Agentic AI and Software Development

A Progression of Language Model Usage in SW development

5/21/2025

Outline

Overview of LMs

Agentic LMs

Agentic LMs in SW development

Summary

Language Model

- Language model:
  - Predicts the next words based on previous words
- How it is trained:
  - Trained to compress world knowledge (text, code)
  - Trained to follow instructions
- ChatGPT like systems:
  - Provide answers within natural conversation

LM Usage

Current Status and Applications

- LMs are rapidly evolving, with new models and products being introduced regularly
- Numerous applications leverage LMs:
  - AI coding assistants
  - Domain-specific AI copilots
  - ChatGPT and other conversational interfaces

LM Usage

Working with Prompts

- Natural language interaction through well-crafted prompts
- Key strategies:
  - Write clear, descriptive instructions
  - Include few-shot examples
  - Provide relevant context (static or dynamic)
  - Enable Chain of Thought (CoT) reasoning. Give model time to think
  - Break down complex tasks. *Chain* complex prompts
  - Systematic trace and evaluation of prompts for performance improvement

A good prompting guideline : https://platform.openai.com/docs/guides/prompt-engineering

Common Limitations of LMs

- Hallucination: Generation of incorrect information with high confidence
- Knowledge cutoff: Limited to training data timeframe
- Lack of attribution: No direct source citations
- Data privacy: Limited to public training data, no access to proprietary information
- Limited context length: Constraints due to attention mechanism architecture

RAG (Retrieval Augmented Generation)

To augment LMs with knowledge from external sources, Retrieval Augmented Generation (RAG) is commonly used.

RAG addresses the following challenges

- Hallucination
- Lack of attribution
- Data privacy
- Limited context length

How it works:

1. Text preprocessing:
    - Chunking large texts appropriately
    - Converting documents to embeddings
2. Storage:
    - Maintaining embedding-text pairs
3. Query processing:
    - Converting queries to embeddings
    - Performing similarity search (Retrieval)
    - Generating enhanced prompts with context (Augmentation)
    - Submitting to LM for final output (Generation)

Tool usage

LMs can generate function signatures to interact with external tools, enabling capabilities such as making API calls or executing code.

Tool usage address the following challenges

- Real-time information
- Computations

- Example: “What is the best cafe based on user reviews?”
  - LLM will generate `{tool: web-search, query: "cafe reviews"}`
  - External tool searches and provides result to LM
- Example: “What is the weather in San Francisco?”
  - LLM will generate `{tool: get-weather, query: "SF"}`
  - External tool is called and provides result to LM
- Example: “If I invest $100 at 7% compound interest for 12 years, what do I have at the end?”
  - LLM generates Python code with this: `{tool: python-interpreter, code: "100 * (1+0.07)**12"}`
  - Generates code, runs it, and produces the output

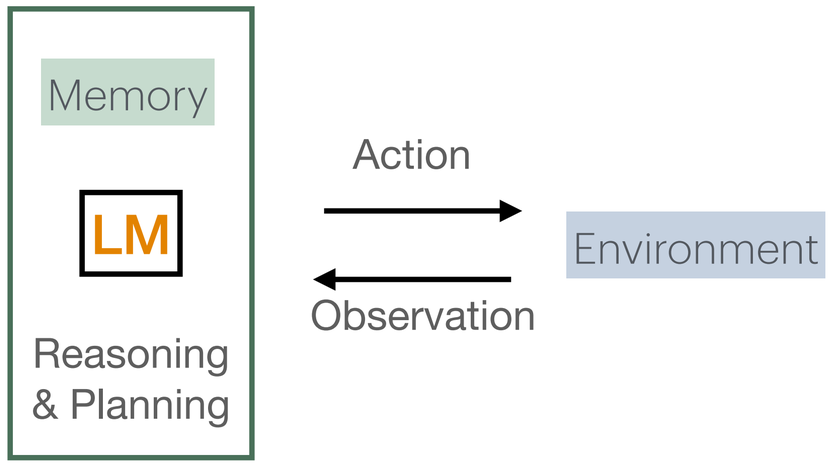
Ref: https://www.deeplearning.ai/the-batch/how-agents-can-improve-llm-performance/

Agentic LMs

Interact with environments

- Simple LM: use cases involve text in and text out

- Agentic LM: Interact with environments *iteratively*



Ref: https://rdi.berkeley.edu/llm-agents-mooc/slides/intro.pdf

Agentic LMs

Overview

A progression of LM usage

- Ability to Reason and Act:
  - Reason about tasks: break down complex tasks and plan actions
  - Execute actions: use external tools, code, and retrieve information
  - Interact with environment: using tools
  - Learn from feedback

Example task:

Customer support case: "*Can I get a refund for product Foo?*"

- Check the refund policy
- Check the customer order information
- Check the product information
- Generate a refund plan and response

Ref: https://www.deeplearning.ai/the-batch/how-agents-can-improve-llm-performance/

Agentic LMs

- Agent workflow: LMs iterate over documents or tasks, using external tools or code to:
  - Research topics using web data, summarize findings, and present output
  - Prepare plans for software fixes and iteratively propose code solutions
- Utilize LMs with different roles (such as generator and critics) iteratively
- Generate plans for using external tools
- Agentic LM usage can achieve more complex tasks than non-iterative and LM-only patterns

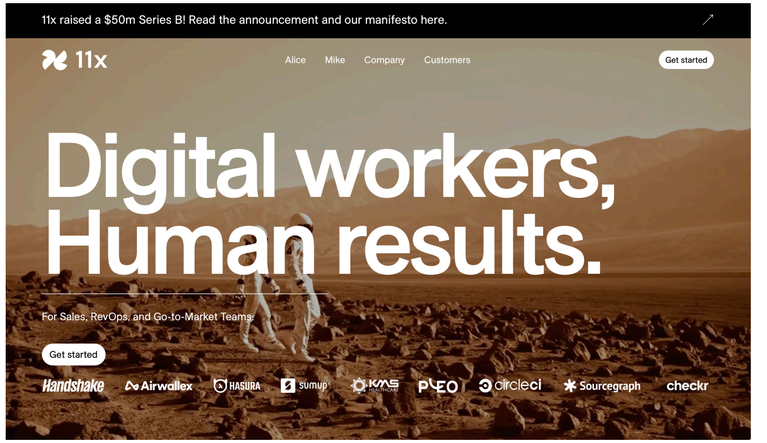
Ref: https://www.deeplearning.ai/the-batch/how-agents-can-improve-llm-performance/

Agentic LMs

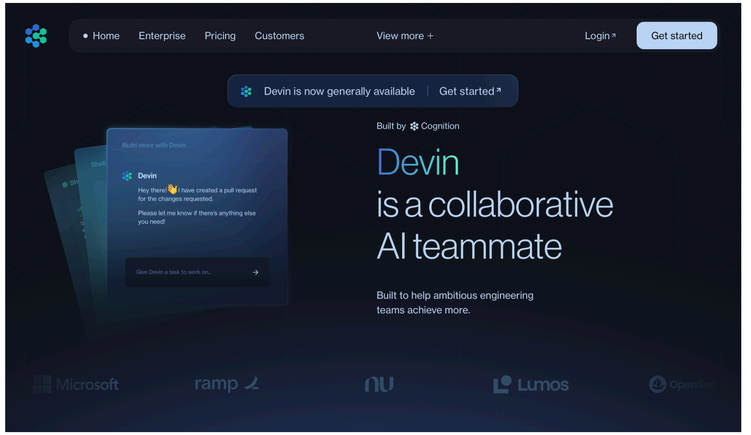
Real-world Applications

- Software Development
  - Code generation and review
  - Bug detection and fixing
- Research and Analysis
  - Data gathering and synthesis
  - Report generation
- Task Automation
  - Workflow optimization
  - Process automation

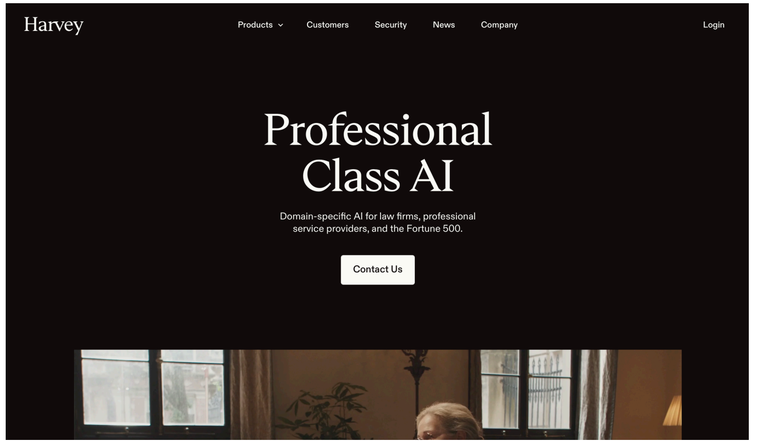
AI SDR



AI SW developer



AI lawyer



Ref: https://cs230.stanford.edu/syllabus/fall_2024/rag_agents.pdf

When to use Agent

- Is the task complex?
- Is the task valuable?
- Is it viable?
- What is the cost of error?

How We Build Effective Agents: Barry Zhang, Anthropic

SW Development

- Is the task complex? Yes
- Is the task valuable? Yes
- Is it viable? Yes
- What is the cost of error? With tests, we can detect issues during development

How We Build Effective Agents: Barry Zhang, Anthropic

AI Progression in SW development

- Auto completion: suggests the code in editor environment
- Chat style AI:
  - AI will answer questions and update the code based on conversations
- Agentic AI:
  - Changes requires understanding multiple files and result of the execution or search

AI software engineer

- AI SWE understand the code base
  - Code indexing and understanding
  - Use tool to read code (add relevant code into context)
- AI can plan and use tools
  - Run posix commands
  - Run code and parse the outputs
  - Run git commands
  - Run the test and check the results

AI in SWE

- Rapid development
- Started with simple ideas but will be applied to more areas in development
- Coding
- Fixing issues
- Finding issues
- Writing tests
- Refactor code

What’s in it for us?

- Use AI tools to improve productivity
- Try out simple tasks first, then extend
- Refactoring or fixing issues is a good start
- Write tests
- Use it as a AI buddy for debugging
- Log or error analysis

Summary

- Agentic LMs represent the next evolution in AI assistance
- Key advantages:
  - Autonomous reasoning and action
  - Tool integration capabilities
  - Iterative improvement through feedback
  - Complex task decomposition
- Growing applications across software development, research, and automation
- AI systems are pushing the boundary of how software is designed, built, tested, and maintained.

References

The following are references for the presentation. We learned from these sources and reused content and images from them:

1. [Agentic Design Patterns Part 1](https://www.deeplearning.ai/the-batch/how-agents-can-improve-llm-performance/)

2. [Large Language Model Agents, MOOC Fall 2024](https://llmagents-learning.org/f24)

3. [Natural Language Processing with Deep Learning, 2024](https://web.stanford.edu/class/cs224n)

4. [Building Effective Agents](https://www.anthropic.com/research/building-effective-agents)

5. [RAG and AI Agents from Deep Learning](https://cs230.stanford.edu/syllabus/fall_2024/rag_agents.pdf)

6. [Tool Use and LLM Agent Basics from Advanced NLP](https://www.phontron.com/class/anlp-fall2024/assets/slides/anlp-15-tooluse-agentbasics.pdf)

7. [What are AI Agents?](https://www.youtube.com/watch?v=F8NKVhkZZWI)

8.[Frontiers-of-AI-Agents-Tutorial](https://frontiers-of-ai-agents-tutorial.github.io/)

Backup materials

Agentic LM design patterns

Here are some key agent design patterns:

- Planning: Multi-step planning to achieve goals
- Reflection: Examines its own work and improves
- Tool usage: Utilizes web search, code executions
- Multi-agent collaboration: Splits work and facilitates discussion

Ref: https://www.deeplearning.ai/the-batch/how-agents-can-improve-llm-performance/

How LMs are trained:

1. Pre-training: LM is trained with large corpus by next token prediction objective
2. Post-training:
    1. Instruction following training adapts pre-trained LM to follow specific instruction and commands. It is also called as SFT (supervised fine-tuning).
        - It makes the models easier to use.
        - It makes the model to respond in a specific style
    2. RLHF (reinforcement learning with human feedback): method that fine-tunes the model using human preference to align generated behaviors with human values and intentions