CS 839 Next Generation Data Systems
Lecture 7: Agentic Retrieval

Tianyu Li
litianyu@cs.wisc.edu

Reading Check-in

- You have ~10 min (until 11:15) to answer 2 questions

- No electronics

Administrivia

- Project Proposals due 10/8. Expected length: 2 -3 pages. Should include:
  - Project title
  - Names and wisc emails of all team members
  - Intro: Why your project matters
    - Why is it interesting?
    - How does it compare to existing work?
    - How does it advance the state-of-the-art? Why is it needed?
    - What is your hypothesis?
  - Methodology: How do you plan to approach the problem?
  - Evaluation: How will you evaluate your project? What is your success metric? What resources do you need to achieve this?
  - Timeline: What do you plan to achieve by the Mid-Term Project report, Project Presentation day, and final project hand-in?
  - Deliverables: By the end of the project, what are your planned deliverables?

Disclaimer

- We are entering uncharted territory from now on
- Expect things to be in flux and a lot more discussion focused

Recap

- So far, various systems and algorithms to identify relevant documents given a natural language query
  - Classical IR
  - Dense Retrieval
  - Various techniques to make vector search work in real systems
  - Multi-vector search
  - Other new techniques to improve embedding models, etc.

- …are we done?

A familiar task

- POV: You are building an agentic coding application

- User ask the coding agent: “Why does an upload sometimes get retried after the user cancels it?”
  - Retrieval task in disguise – must find where cancellation is handled, where retries are scheduled, how status propagates, relevant test cases, etc.

- Can we use what we learned about to build a retrieval pipeline for coding agents?

Problem

- In modern LLM applications, users are often expressing intent, not queries in the traditional retrieval sense

- Worse, important search terms may only become apparent after reading previous results
  - E.g., some steps to solve the coding task may become apparent only after initial searches find the code at fault

Pipeline -> Loop

- Idea: a model chooses successive retrieval actions based on the task and what's learned so far

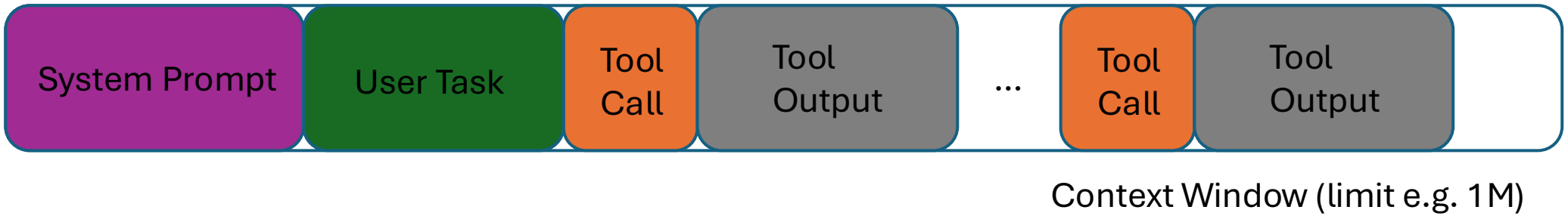
```
ctx = system_prompt + task
while true:
    action, done = parse(model.call(ctx))
    if done:
        break
    ctx += action.exec().output
```

A Primer on Agents

```
ctx = system_prompt + task
while true:
    action, done = parse(model.call(ctx))
    if done:
        break
    ctx += action.exec().output
```

The “agent harness”

Agent: Execution Model



- Scarce Resource: Context
  - Long context is expensive and slow (due to quadratic scaling)
  - Long context often degrades model performance
    - Sometimes due to irrelevant or contradictory information
    - Could also be from degradation due to diluted attention scores, lost-in-the-middle, etc.

A Primer on Agents

```
ctx = system_prompt + task
while true:
    action, done = parse(model.call(ct
    if done:
        break
    ctx += action.exec().output
```

You are an assistant that solves tasks using the tools below.
Available tools:
- search(query: str): Search the web and return relevant snippets.
- run_python(code: str): Execute Python and return its output.

At each step, output exactly one JSON object.
To call a tool:
{"type": "tool_call", "name": "<tool>", "arguments": {...}}
To answer the user:
{"type": "final", "answer": "<your answer>"}
Use tool results to decide what to do next. Do not invent tool results.

Model Context Protocol

- This formulation of agents is very general
  - From coding agents to search agents to almost anything you want
  - However, it's clearly a software engineering nightmare to handroll all of the logic for every use case
- Standard: MCP
  - Idea: a standard API for clients (harness) and servers (tool execution)
  - Harness automatically builds context using tool descriptions and executes tools by calling the server
  - Standard tool definition/request/response format with JSON
- MCP often criticized for unnecessarily bloating context
  - Progressive disclosure techniques like skill

Agentic Retrieval: Design Space

- What actions can the agent take?
  - Keyword/vector search, SQL, file system, web search, etc.
- Model behavior
  - How training/prompting influence the agent's strategy
- Context Management
  - How are tool prompts organized, results presented, and space usage managed, etc.

Plan for Today

- Agentic search is a rapidly evolving field!
  - Lots of new ideas being tried
  - Limited understanding of the design space and trade-offs

- We will examine a few case studies and leave most of the time for discussion

Case Study 1: Cursor

- What actions can the agent take?
  - Vector search, grep, file inspection
- Model behavior
  - Cursor trains custom embedding model & reranker using agent trajectories
  - Bring-your-own-model with extensive system prompting
- Context Management
  - Skills + hide large outputs + self-compaction + searchable history

Cursor: Vector Index

- Code files are split into syntactic chunks and embedded asynchronously
  - Index (TurboPuffer) maps embeddings to obfuscated snippet references
  - Source text is retrieved locally when needed
- Vector index stored remotely
  - Use Merkle trees to discover changes and re-embed on demand
  - Similar workspaces share indexes within the same team/user
  - Synchronization/checks necessary before exposing results
- Lots of investment in better retrieval tools!

Case Study 2: Claude Code

- What actions can the agent take?
  - Glob, Grep, Read, Bash (Only the command line!)
- Model behavior
  - Claude models are mostly likely trained with the cc harness
  - CLAUDE.md + skill plugins
- Context Management
  - Sub-agents + self-compaction
  - Older tool outputs cleared as context fills

Case Study 3: Chroma Context-1

- What actions can the agent take?
  - hybrid_search, grep, read_document, prune_chunks
- Model behavior
  - Fine-tuned specialized model with supervised/RL
- Context Management
  - Harness reports token usage and enforces limit
  - Model self-selects chunks to discard

Case Study 4: Page Index

- Somewhat different setup from before: focused on index design

- Idea: build a document tree


```
...
{
  "node_id": "0006",
  "title": "Financial Stability",
  "start_index": 21,
  "end_index": 22,
  "summary": "The Federal Reserve ...",
  "sub_nodes": [
    {
      "node_id": "0007",
      "title": "Monitoring Financial Vulnerabilities",
      "start_index": 22,
      "end_index": 28,
      "summary": "The Federal Reserve's monitoring ..."
    },
    {
      "node_id": "0008",
      "title": "Domestic and International Cooperation and Coordination",
      "start_index": 28,
      "end_index": 31,
      "summary": "In 2023, the Federal Reserve collaborated ..."
    }
  ]
}
...
```

Case Study 4: Page Index

- PageIndex follows original document structure when it exists
  - Otherwise, use LLM to infer
  - LLM summarizes each node
- Traversal follows progressive disclosure
  - Earlier layers are given in context, and the model chooses which layers to expand and fetch into the context
- Multiple documents are grouped together using file-system abstractions

Discussion

- What drawbacks of agentic retrieval have we overlooked and when do they become salient?
- Where would you place your research bet: better tools or better models? What research will be “bitter-lesson proof” in this space?
- Will the future be about smarter, more generalist models, or specialized agents and tool stack?
- What are the key missing pieces from a convincing demo to production search agents?