Tutorial: Data Engineering

CS4262/5462

Building a Data Processing Pipeline

1. **Clean** raw HTML files
2. **Extract** structured records with a local LLM
3. **Validate** with Pydantic schemas
4. **Store** embeddings in vector DB & metadata in structured DB
5. **Tune** hyperparameters and optimize query performance
6. **Build** a RAG pipeline based on the data in DB

HTML Cleaning

- Real-world data is messy
  - Unrelated pages: index pages, author pages, …
  - Unrelated elements: Navigation, ads, scripts, …
- We work on 140 raw HTML file from Common Crawl
- TODO 1: Use BeautifulSoup to parse metadata and body text

LLM Extraction with Structured Decoding

- Use LLM to extract information from raw articles
- Why structured decoding?
  - Constrains LLM to produce valid JSON matching a schema
  - Avoid complex postprocessing logic to handle free-form output

```
class LLMExtractedRecord(BaseModel):
    article_id: str = Field(pattern=r"^A-\d{4}$")
    keyword: str = Field(pattern=r"^[a-z][a-z0-9-]{2,31}$")
    category: CategoryType
    priority: int = Field(ge=1, le=5)
    year: int = Field(ge=2000, le=2026)
    summary: str = Field(min_length=40, max_length=300)
    caption: str | None = Field(default=None, max_length=80)
```

Post-processing: Validation and Merging

- Why do we need post-processing?
  - Structured decoding cannot avoid semantic errors
  - Conflicts between semantic retrieved value and metadata
- TODO 2
  - Validate keyword presence in text
  - Merge extracted data with metadata while resolving conflicts

Store in Databases

- Embed the text into dense vectors for similarity search
- Store the embeddings in a vector DB (ChromaDB)
- Metadata (e.g., category) are stored in a structured DB (DuckDB)
- TODO 3: Implement vector DB search
  - Encode a query and run similarity search in the vector DB

Hyperparameter Tuning

- Essential for optimizing query performance
- Investigate the tradeoff between recall and latency
  - `M`: higher → denser graph, better recall, more memory
  - `construction_ef`: higher → better index quality
  - `search_ef`: higher → better recall, slower search
- TODO 4: Compute recall@k for each config against ground truth

RAG with Structured DB + Vector DB

- Extract filter values by LLM and fill in a SQL template
- Fetch IDs of satisfying articles
- Retrieve relevant articles among them in the vector DB
- Generate answer based on the retrieved data
- TODO 5: build SQL from LLM-extracted filters, query DuckDB, then vector search on ChromaDB

Getting Started

- Google Colab + T4
- Our notebook: https://drive.google.com/file/d/1i_aS08bZcfQeYi5Ln33HGEMwTLD91pSh/view?usp=sharing
- Submission: upload the notebook with output to Canvas

Thanks for Listening!