AI Systems: Building GenAI workflows

COS 418/518: Distributed Systems
Lecture 21

Mike Freedman, Wyatt Lloyd

1

Today's lecture

- Large language models (ChatGPT, Claude, Llama, etc.) are here
- Today's topic: How can we use them when building systems?
- What we are NOT covering:
  - How LLMs work: model architecture, training, serving
  - See COS 485, COS 568

2

2

Outline

- Foundational models
- Fine-tuning vs. contextual learning
- Retrieval-Augmented Generation (RAG)
- Vector Search on embeddings
- Model Context Protocol (MCP)
- Deploying models behind APIs

3

3

What is Generative AI?

- AI models that generate new content (text, images, code, etc.) from learned probability distributions. Try to produce new samples (e.g., what's the next word/token) given input and model.
- Difference from traditional discriminative models (e.g., classifiers, which try to classify inputs to existing samples.
- Examples:
  - Text generation: GPT family (next-token prediction via transformers).
  - Image synthesis: GANs, diffusion models (denoising data step-by-step).
  - Code generation: GitHub Copilot (autoregressive models trained on code repositories).
- Underlying techniques:
  - Neural networks, especially transformers
  - Trained on massive datasets

4

4

What are Foundational Models?

- Versatile and basis for many GenAI systems
- Large, general-purpose models trained on broad data (today, often "transformer based")
- Examples: GPT (OpenAI), LLaMA (Meta), Claude (Anthropic), Gemini (Google)
- Characteristics:
  - Billions of parameters (GPT-3: 175B parameters, GPT-4 even larger).
  - Training today requires large-scale distributed computing, GPU clusters, and large-scale optimization techniques.

5

5

Improving foundational models for use case

Fine-tuning

- Additional training on specialized data to produce a new version of the model
- Produces a new version of the model
  - Take foundation model's weights as initial.
  - Train (typ. supervised learning) on task-specific labeled datasets with smaller learning rates.
  - Result: Model's weights adjusted to capture new data, altering internal representations.
- Pros: High accuracy, task-specific knowledge
- Cons: Expensive, slower, risk of forgetting

Contextual learning

- Provide additional information directly in the input prompt (prompt engineering).
- Doesn't change model weights; uses original model at inference.
- Examples of context:
  - Recent thread of discussion between user and model
  - "Few-shot" learning with several labelled examples.
- Pros: much more flexible at run-time
- Cons: limited context

6

6

Examples of contextual learning

I plan to visit Princeton University in May during a college tour.

What are the top 5 things I should visit?

That sounds like a fantastic trip! Princeton University is full of history, charm, and stunning architecture. Here are **the top 5 things you should definitely visit** during your tour:

1. Princeton University Art Museum (temporarily relocated)

What about in the town?

Great question—**downtown Princeton** is small but packed with charm, history, and great spots to explore. It's very walkable and seamlessly blends into the university campus. Here are the **top things to check out in the town itself**:

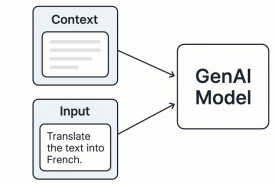
🏛️ 1. Palmer Square

7

7

Few-shot prompting

- LLM is given a few examples of input-output pairs to help guide behavior for a specific task.
- Useful when you want the model to generalize from a small number of demonstrations rather than rely only on task descriptions.



8

8

Few-shot: Text Classification

- Classify the following movie reviews as Positive or Negative.
- Review: "I loved the story and the acting was superb!"
- Sentiment: Positive
- Review: "The plot was predictable and the characters were flat."
- Sentiment: Negative
- Review: "It was a fun and engaging experience with brilliant visuals."
- Sentiment:

9

9

Few-shot: Named Entity Recognition (NER)

- Extract the names of people, locations, and organizations.
- Text: "Barack Obama was born in Hawaii and was president of the United States."
- Entities: Person: Barack Obama; Location: Hawaii; Organization: United States
- Text: "Sundar Pichai is the CEO of Google, which is based in Mountain View."
- Entities:

10

10

Few-shot prompting

- Write a blog post in the style of Mike Freedman, describing X.
- Examples: <Two recent blog posts>

11

11

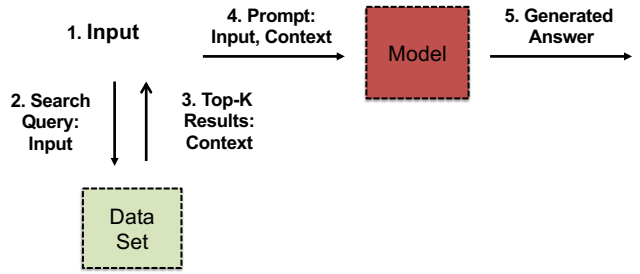
What if your context is really large?

- Examples:
  - Software repository with 1M lines of source code
  - 100s of pages of technical documentation
  - Internal corporate knowledge base / intranet
- Enter Retrieval-Augmented Generation (RAG)…

12

12

Retrieval-Augmented Generation (RAG)



13

13

How to "search" in RAG?

- Traditional "keyword search" bad for semantic meaning
  - "User" and "person" don't match in keyword search, but mean similar things in English
- Semantic search
  - Concept: Search by meaning, not exact match
  - But how to find nearest items in "semantic space"?

14

14

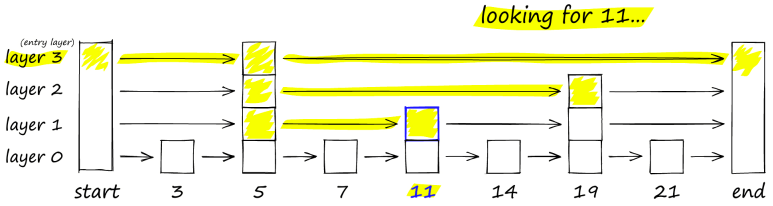
Vector Search on Embeddings

- Method to find similar items (text, images, audio) by comparing mathematical representation of items – called *embeddings* or *vectors* – in high-dimension space
- Problem: Given a set of vectors D and an input vector I, find some *k* vectors in D that are nearest to I according to a distance metric
  - Common distance metrics: cosine similarity, Euclidean distance in high-dimensional space
  - Anywhere from 100s of vector embeddings to billions of embeddings
- This is basically a search or indexing problem to find the top-k vectors:
  - IVFFlat (InVerted File Flat)
  - HNSW (Hierarchical Navigable Small World Graph)
  - DiskANN (Disk-based Approximate Nearest Neighbor)
- Vector DBs: Pinecone, Qdrant, Weaviate, Postgres pgvector, Timescale pgvectorscale

15

15

Ex: Hierarchical Navigable Small World Graph

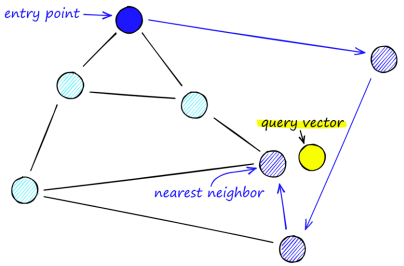


Conceptual idea: Skip List

16

16

Ex: Hierarchical Navigable Small World Graph

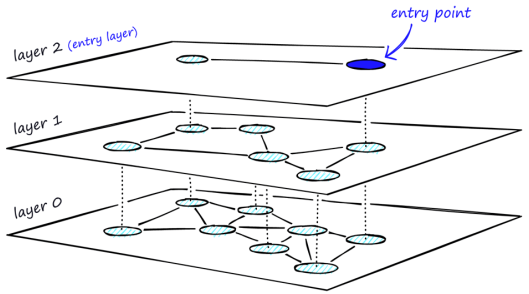


Conceptual idea: Navigable Small World

17

17

Ex: Hierarchical Navigable Small World Graph

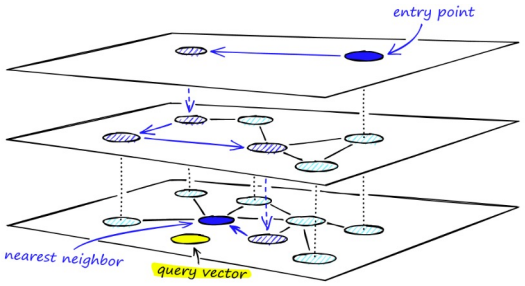


Hierarchical Navigable Small World

18

18

Ex: Hierarchical Navigable Small World Graph



Hierarchical Navigable Small World

19

19

Vector Search on Embeddings

Vector index creation:

- Take input set of documents.
- For each input document:
  - Divide document into text chunks
  - For each chunk, use ***embedding model*** to compute vector embedding
  - Insert vector embedding into database
  - Create mapping from vector embedding to text chunk
- Build a vector index on embeddings

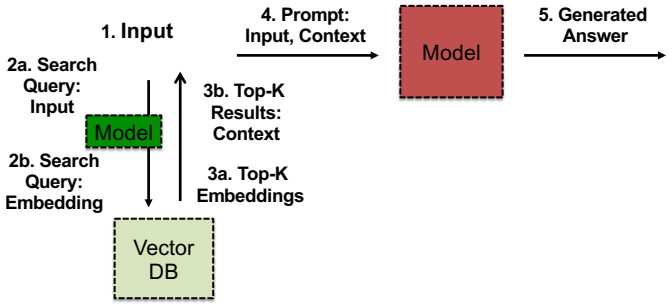
Vector Search:

- Given input query
- Use ***embedding model*** to compute vector embedding of input query
- Perform ***vector search*** to find top-k embeddings closest to input embedding
- Lookup text chunk associated with each top-k embedding
- Return top-k text chunks

20

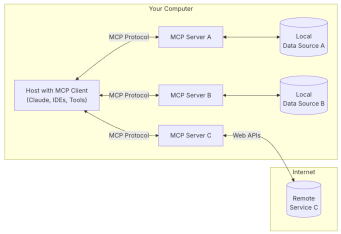
20

Retrieval-Augmented Generation (RAG)



Model-Context Protocol (MCP)

- Generalize notion of “getting context” for our model?
  - In Nov 2024, Anthropic released MCP, a protocol for structuring how applications and tools can provide context to LLMs.
  - Think of MCPs as the “APIs” for LLMs
- What do MCP servers expose?
  - **Resources:** File-like data that can be read by clients (like API responses or file contents)
  - **Tools:** Functions that can be called by the LLM (with user approval)
  - **Prompts:** Pre-written templates that help users accomplish specific tasks



Model-Context Protocol (MCP)

Resource discovery

Clients can discover available resources through two main methods:

Direct resources

Servers expose a list of concrete resources via the `resources/list` endpoint. Each resource includes:

```
{
  uri: string;          // Unique identifier for the resource
  name: string;         // Human-readable name
  description?: string; // Optional description
  mimeType?: string;    // Optional MIME type
}
```

Reading resources

To read a resource, clients make a `resources/read` request with the resource URI.

The server responds with a list of resource contents:

```
{
  contents: [
    {
      uri: string;        // The URI of the resource
      mimeType?: string;  // Optional MIME type

      // One of:
      text?: string;      // For text resources
      blob?: string;      // For binary resources (base64 encoded)
    }
  ]
}
```


Model-Context Protocol (MCP)

Concepts

Tools

Copy page

Enable LLMs to perform actions through your server

Tools in MCP allow servers to expose executable functions that can be invoked by clients and used by LLMs to perform actions. Key aspects of tools include:

- **Discovery**: Clients can list available tools through the `tools/list` endpoint
- **Invocation**: Tools are called using the `tools/call` endpoint, where servers perform the requested operation and return results
- **Flexibility**: Tools can range from simple calculations to complex API interactions

```
const postMessageTool: Tool = {
  name: "slack_post_message",
  description: "Post a new message to a Slack channel",
  inputSchema: {
    type: "object",
    properties: {
      channel_id: {
        type: "string",
        description: "The ID of the channel to post to",
      },
      text: {
        type: "string",
        description: "The message text to post",
      },
    },
    required: ["channel_id", "text"],
  },
};
```

Example MCP: Neon Database Server

For example, in Claude Desktop, or any MCP Client, you can use natural language to accomplish things with Neon, such as:

- `Let's create a new Postgres database, and call it "my-database". Let's then create a table called users with the following columns: id, name, email, and password.`
- `I want to run a migration on my project called "my-project" that alters the users table to add a new column called "created_at".`
- `Can you give me a summary of all of my Neon projects and what data is in each one?`

Example: Tool Discovery

- run_sql_transaction
- get_database_tables
- describe_table_schema
- start_database_migration
- commit_database_migration

These *mostly* map to Neon API endpoints. However, certain tools such as the database migration related ones are tailored specifically for the usage we expect from AI agents and LLMs. As we showed above, these help the model by running the expected SQL on a side branch first and *then* hinting the agent to test the migration and commit it as it sees fit.

```
..
{
  name: 'describe_table_schema' as const,
  description: 'Describe the schema of a table in a Neon database',
  inputSchema: describeTableSchemaInputSchema,
},
..
```

25

25

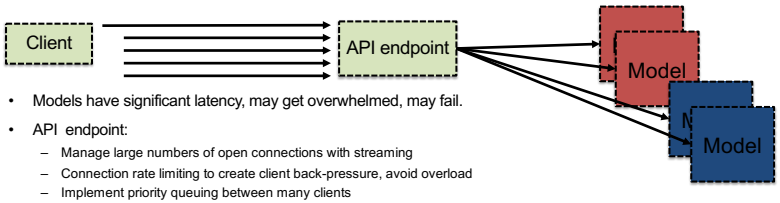
Example MCP: Neon Database Server

```
..
{
  name: 'prepare_database_migration' as const,
  description: `
<use_case>
  This tool performs database schema migrations by automatically generating and execut

  Supported operations:
  CREATE operations:
  - Add new columns (e.g., "Add email column to users table")
  - Create new tables (e.g., "Create posts table with title and content columns")
  - Add constraints (e.g., "Add unique constraint on users.email")

  ALTER operations:
  - Modify column types (e.g., "Change posts.views to bigint")
  - Rename columns (e.g., "Rename user_name to username in users table")
  - Add/modify indexes (e.g., "Add index on posts.title")
  - Add/modify foreign keys (e.g., "Add foreign key from posts.user_id to users.id")

  DROP operations:
  - Remove columns (e.g., "Drop temporary_field from users table")
  - Drop tables (e.g., "Drop the old_logs table")
  - Remove constraints (e.g., "Remove unique constraint from posts.slug")

  The tool will:
  1. Parse your natural language request
  2. Generate appropriate SQL
  3. Execute in a temporary branch for safety
  4. Verify the changes before applying to main branch
</use_case>

<workflow>
  1. Creates a temporary branch
  2. Applies the migration SQL in that branch
  3. Returns migration details for verification
</workflow>

<important_notes>
  After executing this tool, you MUST:
  1. Test the migration in the temporary branch using the 'run_sql' tool
  2. Ask for confirmation before proceeding
  3. Use 'complete_database_migration' tool to apply changes to main branch
</important_notes>

<example>
  For a migration like:
  ALTER TABLE users ADD COLUMN last_login TIMESTAMP;

  You should test it with:
  SELECT column_name, data_type
  FROM information_schema.columns
  WHERE table_name = 'users' AND column_name = 'last_login';

  You can use 'run_sql' to test the migration in the temporary branch that this
  tool creates.
</example>
```

26

26

Deploying models behind APIs



- Models have significant latency, may get overwhelmed, may fail.
- API endpoint:
  - Manage large numbers of open connections with streaming
  - Connection rate limiting to create client back-pressure, avoid overload
  - Implement priority queuing between many clients
- Client endpoint:
  - Request timeouts and retry logic from errors
  - (Exponential) backoff from rate limiting
- Common needs
  - Chained tool calling require ordering / sequence numbers
  - Tool calling without idempotency may require transaction numbers
- Similar problems as many RPC systems!

27

27

Conclusions

- We're in the "Cambrian Era" of GenAI models and systems
- Every 3 months is something new; industry and tools are rapidly changing and evolving.
- One fundamental difference:
  - For past 50 years, our computer systems are heavily deterministic. We rely on that determinism.
  - GenAI systems (and the inputs to them) are inherently non-deterministic. Natural language is not precise. Still figuring out how to build "robust" systems in face of such non-determinism / imprecision.

28

28


A representation of the arrival of Generative AI, and the transition from deterministic computing systems to the rise of non-deterministic LLMs. Make it cyperpunk in appearance. Futuristic, but representing clearly both the past and the future, and actually illustrate the transition. Show part of the image as the "old" and part of the image as the "new".
29

29