Large Language Models for Legal Epidemiology

October 7, 2024

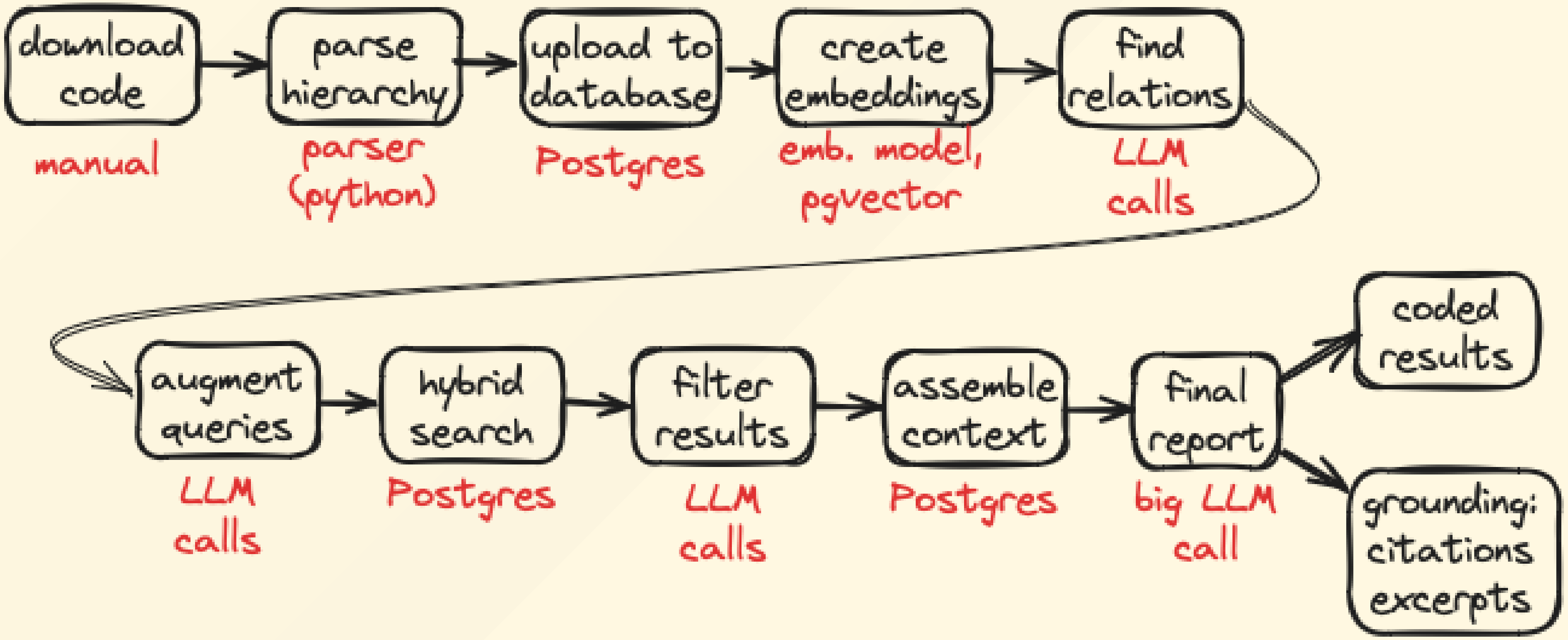
Agenda

1. Recap
2. Large context models
3. Automating configuration for each jurisdiction
4. Testing and quantitative evaluation
5. Infrastructure for collaboration

Guiding principles

1. Ground all results with citations and retrieved source excerpts
2. Find all relevant sources!
3. Keep humans in the loop – design, verify, solve problems
4. To extent possible, minimize complexity and external dependencies

Prototype: overview



Large context windows ('Is RAG dead?')

- Anthropic (Claude 3 family) and Google (Gemini 1.5 family) provide models with context windows greater than **1M tokens** (about 700,000 words)
- This is comparable to the length of many municipal codes (especially if you leave out obviously irrelevant sections)
- Further, **prompt caching** (now offered by Anthropic, Google, and OpenAI) makes this more affordable
- So can you just upload the code and ask the model your question?

Pricing

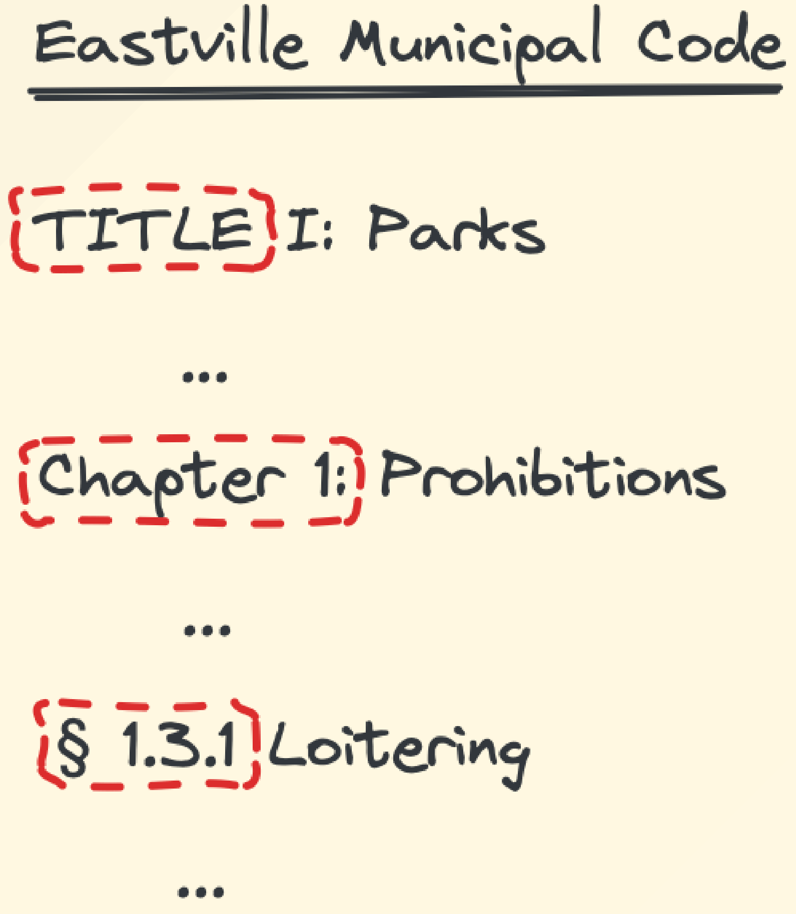
- Pricing has come down a lot
- Gemini 1.5 Pro is $2.50/1M tokens, and $4.50/1M/h context caching
- Likely cost is **$10s of dollars per jurisdiction** for a flagship model
- Economy models (e.g., Gemini 1.5 Flash) cost significantly less, and still do well on many retrieval tasks.

Pros and cons

- **Pro** Minimal programming and setup needed
- **Pro** Anecdotally, seems to do well at retrieval and automated coding tasks
- **Pro** Capabilities are matched to legal epi use case (limited set of million-word documents, small suite of standardized queries)
- **Con** No simple way to get specific cited text that hasn't been processed by LLM, so more work needed to verify outputs (searching though code for citations, etc.)
- **Con** Still greater cost, at least in LLM service fees

Parsing

- Want to segment cleanly according to code hierarchy
- Each jurisdiction uses a different system
- Off-the-shelf tools don't really work



Parsing: configuration using *original* approach

Write and test regular expressions for the headings used in the code:

```
chicago = Jurisdiction(
    name="Chicago",
    hierarchy={
        "title":   r"TITLE \d+",
        "chapter": r"CHAPTER \d+-\d+",
        "article": r"ARTICLE [IVX]+\\.",
        "section": r"\d+-\d+-\d+",
    },
    source_local="../data/chicago/chicago.txt",
)
chicago_tree = chicago.parse()
```

Parsing: *current* approach using cut & paste

An analyst supplies a few examples at each level (no coding or writing regular expressions):

```
H1: "TITLE 1\nGENERAL PROVISION\n"
    "TITLE 2\nCITY GOVERNMENT AND ADMINISTRATION\n"
    "TITLE 3\nREVENUE AND FINANCE\n"

H2: "CHAPTER 1-4\nCODE ADOPTION - ORGANIZATION\n"
    "CHAPTER 1-8\nCITY SEAL AND FLAG\n"
    "CHAPTER 1-12\nCITY EMBLEMS\n"

H3: "1-4-010 Municipal Code of Chicago adopted.\n"
    "2-1-020 Code to be kept up-to-date.\n"
    "3-4-030 Official copy on file.\n"                                    .
```

Parsing: automatic configuration

- An LLM call creates patterns capturing these headings
- The patterns are incorporated into a formal grammar for the document outline using a standard parser-generator (Lark, a modern version of lex)
- The generated parser segments the document into a tree structure for subsequent processing
- It would be nice to have an LLM handle the first step of extracting example headings, since this seems pretty easy; but I haven't been able to get this to work reliably

Testing and quantitative evaluation

- Moving beyond prototype / tinkering phase
- Make it possible for other people to go through workflow without programming
- Evaluate against hand-coded examples to assess performance

Infrastructure for collaboration

- Code on Github (moving from local machine), MIT license
- Postgres server running from high-tech data center (broom closet), securely available remotely by VPN

https://github.com/jsqr/muniscope

Workflow

Data reduction workflow should consist of:

1. Syncing local working environment with Github
2. Copying a municipal code to a `data/city` subdirectory
3. Running a script to clean and convert the code to a single plain text file `data/city/code.txt`
4. Making a copy of `template.ipynb` to `city.ipynb`
5. Going through the notebook section by section, with state and outputs saved to the Postgres database along the way.