normalization job producing canonical transaction records,
- document parsing job producing structured receipt/invoice objects,
- LLM-based posting inference producing structured journal proposals,
- deterministic ledger validation + confidence gating,
- clarification task creation + API endpoints for user resolution.

3.13 Next Steps

Planned extensions (not yet implemented) include:

- tighter transaction–document matching (learned matcher or stronger heuristics),
- reconciliation against bank statements and duplicate detection,
- multi-page invoice support with robust page stitching,
- active learning from user corrections (prompt refinement or supervised model later),
- improved observability (structured logs + metrics dashboards) using pipeline job traces.

4 Infrastructure as Code Implementation

Please checkout link to github repo for Implementation

5 Disaster Recovery Demonstration

Demo: Demo