CSE 291 Virtualization and Cloud Computing: Course Summary

Yiying Zhang
Outline

• Finishing up LLM agents

• Virtualization summary

• Hints for computer systems design

• Final project presentation this Wed
  • (7-8min presentation + 2-3min Q&A) * 8 groups

• Final project summary report due 3/21

• Course evaluation!
What is LLM Agents

Here is a famous picture from Lilian Weng (from OpenAI)

https://gptpluginz.com/llm-agents/

What is LLM Agents

Planning:

- **Subgoal and decomposition:** The agent breaks down large tasks into smaller, manageable subgoals, enabling efficient handling of complex tasks.
- **Reflection and refinement:** The agent can do self-criticism and self-reflection over past actions, learn from mistakes and refine them for future steps, thereby improving the quality of final results.

https://gptpluginz.com/llm-agents/

What is LLM Agents

Memory:

- **Short-term memory**: all the in-context learning is utilizing short-term memory of the model to learn.
- **Long-term memory**: this provides the agent with the capability to retain and recall (infinite) information over extended periods, often by leveraging an external vector store and fast retrieval.
What is LLM Agents

Tool use:

- The agent learns to call **external APIs** for extra information that is missing from the model weights (often hard to change after pre-training), including current information, code execution capability, access to proprietary information sources and more.

https://gptpluginz.com/llm-agents/

What is LLM Agents

Action:

- The agent's ability to execute actions in the real or virtual world is crucial. This can range from performing tasks in a digital environment to controlling physical robots or devices. The execution phase relies on the agent's planning, memory, and tool use to carry out tasks effectively and adaptively.
Why LLM Agents stand out?

- **Language Mastery:** Their inherent capability to both comprehend and produce language ensures seamless user interaction.

- **Decision-making:** LLMs are equipped to reason and decide, making them adept at solving intricate issues.

- **Flexibility:** Their adaptability ensures they can be molded for diverse applications.

- **Collaborative Interactions:** They can collaborate with humans or other agents, paving the way for multifaceted interactions.
Task Decomposition: Chain of thought

Chain of Thought (CoT) has become a standard prompting technique for enhancing model performance on complex tasks. The model is instructed to “think step by step” to utilize more test-time computation to decompose hard tasks into smaller and simpler steps. CoT transforms big tasks into multiple manageable tasks and sheds lights into an interpretation of the model’s thinking process.

![Figure 2: PaLM 540B uses chain-of-thought prompting to achieve new state-of-the-art performance on the GSM8K benchmark of math word problems. Finetuned GPT-3 and prior best are from Cobbe et al. (2021).](image)
Task Decomposition: **Tree of Thoughts**

**Tree of Thoughts** extends CoT by exploring multiple reasoning possibilities at each step. It first decomposes the problem into multiple thought steps and generates multiple thoughts per step, creating a tree structure. The search process can be BFS (breadth-first search) or DFS (depth-first search) with each state evaluated by a classifier (via a prompt) or majority vote.
Self-Reflection: ReACT

ReACT integrates reasoning and acting within LLM by extending the action space to be a combination of task-specific discrete actions and the language space. The former enables LLM to interact with the environment (e.g. use Wikipedia search API), while the latter prompting LLM to generate reasoning traces in natural language.

The ReAct prompt template incorporates explicit steps for LLM to think, roughly formatted as:

```
Thought: ...
Action: ...
Observation: ...
...
(Repeated many times)
```
Self-Reflection: ReACT

In both experiments on knowledge-intensive tasks and decision-making tasks, ReAct works better than the Act-only baseline where a step is removed.

ReAct: Synergizing Reasoning and Acting in Language Models

---

**Question:** Aside from the Apple Remote, what other device can control the program Apple Remote was originally designed to interact with?

**Standard**

**Answer:** iPod

**Thought:** Let's think step by step. Apple Remote was originally designed to interact with Apple TV. Apple TV can be controlled by iPhone, iPad, and iPod Touch. So the answer is iPhone, iPad, and iPod Touch.

**Act Only**

**Act 1:** Search[Apple Remote]

**Obs 1:** The Apple Remote is a remote control ...

**Act 2:** Search[Front Row]

**Obs 2:** Could not find [Front Row]. Similar: ...

**Act 3:** Search[Front Row (software)]

**Obs 3:** Front Row is a discontinued media center software ...

**Act 4:** Finish[Yes]
LLMs + APIs to expert models: HuggingGPT

A text can describe the given image: a herd of giraffes and zebras grazing in a field. In addition, there are five detected objects as giraffe with score 99.9%, zebra with score 99.7%, zebra with 99.9%, giraffe with score 97.1% and zebra with score 99.8%. The bounding boxes are shown in the above image. I performed image classification, object detection and image caption on this image. Combining the predictions of nlpconnet/vit-gpt2-image-captioning, facebook/detr-resnet-101 and google/vit models, I get the results for you.
The system comprises of 4 stages:

- **Task Planning:** LLM analyze the user's requests, breaking them down into solvable tasks through prompts.
- **Model Selection:** LLM is presented with a list of models to choose from and distributes the tasks to expert models. LLM.
- **Task Execution:** Expert models execute on the specific tasks and log results.
- **Response Generation:** LLM receives the execution results and provides summarized results to users.
LLMs + training for tool use: **TALM**

**TALM: Tool Augmented Language Models**
The New England Journal of Medicine is a registered trademark of [QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society] the MMS.

Out of 1400 participants, 400 (or [Calculator(400 / 1400) → 0.29] 29%) passed the test.

The name derives from "la tortuga", the Spanish word for [MT("tortuga") → turtle] turtle.

The Brown Act is California's law [WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.] that requires legislative bodies, like city councils, to hold their meetings open to the public.
Language and System Support for “Composed” LLMs

- How to write programs of LLMs and tools?
- How to efficiently execute these program as a serving system?

LangChain - Language Framework for LLM Chains

**LangChain Components**

- **Models**
  - LLMs
  - Chat Models
  - Text Embedding Models

- **Prompts**
  - PromptValue
  - Prompt Templates
  - Example Selectors
  - Output Parsers

- **Indexes**
  - Document Loaders
  - Text Splitters
  - VectorStores
  - Retrievers

- **Memory**
  - Short Term
  - Long Term
  - ChatMessageHistory

- **Chains**
  - Chain
  - LLMChain
  - Prompt Selector

- **Agents**
  - Tool
  - Toolkit
  - Agent Executor

- **Agent Functions**
  - add_user_message()
  - add_ai_message()
  - message
  - get_relevant_texts

- **Index Related Chain**
  - Stuffing
  - Map Reduce
  - Refine
  - Map-Rerank
SGLang

Language, interpreter, compiler

RadixAttention

- A radix tree of seq=>KV cache
- Reuse KVs by prefix matching
- Eviction using LRU
- Cache-aware scheduling
1. **Sensory Memory**: learning embedding representations for raw inputs, including text, image or other modalities; [Vision encoder/speech encoder]

2. **Short-Term Memory (STM)**: in-context learning. It is short and finite, as it is restricted by the finite context window length of Transformer. [prompt engineering]

3. **Long-Term Memory (LTM)**: the external vector store that the agent can attend to at query time, accessible via fast retrieval. [Retrieval-augmented LMs ]
Retrieval-based language models (LMs)
Retrieval-based LMs = Retrieval + LMs

- It is a language model \( P(x_n \mid x_1, x_2, \ldots, x_{n-1}) \)
  The capital city of Ontario is ___
  (can be broadly extended to masked language models or encoder-decoder models)

- It retrieves from an external datastore (at least during inference time)

(Also referred to semiparametric and non-parametric models)
Why retrieval-based LMs?

LLMs’ knowledge is easily outdated and hard to update

- Who is the CEO of Twitter?

As of my knowledge cutoff in September 2021, the CEO of Twitter is Jack Dorsey.

- Existing knowledge editing methods are still NOT scalable (active research!)

- The datastore can be easily updated and expanded - even without retraining!
Why retrieval-based LMs?

LLMs’ output is challenging to interpret and verify

Can trace knowledge source from retrieval results - better interpretability & control

Generating text with citations

WebGPT: Browser-assisted question-answering with human feedback
Teaching language models to support answers with verified quotes
Why retrieval-based LMs?
LLMs are large and expensive to train and run

Long-term goal: can we possibly reduce the training and inference costs, and scale down the size of LLMs?

e.g., RETRO (Borgeaud et al., 2021): “obtains comparable performance to GPT-3 on the Pile, despite using 25x fewer parameters”
Definition of Retrieval-based LM
A language model (LM) that uses an external datastore at test time

Inference: Index

Find a small subset of elements in a datastore that are the most similar to the query
Questions to answer

What’s the query & when do we retrieve?

What do we retrieve?

How do we use retrieval?

What do we retrieve?

Input

LM

Query

Index

Datastore
Outline

• Finishing up LLM agents
• Virtualization summary
• Hints for computer systems design
Other LLM-Related Systems Problems

Fine tuning and customization (e.g., LoRA)

Space and energy efficient LLM serving

Supporting Mixture-of-Expert models

Model selection and ensemble

Co-locating models on GPUs

Adding structures or regulations to LLM inference

Want to do research in this space? Join WukLab!
Virtualization Approaches

- Hosted interpretation
  - Interpret each instruction, super slow (e.g., Virtual PC on Mac)

- Direct execution with trap-and-emulate
  - Requires a virtualizable processor and only works for the same architecture

- Direct execution with binary translation
  - Works with non-virtualizable processor, but implementing VMM is tricky

- Direct execution with hardware-assisted virtualization
  - Needs new generation of hardware (which is the norm now), mode switching is still not optimized

- Direct execution with paravirtualization
  - Good performance and works with non-virtualizable processors, but require guest OS changes

- OS-level virtualization, library-level, language (app)-level, unikernels, etc.
  - More lightweight and faster to start, but less secure
Virtual Machine Architectures

1. **Xen / VMware ESX**
   - VM-1
   - VM-2
   - VMM
   - Hardware

2. **VMware Workstation / VirtualBox**
   - VM-1
   - VM-2
   - VMM
   - Host OS
   - Hardware

3. **Linux KVM**
   - VM-1
   - VM-2
   - VMM
   - Host OS
   - Hardware
Evolution of serverless

Decreasing concern (and control) over stack implementation

Increasing focus on business logic

Bare Metal

Virtual machines

Containers

Functions
VM ↔ Containers

Virtual Machines
- Host OS
- Hypervisor
- Hardware
- Virtual Machines

Lightweight VM
- LightVM
- Firecracker
- Small OS
- Hypervisor
- Hardware
- Lightweight VM

Unikernels
- LibOS
- Hypervisor
- Hardware
- Unikernels

Secure Container
- gVisor
- Unikernels
- LibOS
- Hypervisor
- Hardware
- Containers

+ Strong security
+ Mostly compatible
- Medium weight
+ Strong security
+ Mostly compatible
- Light weight
+ Strong security
+ Mostly compatible
- Light weight
- Weak security
+ Compatible
- Lightest weight
Outline

• Finishing up LLM agents
• Virtualization summary
• Hints for computer systems design
Systems Design

• The external interface (that is, the requirement) is less precisely defined, more complex, and more subject to change.

• The system has much more internal structure, and hence many internal interfaces.

• The measure of success is much less clear.
<table>
<thead>
<tr>
<th>Why?</th>
<th>Functionality</th>
<th>Speed</th>
<th>Fault-tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Does it work?</strong></td>
<td><strong>Is it fast enough?</strong></td>
<td><strong>Does it keep working?</strong></td>
</tr>
<tr>
<td><strong>Where?</strong></td>
<td><strong>Completeness</strong></td>
<td><strong>Shed load</strong></td>
<td><strong>End-to-end</strong></td>
</tr>
<tr>
<td></td>
<td>Separate normal and worst case</td>
<td><strong>End-to-end</strong></td>
<td><strong>End-to-end</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Safety first</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Interface</strong></td>
<td><strong>Make it fast</strong></td>
<td><strong>End-to-end</strong></td>
<td><strong>Log updates</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Don’t generalize</strong></td>
<td><strong>Split resources</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Get it right</strong></td>
<td><strong>Static analysis</strong></td>
<td><strong>Make actions atomic</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Don’t hide power</strong></td>
<td><strong>Dynamic translation</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Use procedure arguments</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Leave it to the client</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Keep basic interfaces stable</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Keep a place to stand</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td><strong>Plan to throw one away</strong></td>
<td><strong>Cache answers</strong></td>
<td><strong>Make actions atomic</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Keep secrets</strong></td>
<td><strong>Use hints</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Use a good idea again</strong></td>
<td><strong>Use hints</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Divide and conquer</strong></td>
<td><strong>Use brute force</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Compute in background</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Batch processing</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 1: Summary of the slogans**
Keep It Simple

- *KISS: Keep It Simple, Stupid.* (Anonymous)

- *If in doubt, leave if out.* (Anonymous)

- *Exterminate features.* (C. Thacker)

- On the other hand,

- *Everything should be made as simple as possible, but no simpler.* (A. Einstein)
Making Implementation WORK!

• *Perfection must be reached by degrees; she requires the slow hand of time.* (Voltaire)

• *Plan to throw one away; you will anyhow*

• Use a good idea (and implementation) again instead of generalizing
Continuity

• Tension between desire to improve a design and need for stability

• Backward and forward compatibility

• Never break expectation of user space

• Keep a place to stand incase of interface change
Handling Faults and Corner Cases

• Not just avg, look at the tails
• End-to-end error recovery
• Error handling in distributed systems
• Consistency, consensus, atomicity, replication, checkpointing, etc. => take diet sys course
Performance

- Use locality: caching, prefetching, etc.
- Use parallelism and asynchronous execution
- Use hints (application semantics)
- Co-design different layers
- Many other techniques

- Performance is not the only thing: fairness, perf/$, SLA/SLO, ease of management, etc.
Final Thoughts?