LogForge-Apache – Local ETL Pipeline for Apache Access Logs

Objective

Design and build a production-simulated, CLI-based ETL pipeline that **parses**, **deduplicates**, **validates**, and **summarizes** Apache access logs. Data is stored in a **SQLite** database and used to generate daily reports for observability use cases like traffic monitoring, anomaly detection, and SRE auditing.

Background

Apache logs are a rich source of insights into application usage patterns, abuse attempts, or bottlenecks. By engineering this log-processing pipeline from scratch, the team will simulate internal observability tooling typically built by data platform teams in production environments.

This project builds key data engineering skills in:

- Parsing unstructured text (log processing)
- Building idempotent, testable ETL pipelines
- Working with structured storage (SQLite)
- Creating metrics and summary reports
- Writing maintainable CLI tools

Deliverables:

Component	**Description**
etl_apache.py	CLI tool to run ETL steps (extract, transform, load)
parser.py	Apache log parser using regex
database.py	DB schema, connection management, and insert logic
summarizer.py	Daily reports generator (JSON or CSV)
logs.db	SQLite database with structured logs
README.md	Usage instructions, schema diagram,

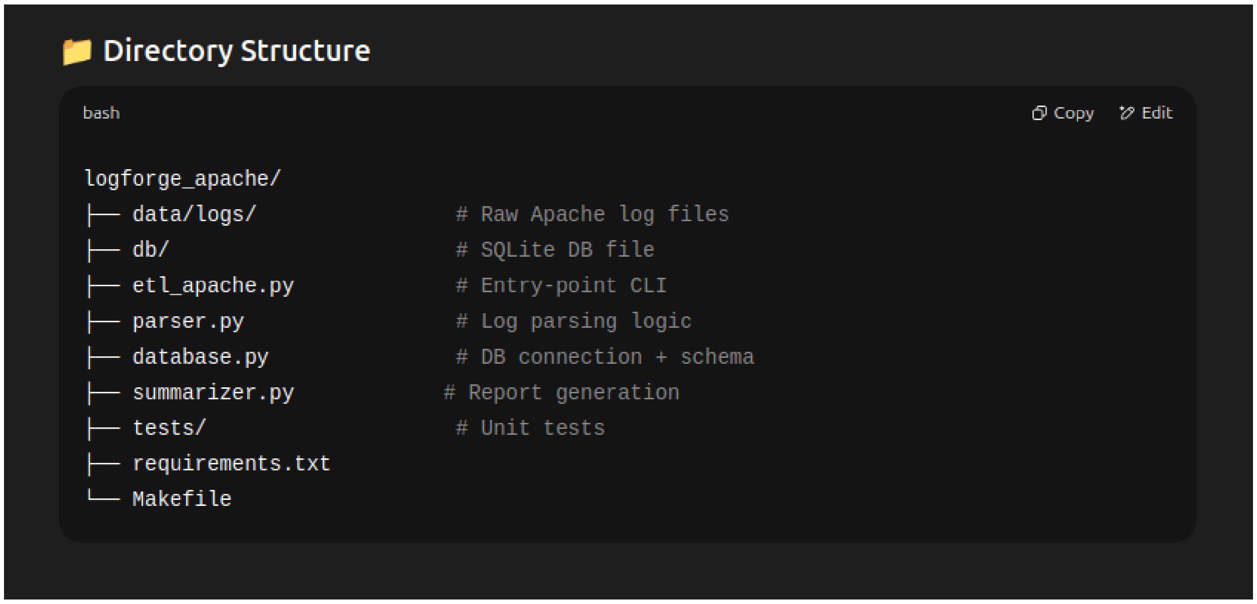
	architecture notes
requirements.txt	Python dependencies

Tech Stack:

- **Python** (primary language)
- **SQLite** (lightweight storage engine)
- **Argparse** (CLI interface)

Pipeline Structure:

Directory Structure



```bash
logforge_apache/
├── data/logs/              # Raw Apache log files
├── db/                     # SQLite DB file
├── etl_apache.py           # Entry-point CLI
├── parser.py               # Log parsing logic
├── database.py             # DB connection + schema
├── summarizer.py          # Report generation
├── tests/                  # Unit tests
├── requirements.txt
└── Makefile
```

Extract

- Read Apache log files (*.log) from data/logs/

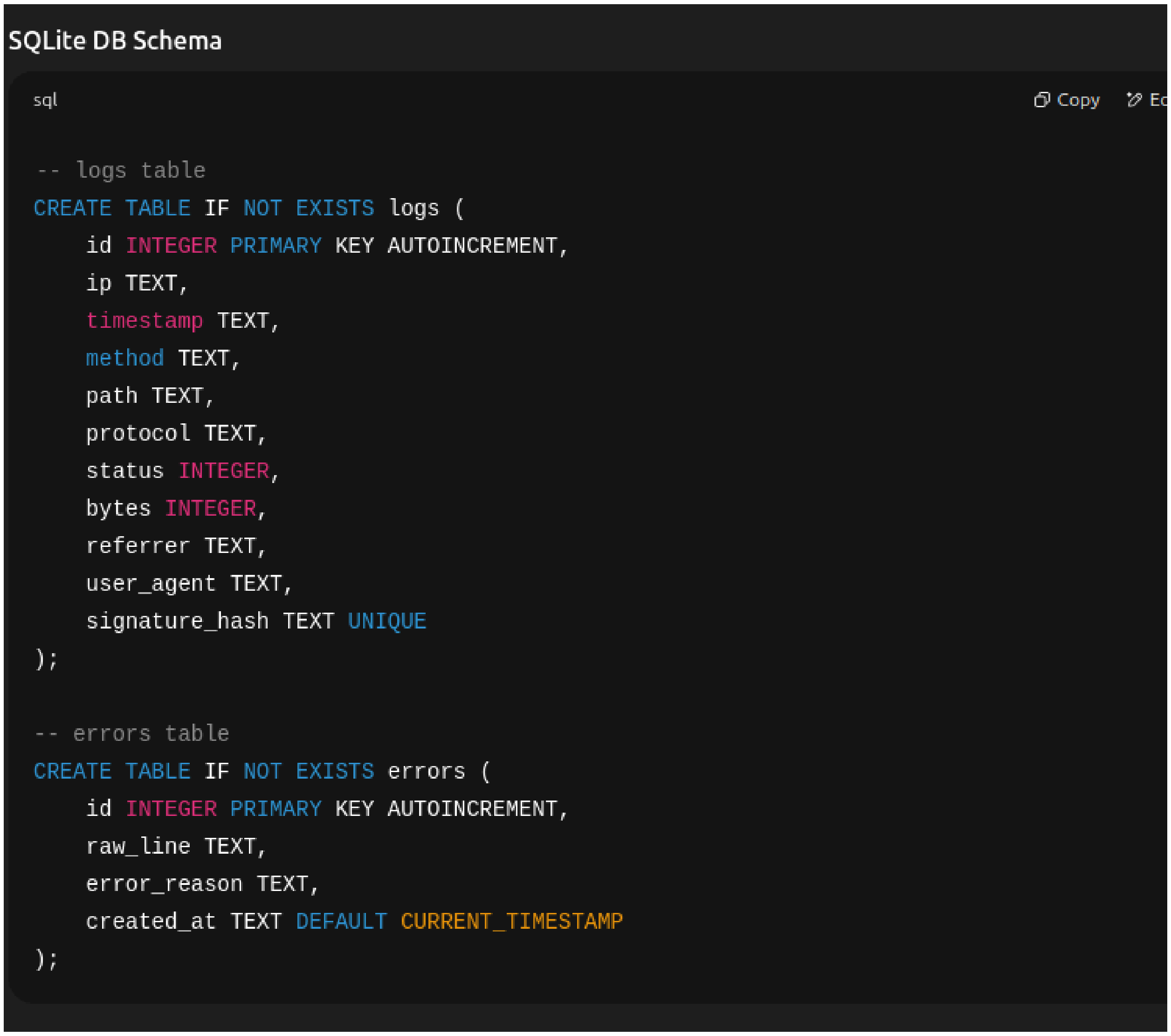
Transform

- Use regex to extract fields:

  - IP address
  - Timestamp
  - HTTP method
  - Resource path
  - Protocol
  - Status code
  - Bytes sent
  - Referrer
  - User-agent
- Validate timestamp and HTTP status format
- Deduplicate using hash of IP + timestamp + endpoint

Load

- Store results in a SQLite DB:
  - logs table: structured fields
  - errors table: malformed lines



```sql
-- logs table
CREATE TABLE IF NOT EXISTS logs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    ip TEXT,
    timestamp TEXT,
    method TEXT,
    path TEXT,
    protocol TEXT,
    status INTEGER,
    bytes INTEGER,
    referrer TEXT,
    user_agent TEXT,
    signature_hash TEXT UNIQUE
);

-- errors table
CREATE TABLE IF NOT EXISTS errors (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    raw_line TEXT,
    error_reason TEXT,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);
```

Reporting & Summaries

Generate JSON/CSV summaries:

- Top 10 endpoints
- Status code distribution
- Top client IPs

Sample CLI Usage

python etl_apache.py extract --input data/logs/

python etl_apache.py transform

python etl_apache.py load --db db/logs.db

python etl_apache.py summary --output-format json

Engineering Challenges:

- Handling incomplete or malformed logs gracefully
- Writing robust regex and tests for every log field
- Designing idempotent ETL (re-runs won't create duplicates)
- Providing meaningful error logs and debug messages
- Efficient DB design for querying/reporting

Resources

- Understanding Regex **-** https://regex101.com/
- Working with Files in Python - https://realpython.com/working-with-files-in-python/
- SQLite - https://realpython.com/python-sqlite-sqlalchemy/
- Understanding Apache Log - https://httpd.apache.org/docs/2.4/logs.html#accesslog