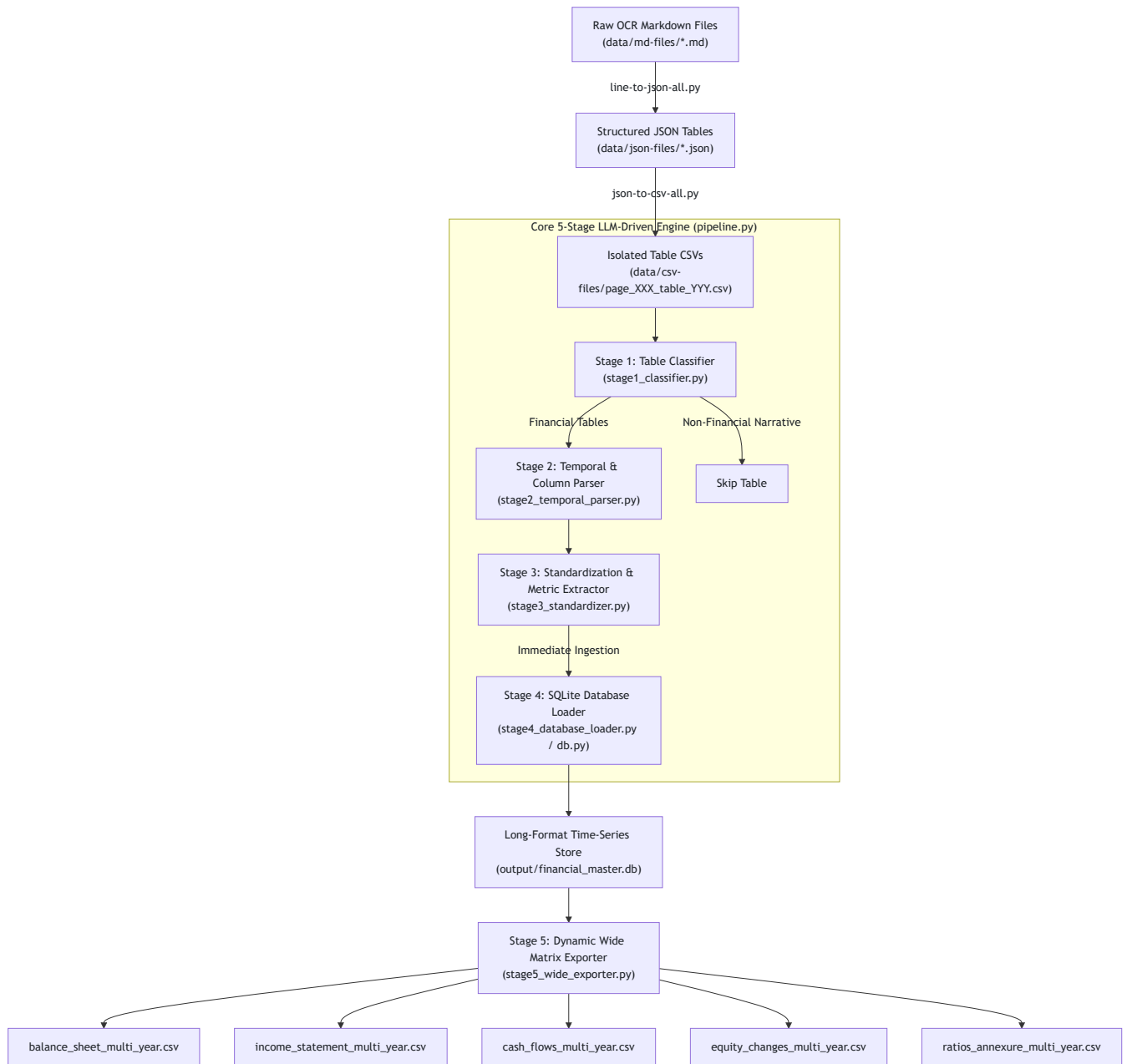


End-to-End Multi-Year Financial Data Extraction Pipeline

This document provides a comprehensive technical walkthrough of the automated pipeline built to extract, classify, standardize, harmonize, and export multi-year financial statements and footnote schedules from scanned/OCR annual report tables.

1. High-Level Architecture & Workflow

The architecture transitions unstructured, fragmented OCR markdown tables across diverse annual reports into an **auditable, long-format time-series SQLite database** and **harmonized wide multi-year matrix CSV files** (with Fiscal Years across columns and financial line items down rows).



2. Step-by-Step Data Flow & Examples

Below is a detailed breakdown of each processing stage, showing the **exact input data format**, the **transformation logic**, and the **resulting output data format**.

Step 0A: Markdown Line Parser (`line-to-json-all.py`)

- **Purpose:** Converts line-by-line markdown tables with alignment rows (`|---|---`) and text wrappers into structured JSON table objects.
- **Script:** [`line-to-json-all.py`](file:///d:/Work/Research/OCR%20VLLM/scripts/line-to-json-all.py)

Input (Markdown File: `data/md-files/page_009.md` **):**

Particulars	Note	30-Jun-2024 (BDT)	30-Jun-2023 (BDT)
Non-Current Assets			
Property, Plant and Equipment	4.00	38,512,120,450	35,110,450,200
Capital Work-in-Progress	5.00	2,140,500,000	1,850,200,000

Output (JSON File: `data/json-files/page_009.json` **):**

```
{
  "page": 9,
  "tables": [
    {
      "table_index": 1,
      "headers": ["Particulars", "Note", "30-Jun-2024 (BDT)", "30-Jun-2023 (BDT)"],
      "rows": [
        ["Non-Current Assets", "", "", ""],
        ["Property, Plant and Equipment", "4.00", "38,512,120,450", "35,110,450,200"],
        ["Capital Work-in-Progress", "5.00", "2,140,500,000", "1,850,200,000"]
      ]
    }
  ]
}
```

Step 0B: Table CSV Isolator (`json-to-csv-all.py`)

- **Purpose:** Deconstructs JSON page files into individual CSV tables named `page_XXX_table_YYY.csv` so each table can be independently classified and parsed.
- **Script:** [`json-to-csv-all.py`](file:///d:/Work/Research/OCR%20VLLM/scripts/json-to-csv-all.py)

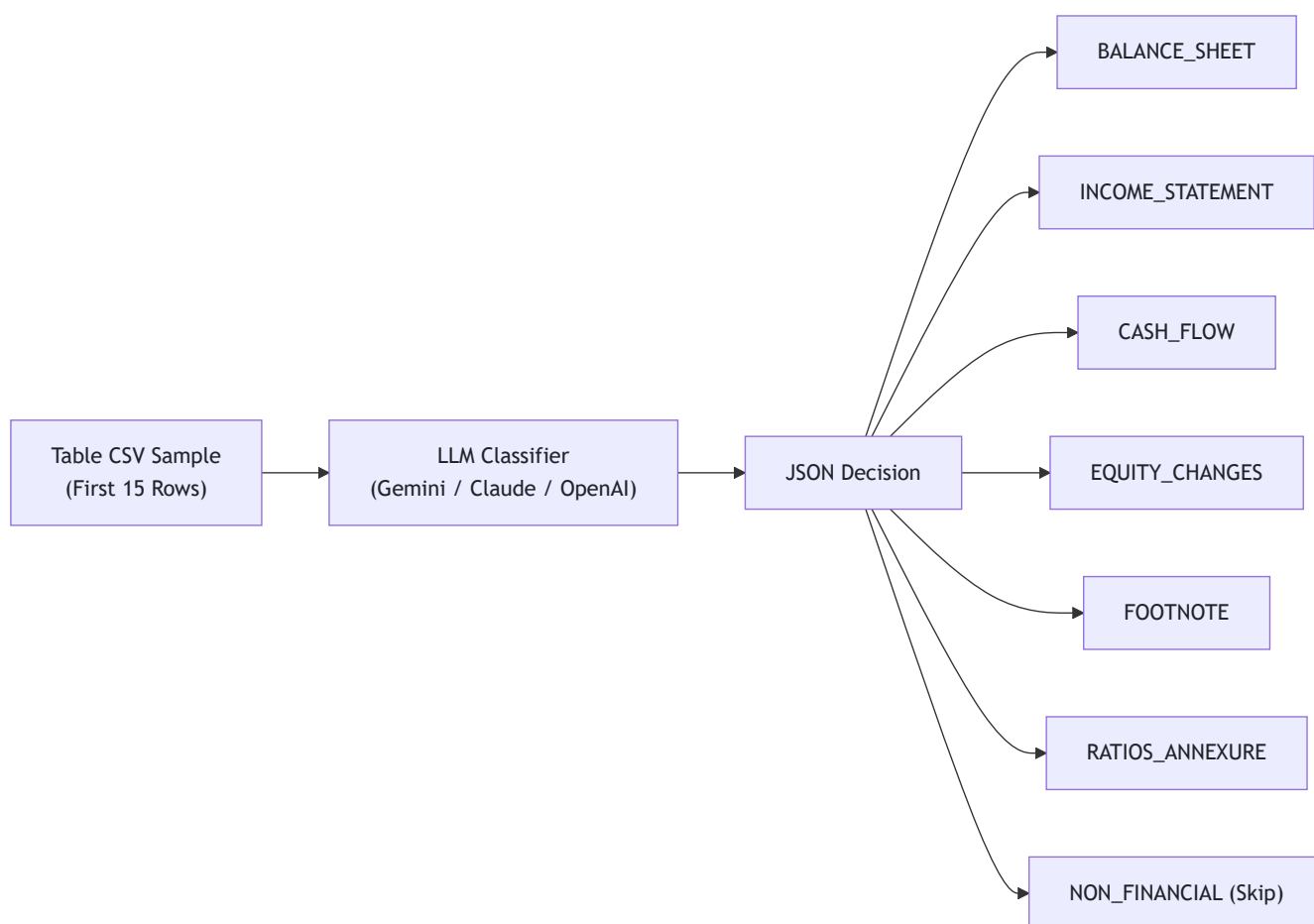
Output (CSV File: `data/csv-files/page_009_table_001.csv` **):**

Particulars	Note	30-Jun-2024 (BDT)	30-Jun-2023 (BDT)
Non-Current Assets			
Property, Plant and Equipment	4.00	38,512,120,450	35,110,450,200
Capital Work-in-Progress	5.00	2,140,500,000	1,850,200,000

Stage 1: LLM Table Classifier (`stage1_classifier.py`)

- **Purpose:** Identifies the exact accounting typology of the table using an LLM. Non-financial narrative tables (e.g., Key Audit Matters, risk descriptions) are skipped.

- **Script:** [stage1_classifier.py](file:///d:/Work/Research/OCR%20VLLM/planned-pipeline/stage1_classifier.py)



LLM Structured Output:

```
{
  "statement_type": "BALANCE_SHEET",
  "confidence": 1.0,
  "is_financial_statement": true,
  "reason": "Contains Statement of Financial Position line items (Non-Current Assets, Total Assets, Equity and Liabilities)."
```

Stage 2: Temporal & Column Header Parser (stage2_temporal_parser.py)

- **Purpose:** Resolves dynamic column headers, distinguishes line item descriptions and footnote references from value columns, detects reporting fiscal years (e.g. 2024 , 2023), and extracts period-end dates and restatement flags.
- **Script:** [stage2_temporal_parser.py](file:///d:/Work/Research/OCR%20VLLM/planned-pipeline/stage2_temporal_parser.py)

Input Header Array:

```
["Particulars", "Note", "30-Jun-2024 (BDT)", "30-Jun-2023 (Restated)"]
```

LLM Structured Output:

```
{
  "currency": "BDT",
  "columns": [
    {
      "index": 0,
      "column_type": "LINE_ITEM",
      "fiscal_year": null,
      "period_end": null,
      "is_restated": false
    },
    {
      "index": 1,
      "column_type": "NOTE_REF",
      "fiscal_year": null,
      "period_end": null,
      "is_restated": false
    },
    {
      "index": 2,
      "column_type": "YEAR_VALUE",
      "fiscal_year": 2024,
      "period_end": "2024-06-30",
      "is_restated": false
    },
    {
      "index": 3,
      "column_type": "YEAR_VALUE",
      "fiscal_year": 2023,
      "period_end": "2023-06-30",
      "is_restated": true
    }
  ]
}
```

Stage 3: Semantic Standardization & Metric Extraction

(stage3_standardizer.py)

- **Purpose:**

1. Preserves hierarchical context (e.g., active category `Non-Current Assets` remains attached to child items).

2. Cleans accounting numeric cell formats: commas (38,512,120,450 → 38512120450.0), negative parentheses ((1,250.00) → -1250.0), dashes/nils (- → 0.0).
3. Formats each row into an independent time-series observation.

- **Script:** [stage3_standardizer.py](file:///d:/Work/Research/OCR%20VLLM/planned-pipeline/stage3_standardizer.py)

Extracted Data Objects (FinancialMetricRecord):

```
# Record 1 (2024 Observation)
FinancialMetricRecord(
    statement_type="BALANCE_SHEET",
    category="Non-Current Assets",
    standard_field_name="Property, Plant and Equipment",
    raw_field_name="Property, Plant and Equipment",
    note_ref="4.00",
    fiscal_year=2024,
    period_end="2024-06-30",
    value=38512120450.0,
    unit="BDT",
    is_subtotal=False,
    is_total=False,
    is_restated=False,
    source_pdf="Annual_Report_2024.pdf",
    source_csv="page_009_table_001.csv"
)

# Record 2 (2023 Observation)
FinancialMetricRecord(
    statement_type="BALANCE_SHEET",
    category="Non-Current Assets",
    standard_field_name="Property, Plant and Equipment",
    raw_field_name="Property, Plant and Equipment",
    note_ref="4.00",
    fiscal_year=2023,
    period_end="2023-06-30",
    value=35110450200.0,
    unit="BDT",
    is_subtotal=False,
    is_total=False,
    is_restated=True,
    source_pdf="Annual_Report_2024.pdf",
    source_csv="page_009_table_001.csv"
)
```

Stage 4: Real-Time SQLite Database Ingestion (`stage4_database_loader.py` & `db.py`)

- **Purpose:** Persists metrics directly into `output/financial_master.db` immediately after each table is processed.
- **Deduplication / Restatement Policy:** If an earlier report had `2023` original figures, and a subsequent report provides `2023 Restated` figures, the restated figures are automatically prioritized upon export.
- **Script:** [`stage4_database_loader.py`](file:///d:/Work/Research/OCR%20VLLM/planned-pipeline/stage4_database_loader.py), [`db.py`](file:///d:/Work/Research/OCR%20VLLM/planned-pipeline/db.py)

SQLite Table Schema (`financial_metrics`):

```
CREATE TABLE IF NOT EXISTS financial_metrics (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    statement_type TEXT NOT NULL,  
    category TEXT,  
    standard_field_name TEXT NOT NULL,  
    raw_field_name TEXT,  
    note_ref TEXT,  
    fiscal_year INTEGER NOT NULL,  
    period_end TEXT,  
    value REAL NOT NULL,  
    unit TEXT DEFAULT 'BDT',  
    is_subtotal BOOLEAN DEFAULT 0,  
    is_total BOOLEAN DEFAULT 0,  
    is_restated BOOLEAN DEFAULT 0,  
    source_pdf TEXT,  
    source_csv TEXT,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    UNIQUE(statement_type, category, standard_field_name, fiscal_year, is_restated,  
    source_pdf)  
);
```

Stage 5: Dynamic Multi-Year Wide Matrix Exporter

(`stage5_wide_exporter.py`)

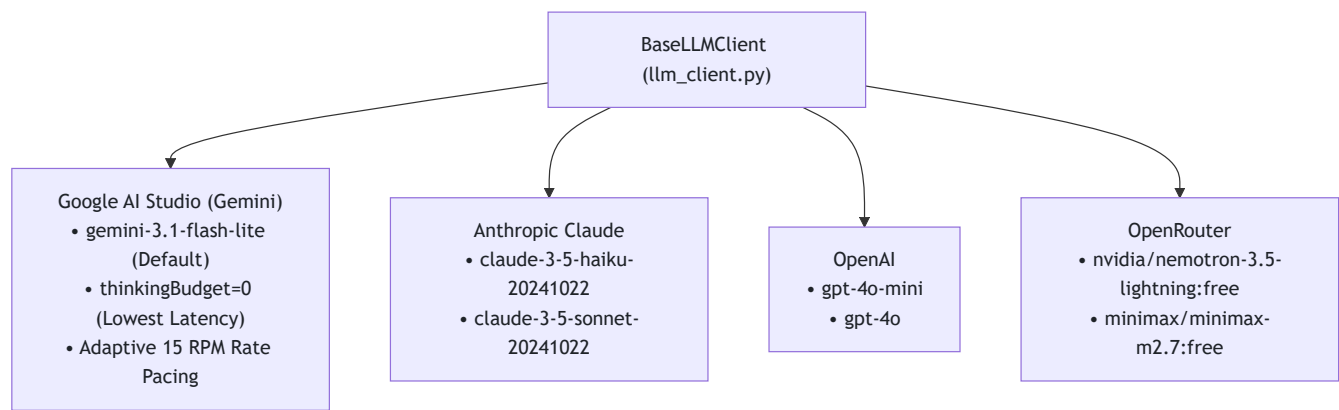
- **Purpose:** Queries the unified long-format SQLite database, finds all unique fiscal years across reports (e.g. `2021` , `2022` , `2023` , `2024`), pivots the data so that **Years are on Columns** and **Financial Line Items are on Rows**, and exports clean CSVs.
- **Script:** [`stage5_wide_exporter.py`](file:///d:/Work/Research/OCR%20VLLM/planned-pipeline/stage5_wide_exporter.py)

Output CSV Matrix (output/balance_sheet_multi_year.csv):

Statement_Type	Category	Particulars	Note_Ref	FY2023	FY2024
BALANCE_SHEET	Non-Current Assets	"Property, Plant and Equipment"		4.00,35110450200.0,38512120450.0	
BALANCE_SHEET	Non-Current Assets	"Capital Work-in-Progress"		5.00,1850200000.0,2140500000.0	
BALANCE_SHEET	Non-Current Assets	"Intangible Assets"		6.00,420100000.0,390000000.0	
BALANCE_SHEET	Current Assets	"Inventories"		7.00,12450800100.0,14120900000.0	
BALANCE_SHEET	Current Assets	"Trade and Other Receivables"		8.00,8540120000.0,9120400000.0	
BALANCE_SHEET	Current Assets	"Cash and Cash Equivalents"		9.00,1420500000.0,1890300000.0	
BALANCE_SHEET	Summary	"Total Assets"		,,59792170300.0,66174220450.0	

3. Supported Multi-Provider LLM Engine (llm_client.py)

The pipeline includes unified support for major LLM providers:



Key Capabilities:

- Deterministic JSON Output:** Forces structured JSON via provider-native schema parameters (`responseMimeType: "application/json"` , `response_format: {"type": "json_object"}`).
- Lowest Thinking Level:** Configures `thinkingBudget: 0` for Google AI Studio, eliminating unnecessary reasoning tokens to achieve ~1.1s latency per table.
- Adaptive Rate Pacing:** Automatically respects free-tier limits (e.g., 15 RPM on Google AI Studio) with exponential backoff and recovery.
- Instant Ingestion:** Database commits happen per-table in real time, preventing data loss on interruption.

4. Execution Commands

Running with Google AI Studio (Gemini Free Tier):

```
python "d:\Work\Research\OCR VLLM\planned-pipeline\pipeline.py" --provider GoogleAI --api_key "YOUR_GEMINI_API_KEY"
```

Running with Anthropic Claude:

```
python "d:\Work\Research\OCR VLLM\planned-pipeline\pipeline.py" --provider Claude --api_key "YOUR_ANTHROPIC_API_KEY"
```

Running with OpenAI:

```
python "d:\Work\Research\OCR VLLM\planned-pipeline\pipeline.py" --provider OpenAI --api_key "YOUR_OPENAI_API_KEY"
```

Running with OpenRouter:

```
python "d:\Work\Research\OCR VLLM\planned-pipeline\pipeline.py" --provider OpenRouter --api_key "YOUR_OPENROUTER_API_KEY"
```

5. Directory Structure Reference

```
d:\Work\Research\OCR VLLM\
├─ data/
│   ├── md-files/           # Source OCR Markdown pages (page_001.md - page_062.md)
│   ├── json-files/        # Parsed JSON table structures
│   ├── csv-files/         # 172 individual table CSVs (page_XXX_table_YYY.csv)
│   ├── financial-csv-only/ # Primary statements (Balance Sheet, Income, Cash Flow, Ratios)
│   ├── foot-notes-only/   # Footnote schedule CSVs (Notes 4.00 - 49.00)
│   └─ other-csv-only/     # Key Audit Matters and qualitative narrative tables
├─ planned-pipeline/
│   ├── config.py          # Centralized configuration, endpoints, models & defaults
│   ├── schema.py          # Data models (ColumnMetadata, FinancialMetricRecord)
│   ├── db.py              # SQLite schema, tables & deduplication logic
│   ├── llm_client.py       # Unified client for Google AI, Claude, OpenAI, OpenRouter
│   └─ stage1_classifier.py # Stage 1: LLM table classifier
```

```

|   |─ stage2_temporal_parser.py # Stage 2: LLM column & fiscal year parser
|   |─ stage3_standardizer.py  # Stage 3: Numeric cleaner & hierarchy tracker
|   |─ stage4_database_loader.py # Stage 4: Real-time SQLite loader
|   |─ stage5_wide_exporter.py # Stage 5: Multi-year wide CSV matrix generator
|   |─ pipeline.py              # Main CLI pipeline orchestrator
|
|   |─ output/
|   |   |─ financial_master.db          # Master SQLite time-series database
|   |   |─ balance_sheet_multi_year.csv # Wide Balance Sheet (Years across columns)
|   |   |─ income_statement_multi_year.csv # Wide Income Statement
|   |   |─ cash_flows_multi_year.csv    # Wide Statement of Cash Flows
|   |   |─ equity_changes_multi_year.csv # Wide Changes in Equity
|   |   |─ ratios_annexure_multi_year.csv # Wide Financial Ratios & Annexures
|   |
|   |─ runs/
|   |   |─ goggle-ai-run.txt            # Google AI Studio raw execution log
|   |   |─ google_ai_extraction_results.csv # Google AI 172-table extraction results
|   |   |─ google_ai_run_summary.md     # Google AI run summary report
|   |   |─ open-router-run.txt          # OpenRouter raw execution log
|   |   |─ openrouter_extraction_results.csv # OpenRouter 172-table extraction results
|   |   |─ openrouter_run_summary.md    # OpenRouter run summary report
|   |   |─ runs_comparison.csv          # Head-to-head comparison dataset
|   |   |─ runs_comparison_summary.md   # Comparative benchmark report
|   |
|   └─ scripts/
|       |─ line-to-json-all.py         # Batch Markdown to JSON parser
|       |─ json-to-csv-all.py         # Batch JSON to Table CSV parser
|       └─ sort_csv_files.py           # Categorizes CSVs into financial, footnote, and other
folders

```