Agentic AI

A Progression of Language Model Usage

4/2/2025

Outline

LM (Language Model) Overview

LM Usage

LM Usage Common Limitations

RAG (Retrieval Augmented Generation)

Tool usage

Agentic LMs

Agentic LM design patterns

Summary

Language model (LM)

LM (language model): *predicts* next word given input

- Input: text
- Output: next word prediction

Image source: https://web.stanford.edu/class/cs224n/

Language model (LM)

LM (language model): *predicts* next word given input

- Input: text
- Output: next word prediction

Image source: https://web.stanford.edu/class/cs224n/

Language model (LM)

LM (language model): predicts next word given input

- Input: text
- Output: next word prediction

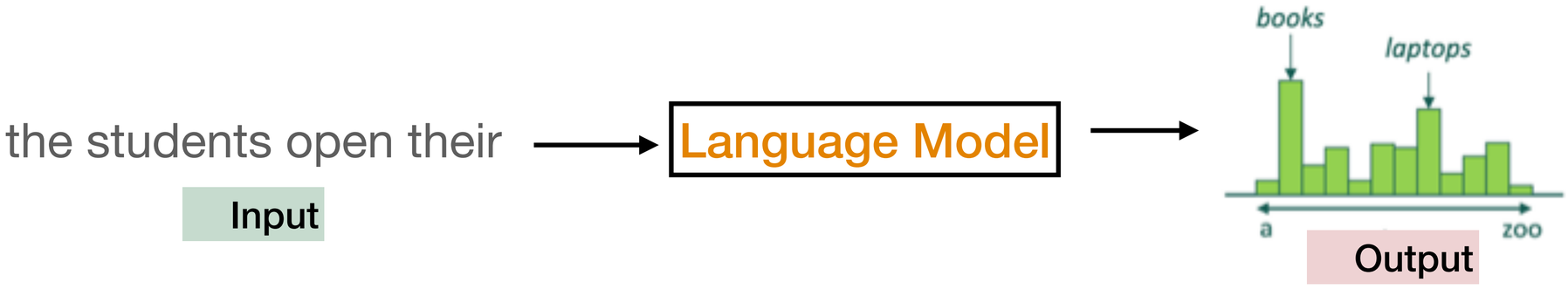


Image source: https://web.stanford.edu/class/cs224n/

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

A typical instruction dataset template

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

Instruction:

{instruction}

Input:

{input}

Response:

{response}

Input is optional

Note that there are many ways to prompt LM

LM Usage

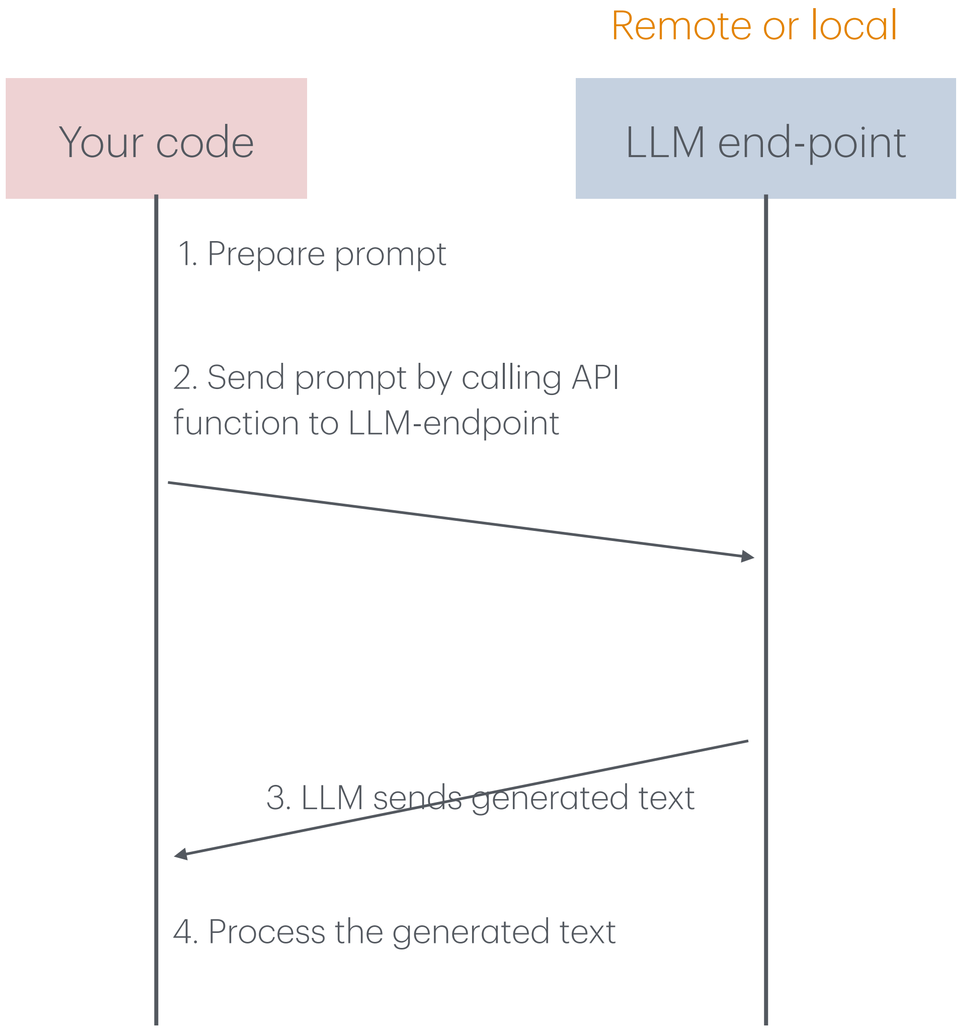
Current Status and Applications

- 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
- Various deployment options:
  - Cloud API providers
  - Local model deployments

LM Usage

Working with APIs

- Text generation via API endpoints
- Integration with service providers
- Implementation steps:
    1. Prepare prompts programmatically
    2. Make API calls
    3. Receive and process responses
    4. Iterate or chain with other prompts and tools as needed



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

Write clear, descriptive instructions

Write clear instructions:

LLMs cannot read your mind, so be more specific. The less the model has to guess at what you want, the more likely you'll get it.

Tactic: Include details in your query to get more relevant answers

In order to get a highly relevant response, make sure that requests provide any important details or context. Otherwise you are leaving it up to the model to guess what you mean.

Worse	Better
How do I add numbers in Excel?	How do I add up a row of dollar amounts in Excel? I want to do this automatically for a whole sheet of rows with all the totals ending up on the right in a column called "Total".
Who's president?	Who was the president of Mexico in 2021, and how frequently are elections held?
Write code to calculate the Fibonacci sequence.	Write a TypeScript function to efficiently calculate the Fibonacci sequence. Comment the code liberally to explain what each piece does and why it's written that way.
Summarize the meeting notes.	Summarize the meeting notes in a single paragraph. Then write a markdown list of the speakers and each of their key points. Finally, list the next steps or action items suggested by the speakers, if any.

Content credit: https://platform.openai.com/docs/guides/prompt-engineering

Include few-shot examples

Tactic: Provide examples

Providing general instructions that apply to all examples is generally more efficient than demonstrating all permutations of a task by example, but in some cases providing examples may be easier. For example, if you intend for the model to copy a particular style of responding to user queries which is difficult to describe explicitly. This is known as "few-shot" prompting.

SYSTEM	Answer in a consistent style.
USER	Teach me about patience.
ASSISTANT	The river that carves the deepest valley flows from a modest spring; the grandest symphony originates from a single note; the most intricate tapestry begins with a solitary thread.
USER	Teach me about the ocean.

Content credit: https://platform.openai.com/docs/guides/prompt-engineering

Provide relevant context (static or dynamic)

Provide reference text:

LLMs can generate fake answers for those they don't know. Provide reference text (context), like a cheat sheet for a student exam, LLMs will use reference instead of making things up

Tactic: Instruct the model to answer using a reference text

If we can provide a model with trusted information that is relevant to the current query, then we can instruct the model to use the provided information to compose its answer.

SYSTEM	Use the provided articles delimited by triple quotes to answer questions. If the answer cannot be found in the articles, write "I could not find an answer."
USER	<insert articles, each delimited by triple quotes> Question: <insert question here>

Content credit: https://platform.openai.com/docs/guides/prompt-engineering

Strategy: Give models time to "think"

Give the model time to “think”:
LLMs make more reasoning errors when asked answer right away instead of taking time to work it out. Ask for “chain of thought (CoT)” method before answering can help the model to reason toward correct answers

SYSTEM

Determine if the student's solution is correct or not.

USER

Problem Statement: I'm building a solar power installation and I need help working out the financials.
- Land costs $100 / square foot
- I can buy solar panels for $250 / square foot
- I negotiated a contract for maintenance that will cost me a flat $100k per year, and an additional $10 / square foot
What is the total cost for the first year of operations as a function of the number of square feet.

Student's Solution: Let x be the size of the installation in square feet.
1. Land cost: 100x
2. Solar panel cost: 250x
3. Maintenance cost: 100,000 + 100x
Total cost: 100x + 250x + 100,000 + 100x = 450x + 100,000

Model likely answer it is correct

Content credit: https://platform.openai.com/docs/guides/prompt-engineering

Strategy: Give models time to "think"

Strategy: Give models time to "think"

SYSTEM

First work out your own solution to the problem. Then compare your solution to the student's solution and evaluate if the student's solution is correct or not. Don't decide if the student's solution is correct until you have done the problem yourself.

USER

Problem Statement: I'm building a solar power installation and I need help working out the financials.
- Land costs $100 / square foot
- I can buy solar panels for $250 / square foot
- I negotiated a contract for maintenance that will cost me a flat $100k per year, and an additional $10 / square foot
What is the total cost for the first year of operations as a function of the number of square feet.

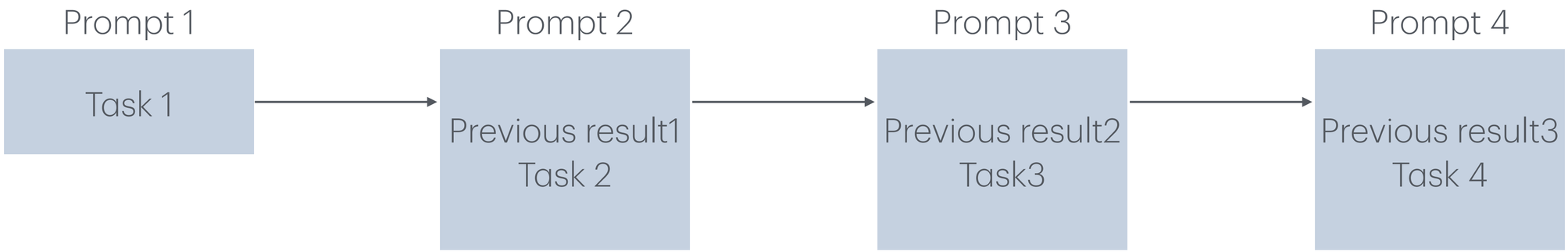
Student's Solution: Let x be the size of the installation in square feet.
1. Land cost: 100x
2. Solar panel cost: 250x
3. Maintenance cost: 100,000 + 100x
Total cost: 100x + 250x + 100,000 + 100x = 450x + 100,000

Model likely answer it is incorrect

Content credit: https://platform.openai.com/docs/guides/prompt-engineering

Break down complex tasks: chain complex prompts

- Split complex tasks into simpler subtasks, and chain them with separate prompts: Complex tasks result in higher errors. Decomposing a complex task into a set of simpler tasks is a good idea



Content credit: https://cs230.stanford.edu/syllabus/fall_2024/rag_agents.pdf

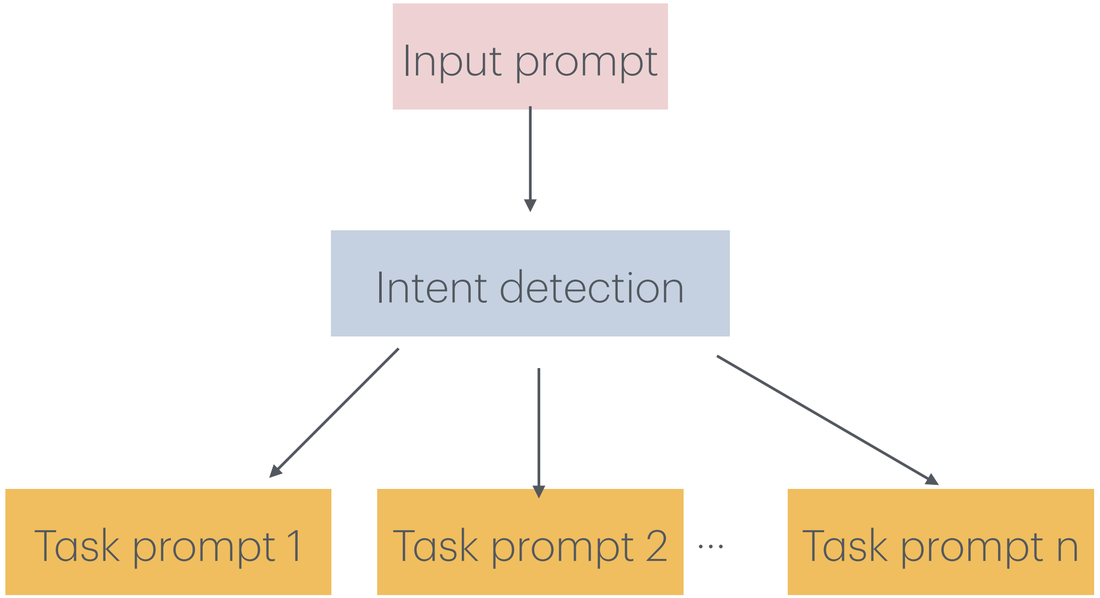
Systematic trace and evaluation of prompts for performance improvement

- Test changes systematically:
  Improving performance is easier if we can measure it. Prepare ways to evaluate systematically

Have evaluation plan early
Prepare evaluation data
LM as a judge together with human evaluation will be a good starting point

Intent Detection, routing

- Handling diverse tasks requiring different prompt strategies
- Benefits:
  - Determines appropriate prompt selection
  - Enables task decomposition
  - Facilitates LM model selection based on task type
- Helps manage complex workflows through systematic task breakdown
- Determine which different LMs to use depending on the detected intent



Prompt to identify intent:

Based on the intent use different prompt to follow up
It can be iterative calls to LLMs

It can be different LLMs based on the detected intent

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

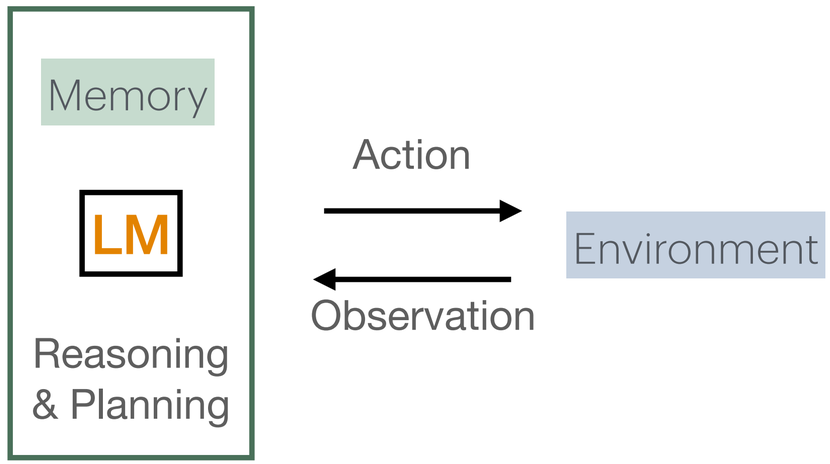
Ref: https://rdi.berkeley.edu/llm-agents-mooc/slides/intro.pdf

Agentic LMs

Interact with environments

- Simple LM use cases involve text in and text out

- Interact with environments

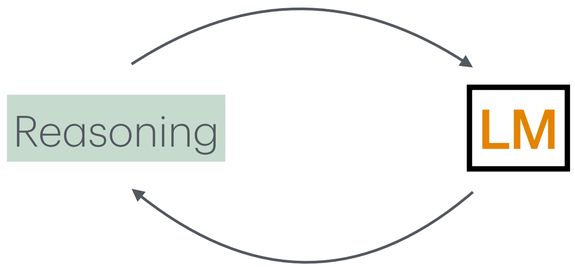


Ref: https://rdi.berkeley.edu/llm-agents-mooc/slides/intro.pdf

Agentic LMs

ReAct (Reason and action)

CoT (chain of thought)



Flexible and general to augment inference-time generation

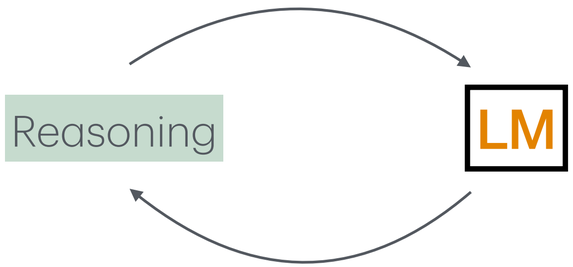
Lacking external information

Slide source: https://rdi.berkeley.edu/llm-agents-mooc/slides/llm_agent_history.pdf

Agentic LMs

ReAct (Reason and action)

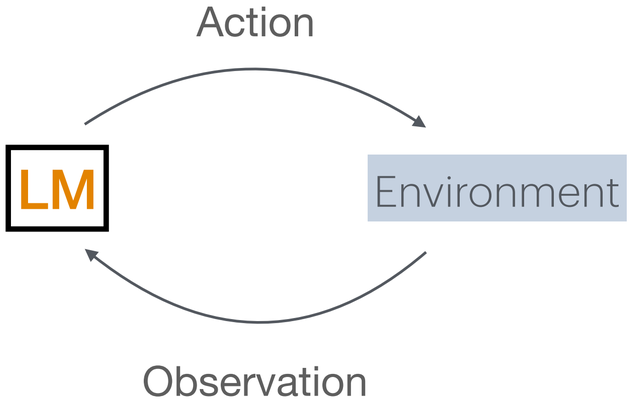
CoT (chain of thought)



Flexible and general to augment inference-time generation

Lacking external information

RAG/Retrieval/Tool use



Retrieval
Search engine
Calculator
Weather API
Python
API calls

Lacking reasoning

Flexible and general to augment knowledge, computation, feedback

Slide source: https://rdi.berkeley.edu/llm-agents-mooc/slides/llm_agent_history.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

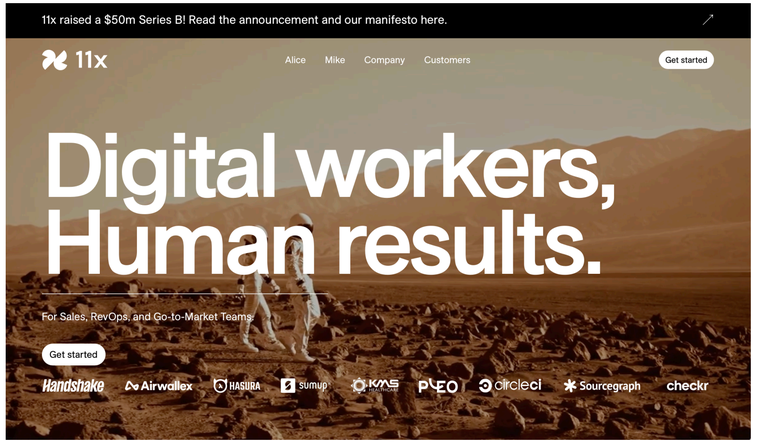
Ref: https://www.deeplearning.ai/the-batch/how-agents-can-improve-llm-performance/

Agentic LMs

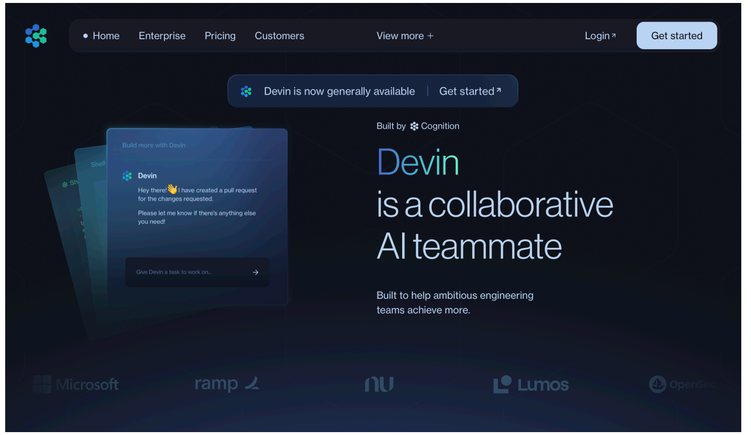
Real-world Applications

- 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

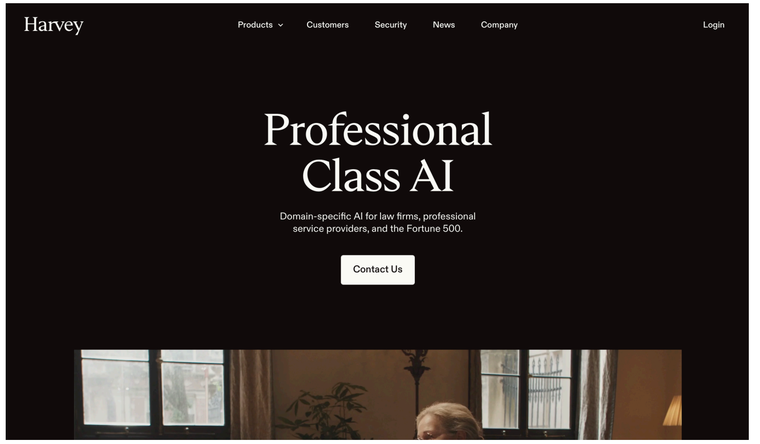
AI SDR



AI SW developer



AI lawyer



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

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/

Reflexion

- A pattern that's quick to implement and leads to good performance
- Example of fixing an existing codebase:
- Ask LM to provide feedback:
  - ```Here is the <code>. Check the code and provide constructive feedback```
- Then ask LM using the response:
  - ```Here is the <code> and <feedback>. Use the feedback and rewrite the code```
- Repeating the criticism and regeneration process can improve the output
- This self-reflection helps LMs find issues and improve their generation

*: prompt will be more elaborated, it is shorten only for the simplicity for presentation.

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

Tool usage

- Tool usage is a key design pattern:
  - Executes external tools for searching information, taking actions, such as *modifying* data
- Example: ```What is the best cafe based on user reviews?```
  - LM will generate ```{tool: web-search, query: "cafe reviews"}```
  - External tool searches and provides results to LM
- Example: ```If I invest $100 at 7% compound interest for 12 years, what will I have at the end?```
  - LM generates Python code: ```{tool: python-interpreter, code: "100 * (1+0.07)**12"}```
  - Code runs and produces the output

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

Multi-agent

Design a multi-agent system for smart home automation

- Climate control agent: Manages temperature and humidity based on setting and weather conditions
- Lighting control agent: Manages lighting based on schedule and time of day
- Security agent: Monitors cameras and alerts when there is a security breach
- Energy management agent: Monitors energy usage and suggests savings opportunities
- Entertainment agent: Manages TVs, audio, and streaming services based on user preferences
- Orchestration agent: Coordinates all agents to ensure smooth operation

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

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
- Continuous evolution of patterns and capabilities

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)

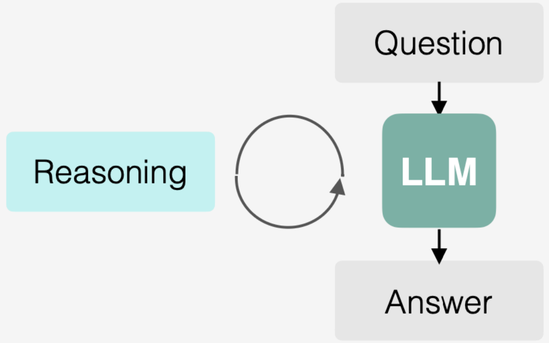
8.[Frontiers-of-AI-Agents-Tutorial](https://frontiers-of-ai-agents-tutorial.github.io/)

Backup materials

Agentic LMs

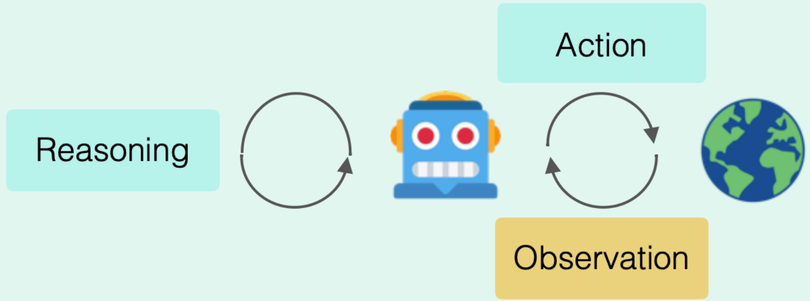
ReAct (Reason and action)

Reasoning (update internal belief)



Acting (obtain external feedback)

ReAct: a new paradigm of agents that **reason and act**



- **Synergy** of reasoning and acting
- **Simple** and intuitive to use
- **General** across domains

Slide is from: link

Fine-tuning LLMs

Further train closed or open source LLMs since you need a custom model

- Continuous pre-training: train the model the prepared text using next token prediction
- Supervised fine-tuning (SFT): train the model with Q&A pair

Both closed and open LLMs can be used for this approach

Why consider custom LLMs

Data residency and privacy

Intellectual property

Customization

Cost

Performance

Latency

Control over updates

Ownership

Transparency

Playbook for LLM-based apps

Playbook for LLM-based applications

Identify use case(s), define clean inputs and outputs of the case

Create and test prompt using playground. Use descent models, such as gpt-35-turbo, gpt-4o

Think about how to measure the generated output. Create some test cases

Once you determine LLM can help your use case using simple testing, then move to the next step and use API calls to automate development.

Write python scripts that can automate the experiment, test and evaluation

Use a descent model API endpoint, or use local LLMs API endpoint depends on the task complexity

Evaluation the results and iterate

Depends on the task complexity and task nature, consider using external tools, which includes embedding based retrieval or calling external APIs to get more information.

Intent detection and iterative generation can improve output performance, and methods used in agentic LLMs can be helpful for certain types of task.

For work related text (code, documentation) be sure to use LLMs that are approved by the company. The reason is that we do not want to send work related text to externally hosted LLMs, which might be used for training and be read.

Write clear instructions

- Include details in your query to get more relevant answers
- Ask the model to adopt a persona
- Use delimiters to clearly indicate distinct parts of the input
- Specify the steps required to complete a task
- Provide examples
- Specify the desired length of the output

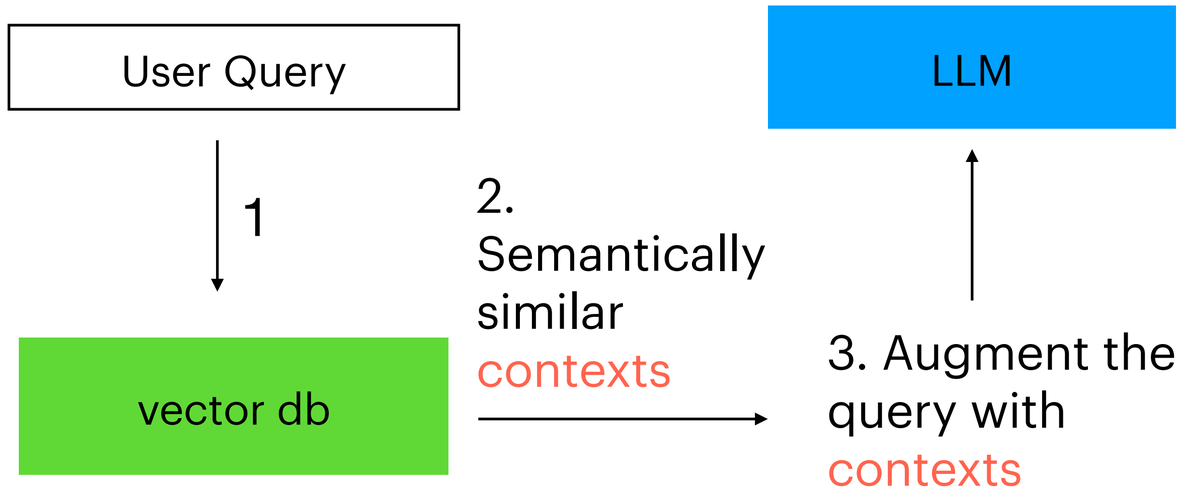
Content credit: link

Retrieval augmented generation (RAG)

Overview

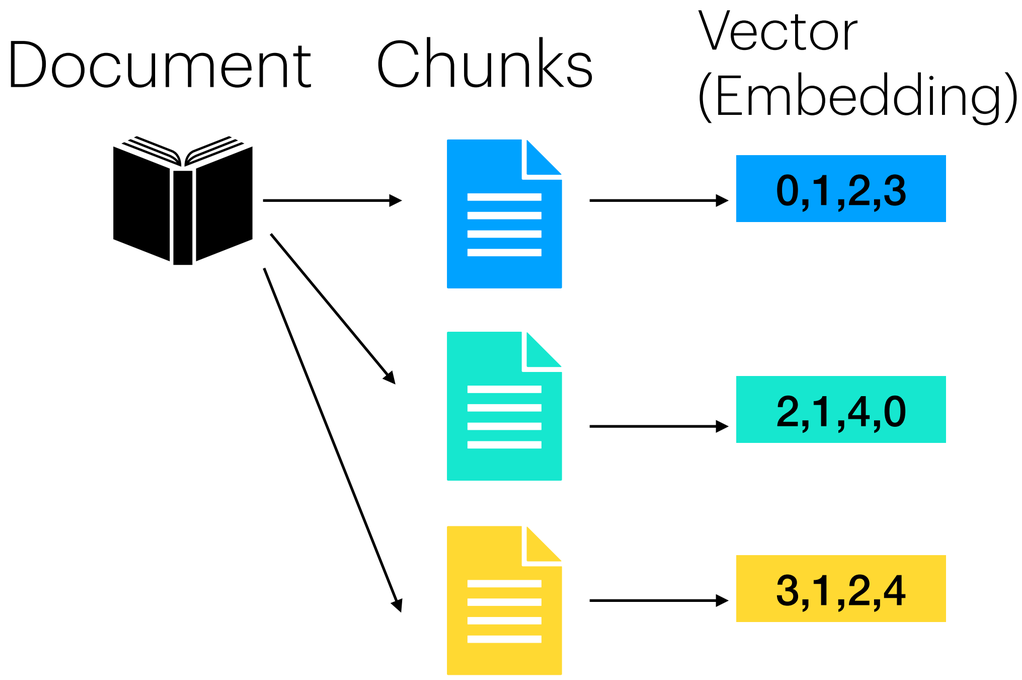
- Retrieval augmented system augments a query with semantically related text from vector db

- Two components
  - Vector database
  - LLM



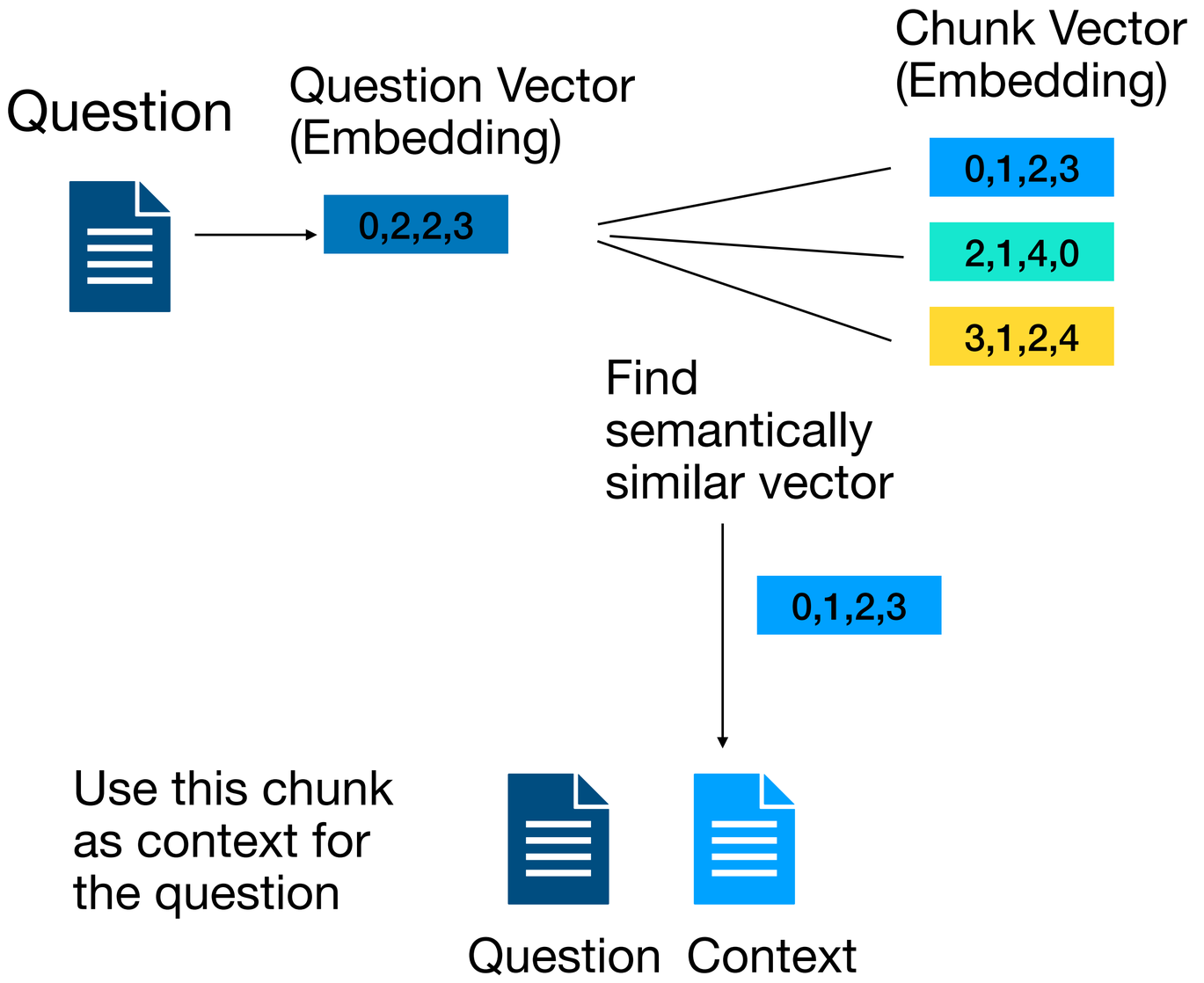
Indexing (turn documents meaningful vector)

- Chunk document to small chunk
- Convert text chunks to vectors
  - A dedicated model does this
- Store text chunks and vectors (vector dbs are used to store and provide search)
- We do this off-line



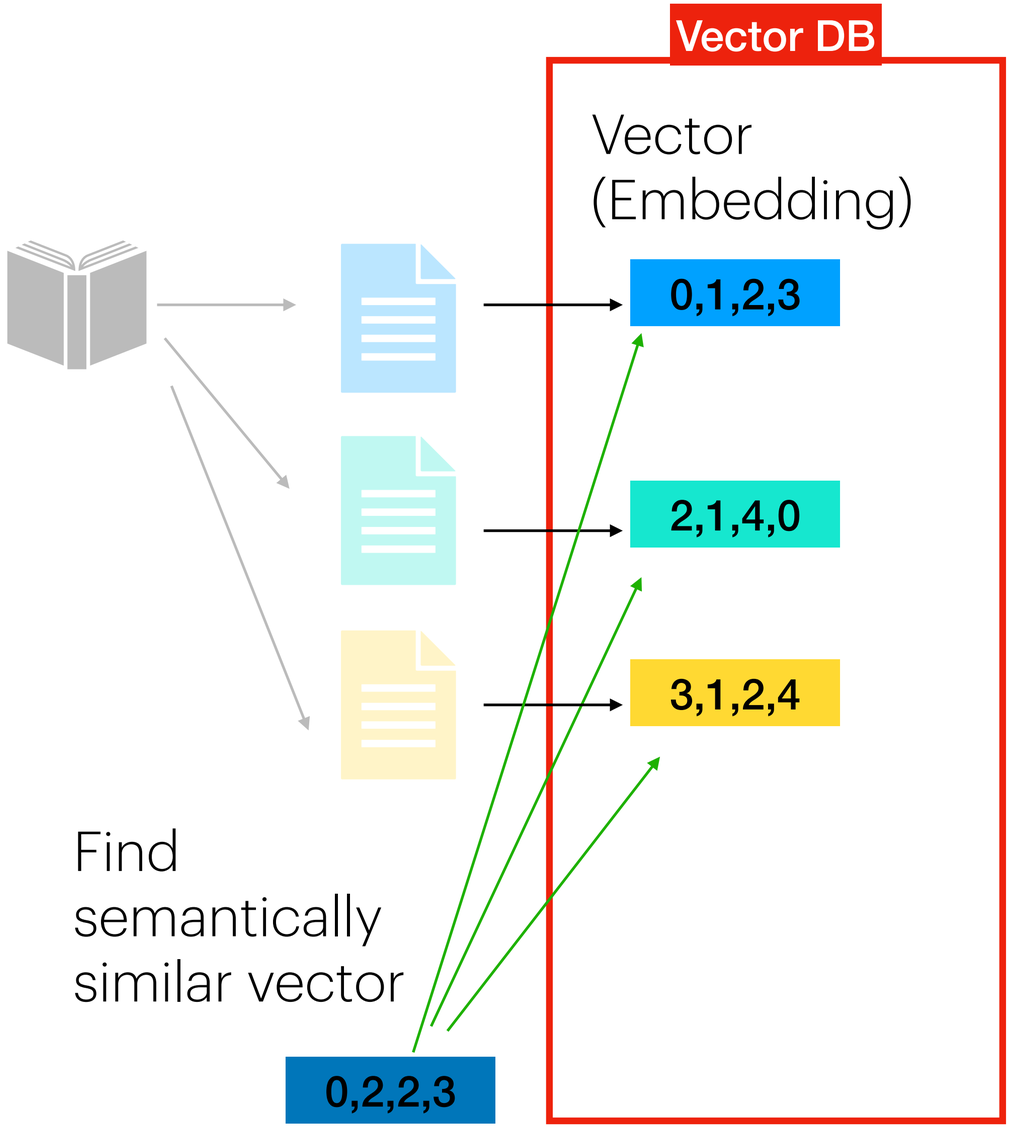
Similarity search

- Convert the question to vector
- Compare the question vector and the chunk vectors
- Find similar chunk using cosine similarity (measure angles of the vectors)
  - Select top-k similar chunks
  - In the example (right), we use top-1 chunk (k=1)
- Use the similar chunk as context with the question



Vector DB

- Manages large dimension vectors
- Find a similar vectors based measures (cosine similarity or dot product)



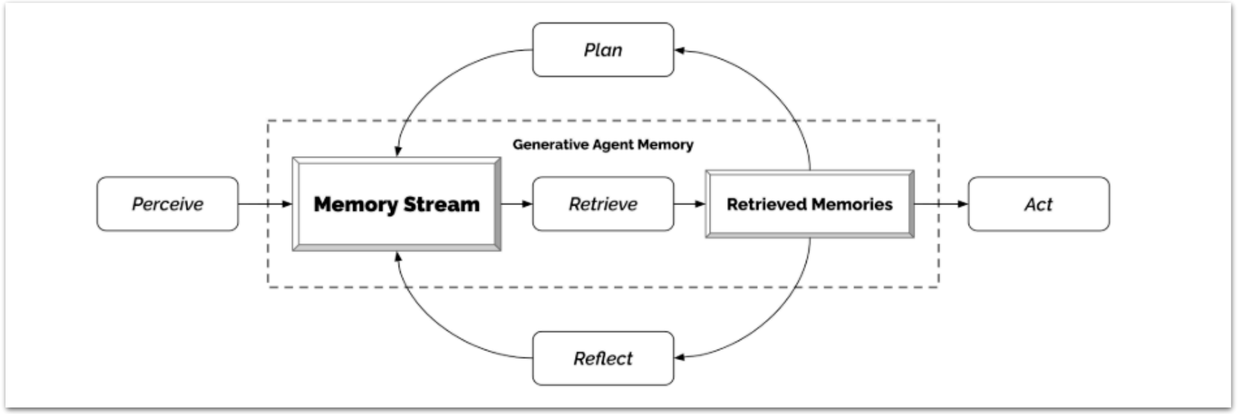
Design considerations for retrieval based system

- Hybrid search: combine semantic and syntactic (traditional) search
- Rerank: find coarse grain match, then apply fine grain re-ranking
- Use meta data to help filter out the semantic search results
- Various ways to pre and post process prompts
- Evaluation is important. Evaluate retrieval and final generation using prepared dataset.

Agentic LLM

Multi agent example

- Examples:
  - Generative agent (simulated village)



Each agent is backed by LLM with memory

https://arxiv.org/pdf/2304.03442

Agentic LLM

SW Development Agent

- Examples:
  - Devin
  - And similar tools

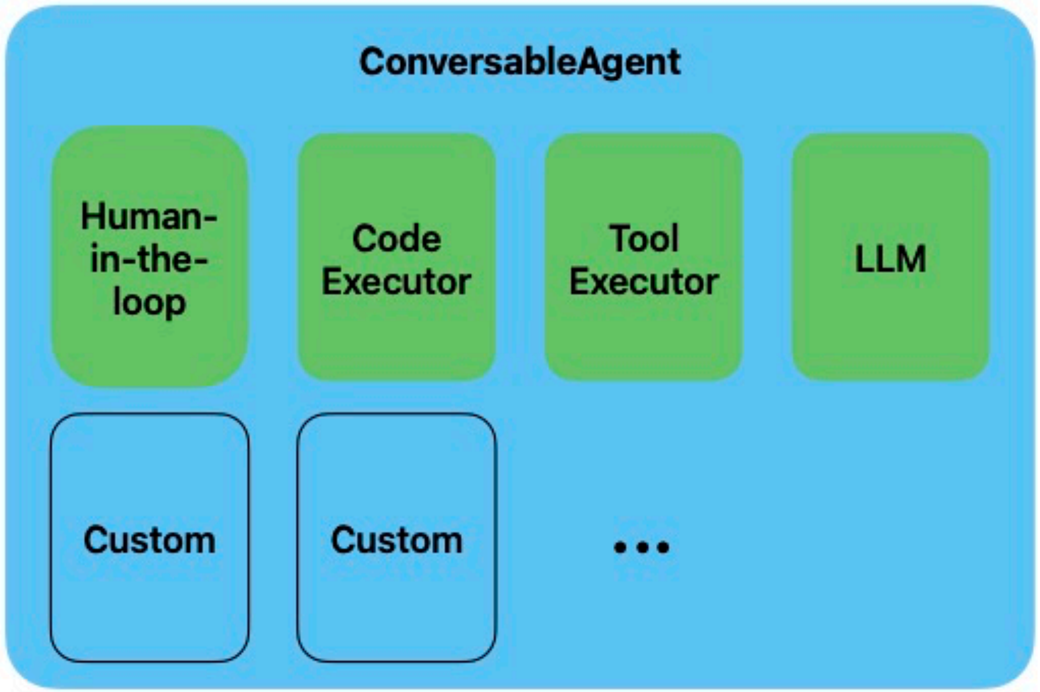
Agentic LLM

- Examples:
  - Github copilot workspace (new github feature)

Gh workspace test repo: https://github.com/insop/lockdep/issues

SW framework

- Many OSS SW framework: AutoGen, MetaGPT, ChatDev, SWE-agent (open source version of Devin)



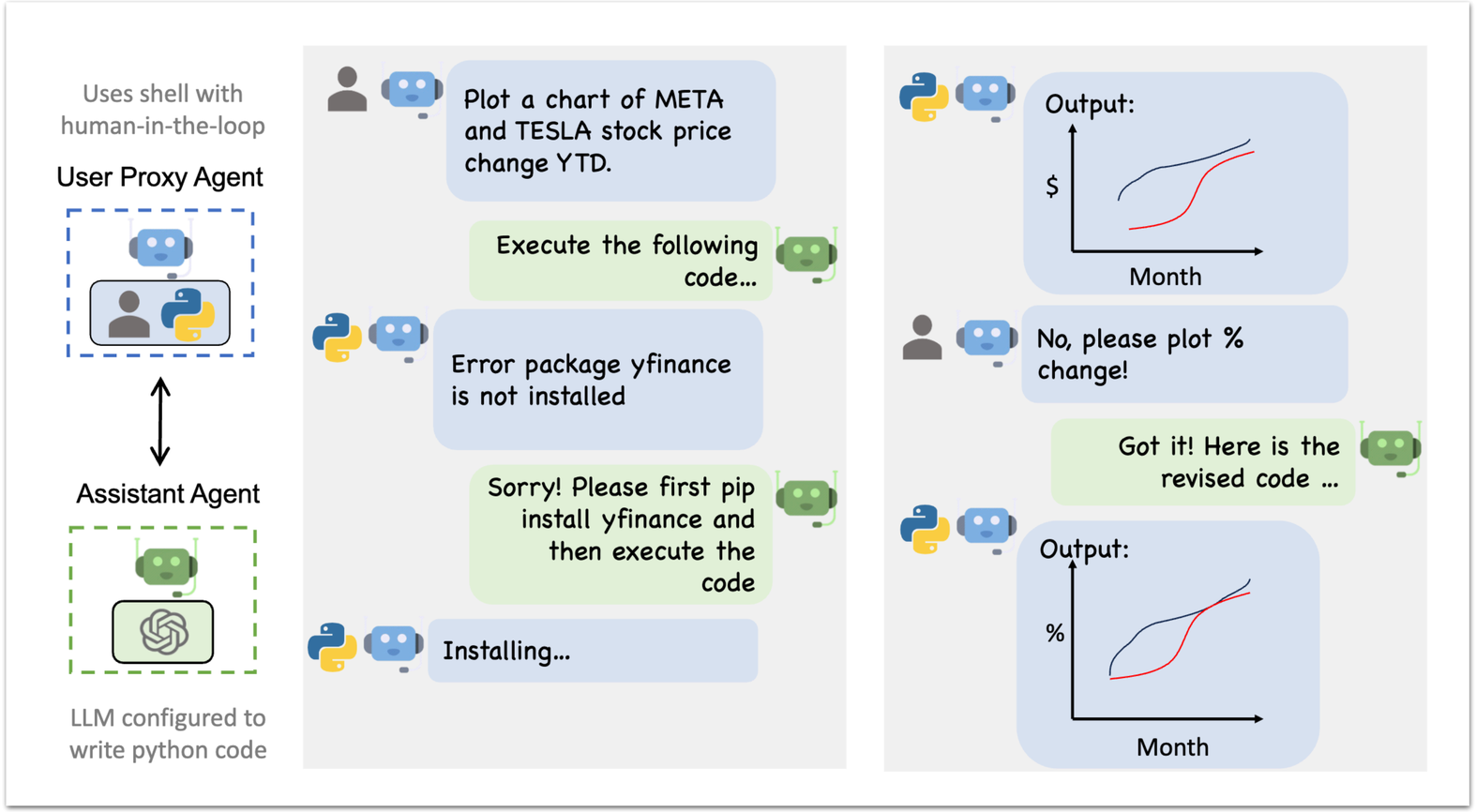
From https://microsoft.github.io/autogen/docs/Getting-Started

From From https://microsoft.github.io/autogen/docs/Getting-Started

SW framework

AutoGen

- Autogen is a multi-agent framework
  - An agent is backed by LLM, code executers, can send & receive messages to and from other agents



From https://microsoft.github.io/autogen/docs/Getting-Started

From From https://microsoft.github.io/autogen/docs/Getting-Started

What is a Large Language Model (LLM)?

- An LLM is a model takes text input and generates a high probability continuation of that text output
- The probabilities are trained form massive quantities of text data
- The probabilities are learned from highly abstract matrix operations (self-attention):
  - No symbolic computer code
  - No interpretable database storage
  - No analytical control over behaviors
- It is a prediction mechanism for next text, which is a language model (LM) shown in previous slides.

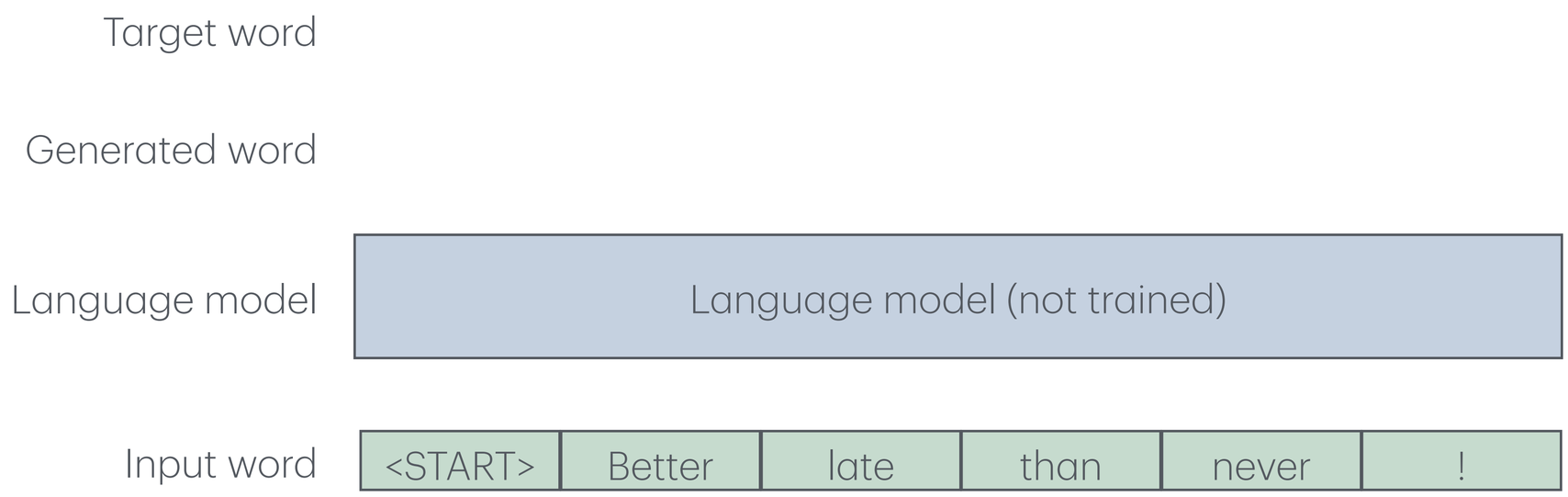
Ref: [4]

Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's *self-supervised* training because there are no explicit labels, but input text is used as both input and target label.



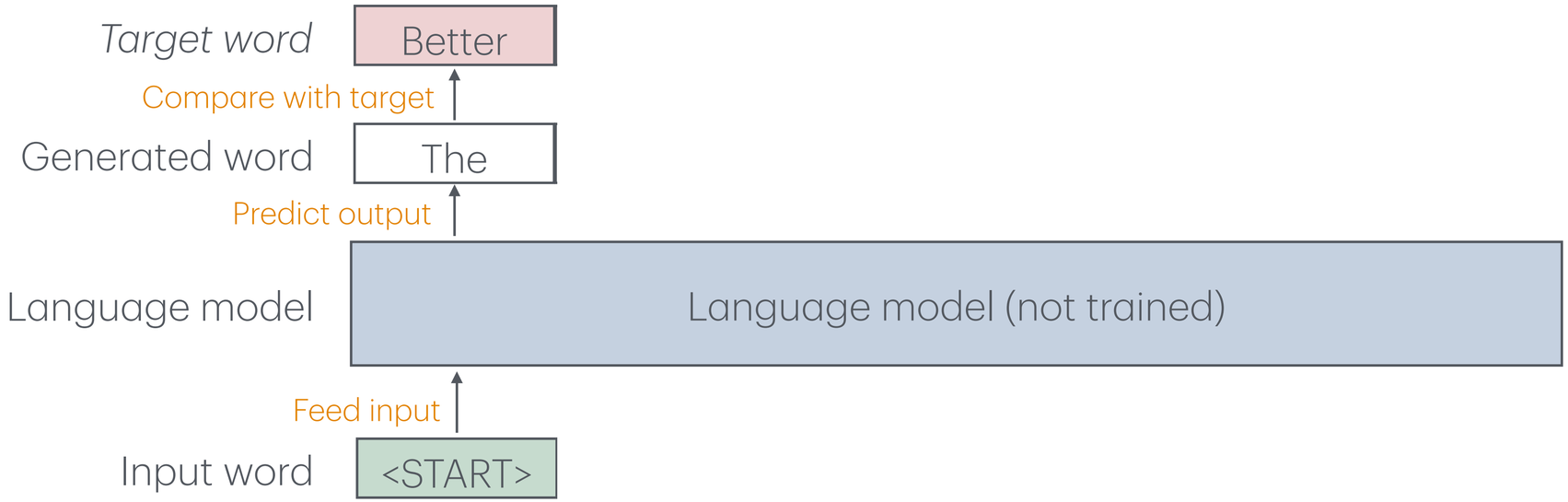
Ref: [4]

Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.



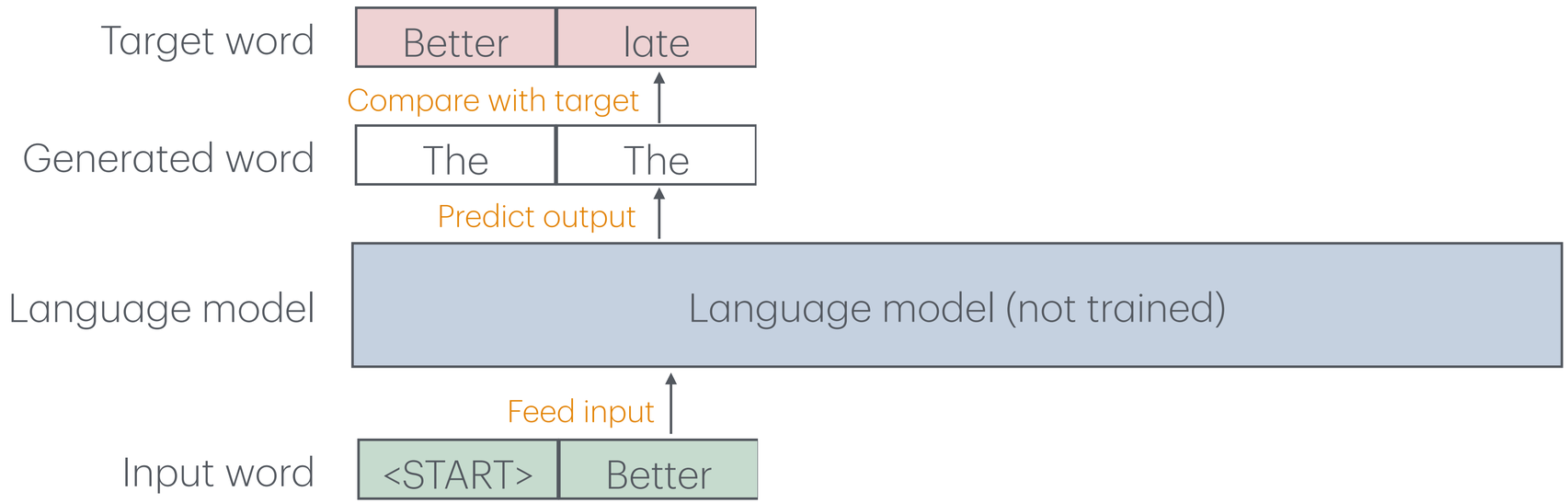
Ref: [4]

Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.



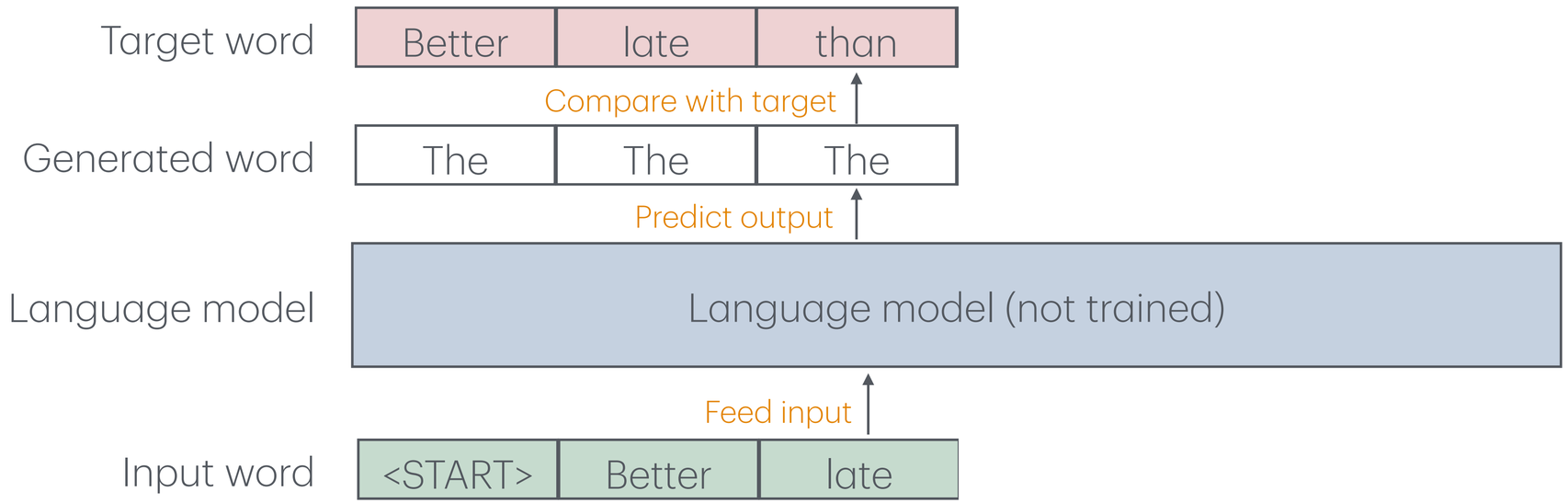
Ref: [4]

Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.



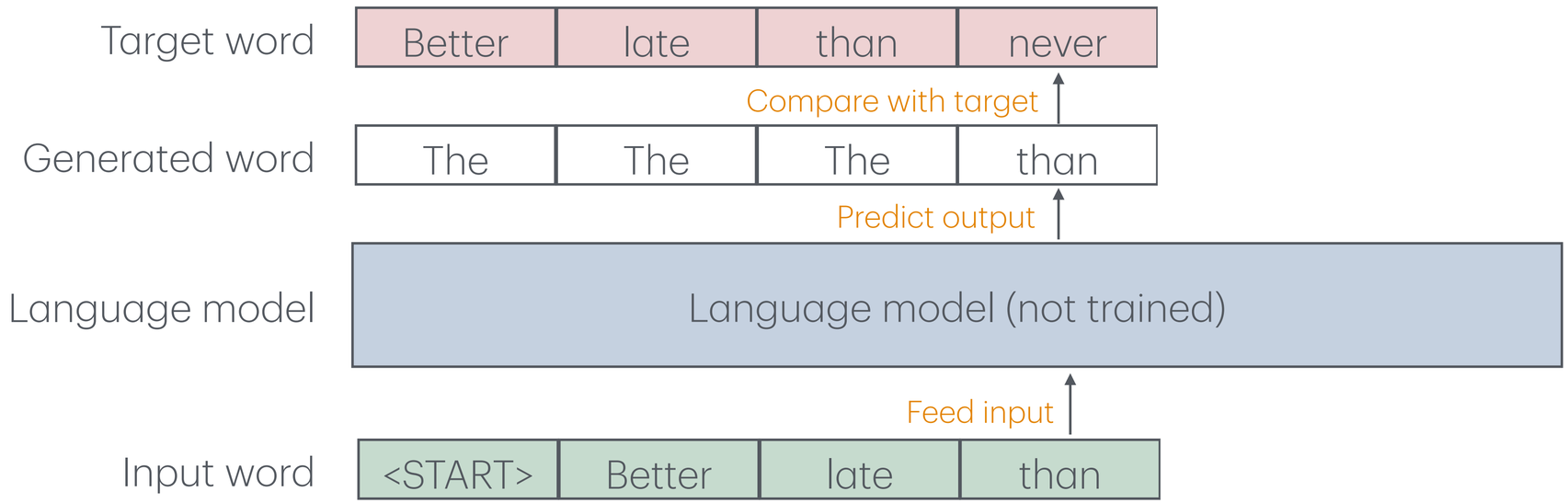
Ref: [4]

Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.

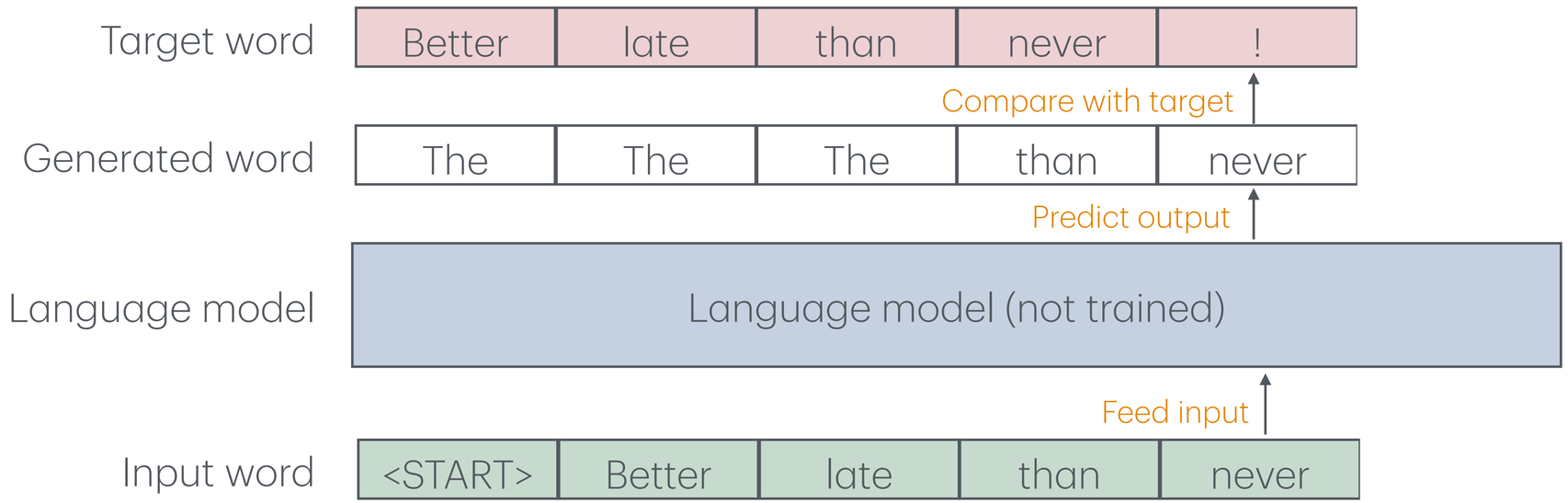


Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.

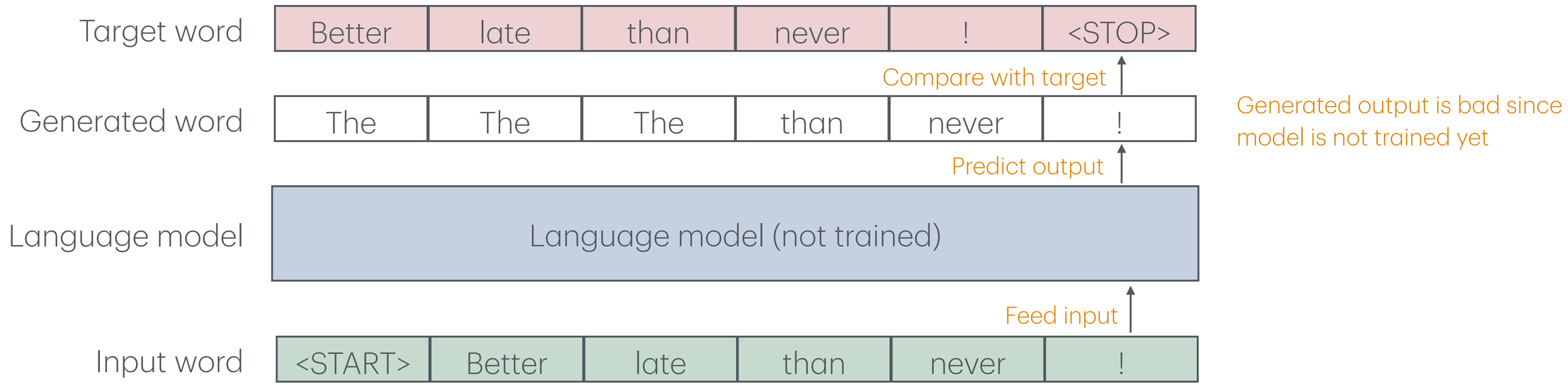


Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.



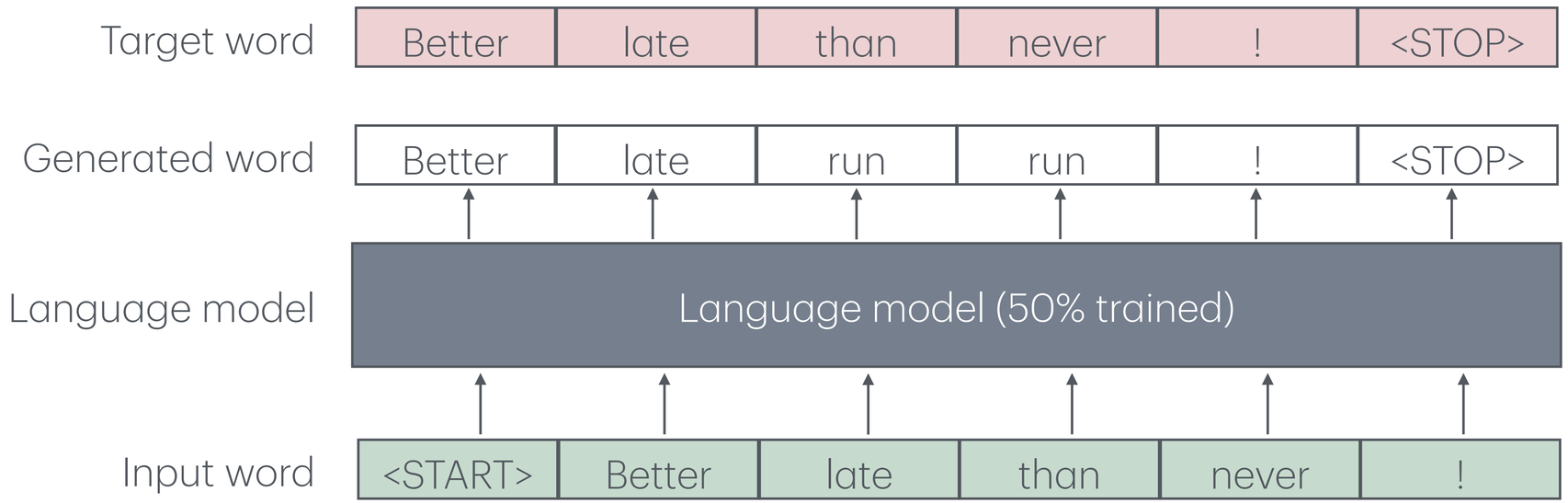
Ref: [4]

Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.



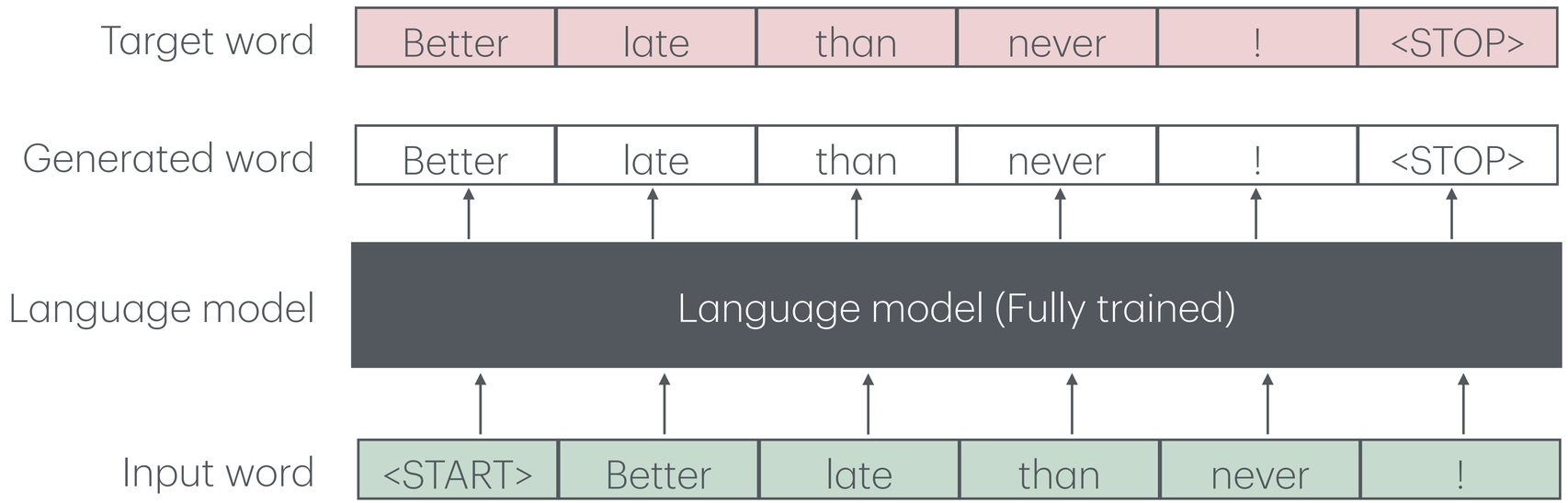
Ref: [4]

Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

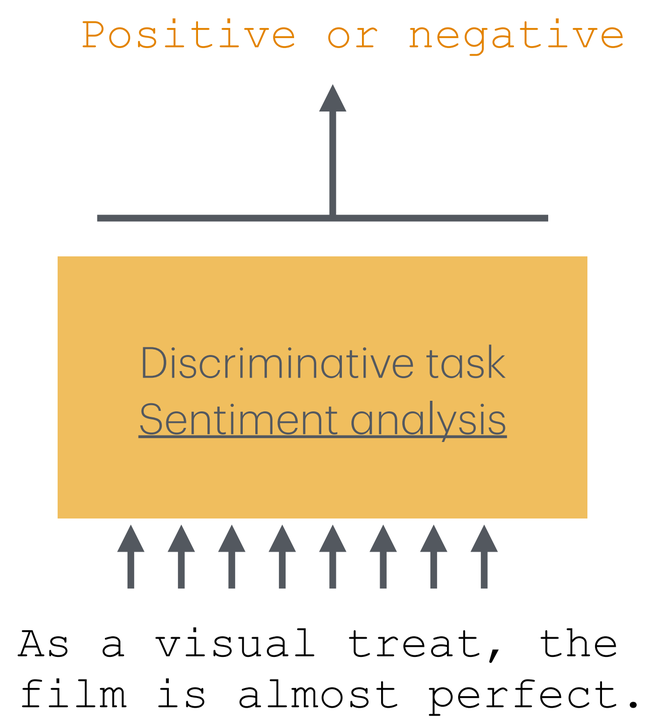
It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.

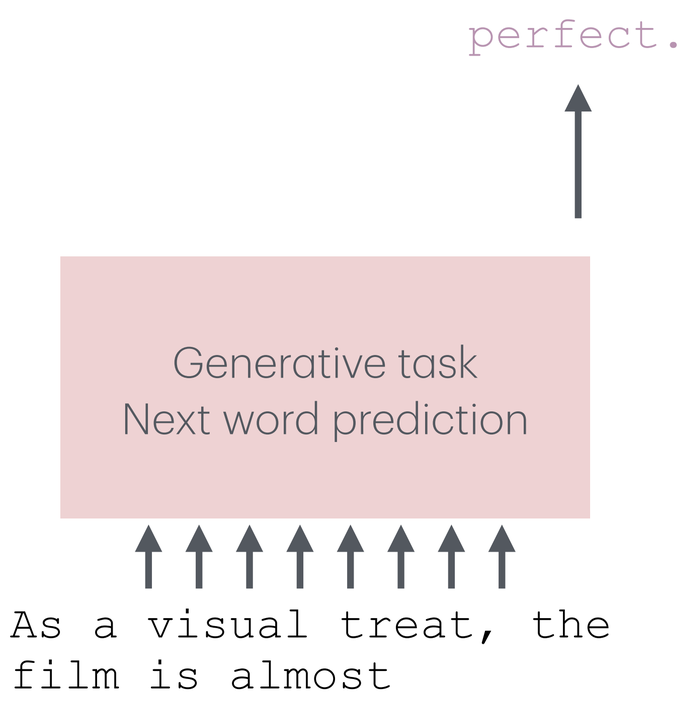


Ref: [4]

Discriminative vs Generative tasks in NLP

- Discriminative tasks: classifying input, e.g. sentiment analysis, text classification
- Generative tasks: generate text, language model, summarization, machine translation





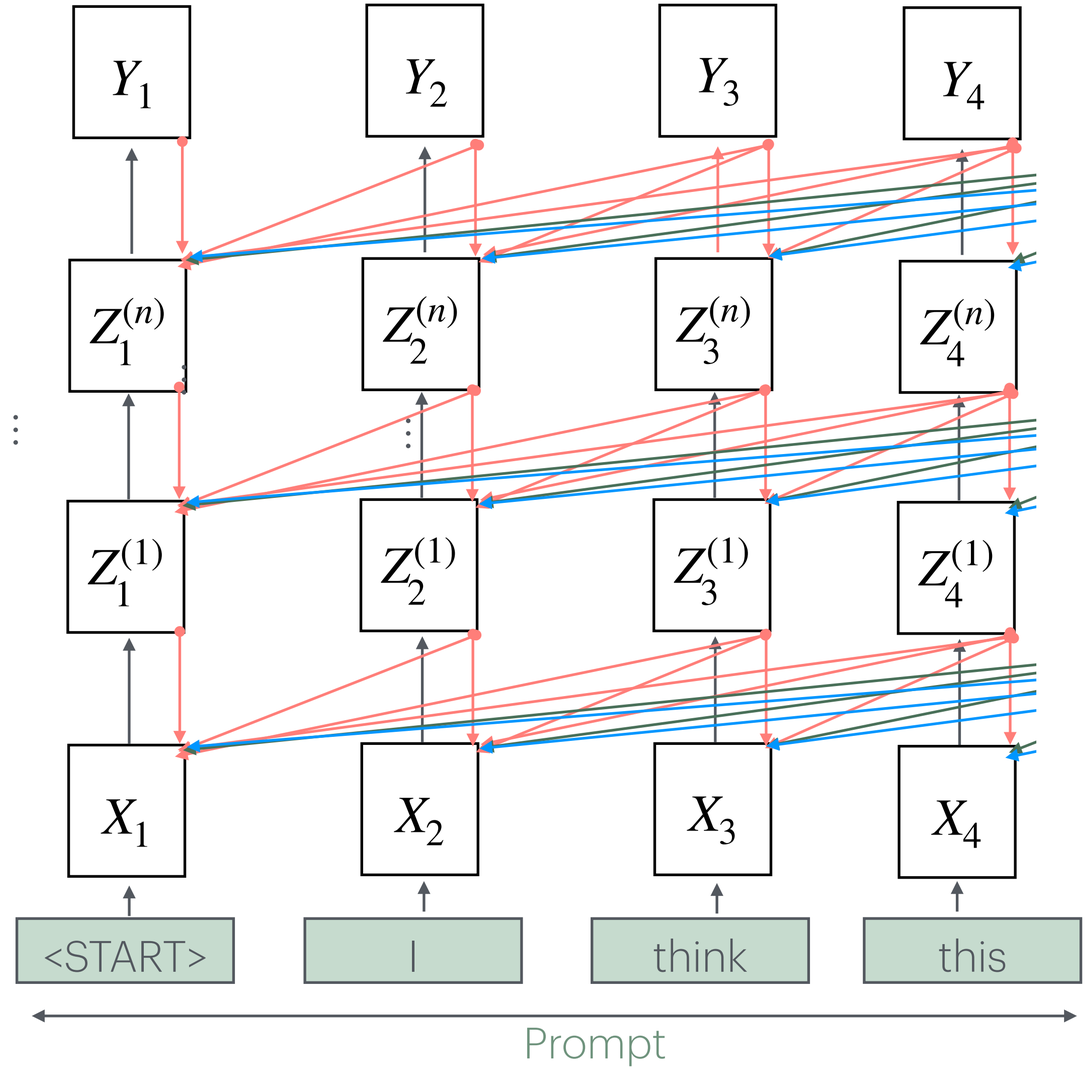
Note that in generation of LM, there are two types of generations, i) prompt phase that takes input at once and process together, ii) text generation phase process one token at a time as autoregressive way

Adapted from : link

Attention Review

Language model is built with transformer blocks

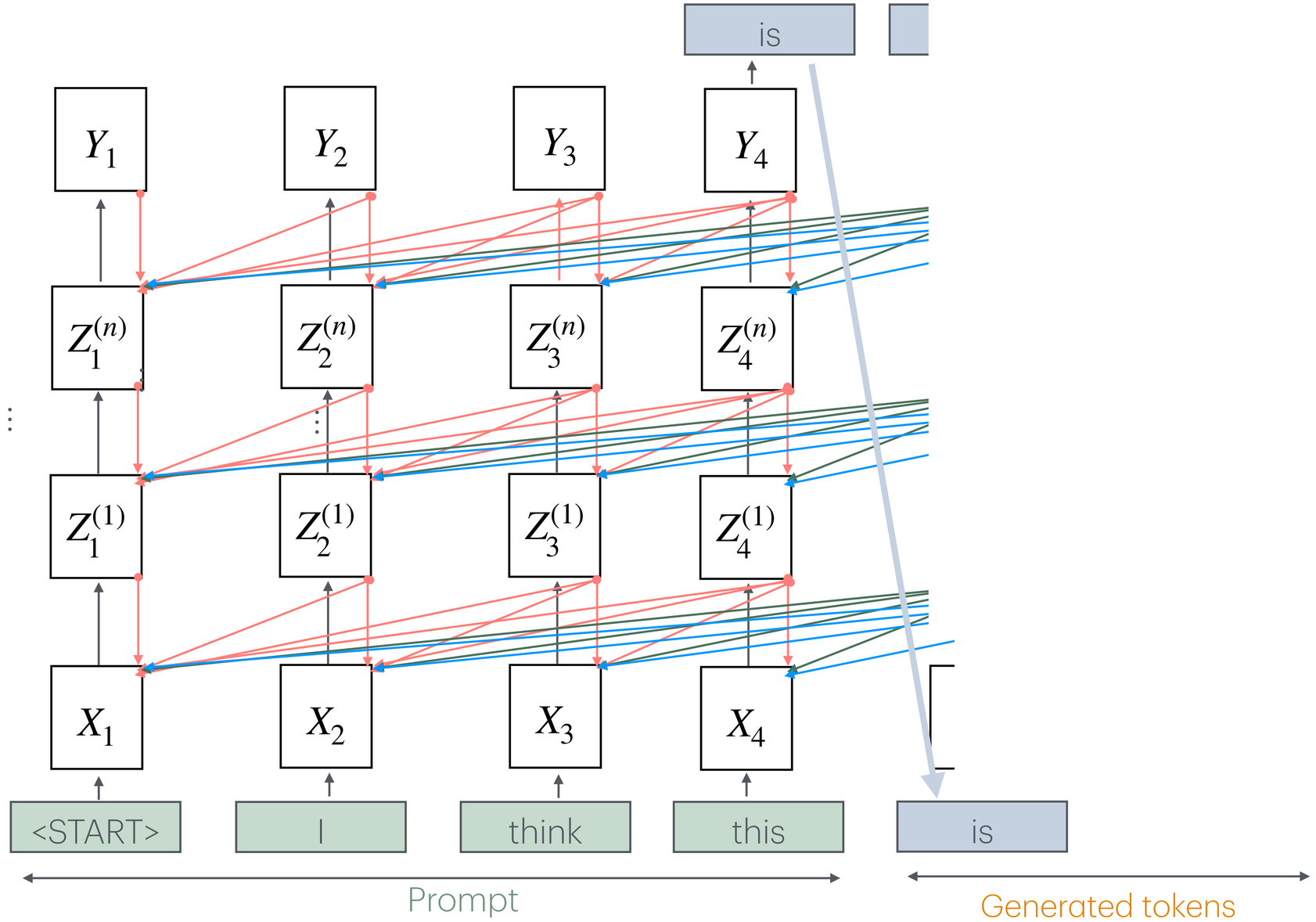
$Z_t^{(i+1)} = \text{Transformer\_block}(Z_t^{(i)})$



Attention Review

Language model is built with transformer blocks

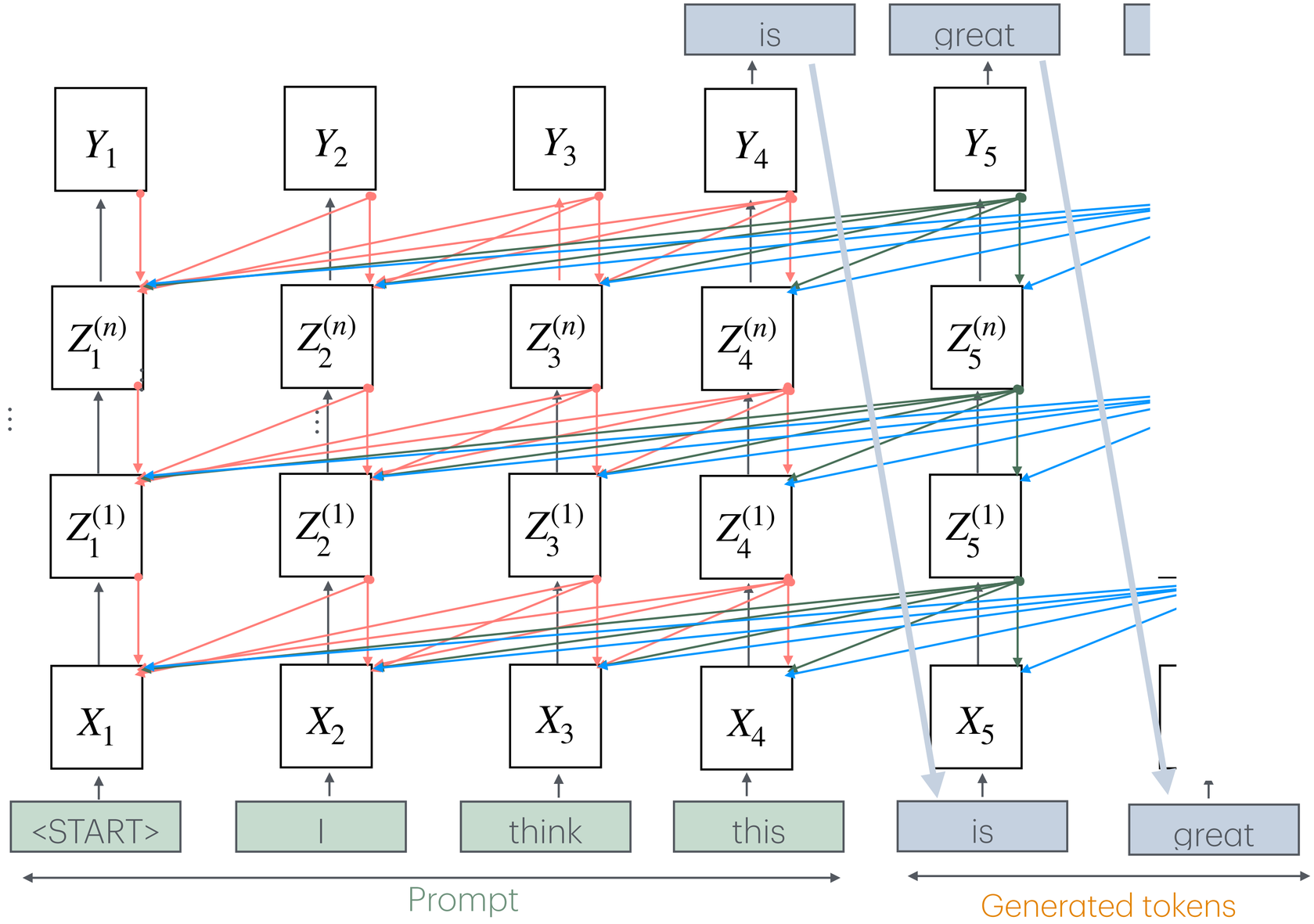
$Z_t^{(i+1)} = \text{Transformer\_block}(Z_t^{(i)})$



Attention Review

Language model is built with transformer blocks

$Z_t^{(i+1)} = \text{Transformer\_block}(Z_t^{(i)})$



Attention Review

Language model is built with transformer blocks

$Z_t^{(i+1)} = \text{Transformer\_block}(Z_t^{(i)})$