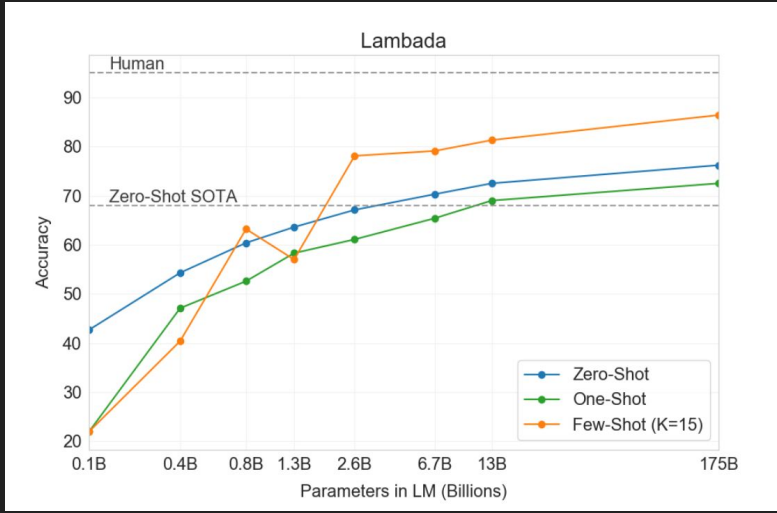
CMPE492 - Week 2

Zeynep Buse Aydın - Asım Dağ

Zero-Shot, One-Shot, Few-Shot Prompting

- As discussed in OpenAI's article `Language Models are Few-Shot Learners`
- `Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm` argues that, GPT-3 is often not actually learning the task during run time from few-shot examples. Rather than instruction, the method's primary function is task location in the model's existing space of learned tasks.
- alternative prompts which, with no examples or instruction, can elicit comparable or superior performance to the few-shot format



Prompt	Babbage / 6.7B	Curie / 13B
OpenAI 0-shot	15.5	22.4
OpenAI 1-shot	31.6	31.4
OpenAI 64-shot	36.4	38.3
Reproduced OpenAI 0-shot	15.9	18.7
Reproduced OpenAI 1-shot	21.8	24.1
Reproduced OpenAI 10-shot	25.1	27.9
Simple colon 0-shot	23.5	33.3
Simple colon 1-shot	18.0	27.6
Simple colon 10-shot	24.1	33.4
Master translator 0-shot	26.5	32.9

the example is interpreted as part of a consecutive narrative

Prompt Programming

- programming in natural language
- anthropomorphic approach

Direct task specification: constructing the signifier: providing a clear instruction or signifier, 0-shot

Task specification by demonstration: few-shot, with examples

Task specification by memetic proxy: ex. Summarize like Mahatma Gandhi

Prompt programming as constraining behavior: not Translate French to English:, but Translate this sentence to English:, and adding quotation marks

Serializing reasoning for closed-ended questions: For tasks requiring reasoning, this approach involves breaking down problems into smaller, tractable sub-tasks. (math problems)

Metaprompt programming: automated methods to generate task-specific prompts, “Let's solve this problem by splitting it into steps.”, “This problem asks us to”

Instruction-Based Prompting

- Most basic form of prompting: ‘do this, explain that, write those, give examples’
- **Clarity is Key**: The most effective instructional prompts are clear and unambiguous. The goal is to eliminate confusion and guide the LLM to the exact type of response we need. For instance, instead of saying, “*Tell me about healthy foods,*” be specific: “*List five fruits that are high in Vitamin C.*”
- **Provide Context:** Context helps the LLM understand the prompt better, especially if it relates to a previous conversation or a specific scenario. For example, “*Given that I have a gluten allergy, suggest three gluten-free breakfast options.*”
- **Define the Scope and Format:** If you’re looking for information in a specific format, include this in your prompt. For example, “*Explain the process of photosynthesis in a five-step list,*” or “*Describe the benefits of meditation in a short paragraph.*”
- **Utilize Closed-Ended Instructions for Specific Answers:** When you need precise information, frame your prompt to narrow down the AI’s focus. For example, “*Compare the climates of California and Florida in a few sentences.*”

Chain-of-Thought Prompting

- Enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks.
- Instead of fine-tuning a separate language model checkpoint for each new task, one can simply “prompt” the model with a few input–output exemplars demonstrating the task. A prompting only approach is important because it does not require a large training dataset and because a single model checkpoint can perform many tasks without loss of generality.
- Chain-of-thought prompting does not positively impact performance for small models, and only yields performance gains when used with **models of ~100B parameters**. Models of smaller scale produce fluent but illogical chains of thought, leading to lower performance than standard prompting.
- Chain-of-thought prompting has larger performance gains for **more-complicated problems**.
- Variable computation by itself is not the reason for the success of chain- of-thought prompting
- Successful use of chain of thought does not depend on a particular linguistic style
- **in-domain test:** set for which examples had the same number of steps as the training/few-shot exemplars
  out-of-domain (OOD) test set: for which evaluation examples had more steps than those in the exemplars

Self-Consistency Prompting

- Works by generating multiple diverse reasoning paths for a single query and selecting the most consistent or frequent answer across the variations
- This method increases the confidence of output by mitigating the risk of over-reliance on any single generated reasoning chain
- Self-consistency can be optimally applied in several ways, depending on the task:
    - **Chain-of-Thought Reasoning Tasks**: In tasks that require multi-step reasoning (e.g., math problems or logical deductions), generating multiple chains of thought and selecting the most frequent conclusion increases the robustness of the model's responses .
    - **Few-shot Learning**: Self-consistency can complement few-shot learning setups. Brown et al. (2020) showed that few-shot prompting allows models to generalize with minimal examples, but integrating self-consistency can ensure that generated answers across few-shot examples are reliable by selecting the most common result across independent trials .
    - **Instruction-based Generalization**: Self-consistency enhances performance in tasks that involve following natural language instructions. By testing multiple interpretations of instructions and selecting the most common output, models can more effectively generalize across tasks, as highlighted by Mishra et al. (2021) .

Dynamic Prompting

Dynamic Prompting enables LLMs to adapt to changing conversational contexts by **updating** or **modifying** the input instructions **based on the history of the interaction**.

Brown et al. (2020) demonstrated how LLMs can generalize across tasks using few-shot learning, but dynamic prompting goes beyond this by modifying the input in real time, allowing for more personalized and context-aware responses in multi-turn dialogues.

Mishra et al. (2021) emphasized that dynamic prompts, often derived from user instructions, help models better generalize across tasks, improving performance in conversational settings where instructions and dialogue context evolve dynamically.

Trade-offs Between Static and Dynamic Prompting

- **Static Prompting**: In static prompting, the prompt remains fixed throughout a conversation or task.
    - This reduces computational overhead since the same prompt is reused for multiple queries.
- However, the downside is that static prompts lack flexibility and adaptability, leading to a decline in response quality in dynamic contexts or multi-turn dialogues.
- Reynolds & McDonell (2021) argue that static prompts are often less suited for complex, evolving tasks, as they don't account for new information introduced during a conversation.

- **Dynamic Prompting**: In contrast, dynamic prompting continuously updates the model's input based on the current state of the conversation, providing better contextual understanding and allowing the model to respond more accurately as the dialogue evolves.
- While dynamic prompting results in better response quality, it comes at a higher computational cost.
    - Generating new prompts in real time requires additional processing, particularly in tasks involving long dialogues or multi-step reasoning, as highlighted by Zhao et al. (2021).
    - This trade-off between higher response quality and computational cost is a key consideration when choosing between static and dynamic approaches.

Automated Prompt Generation (AutoPrompt)

- an automated method to create prompts for a diverse set of tasks
- gradient-based optimization
- potentially a replacement for fine-tuning
- outperforms manually-generated prompts while requiring less human effort
- computationally expensive

Prefix-Tuning

- Fine-tuning requires updating and storing all the parameters of the LM.
- **Lightweight fine-tuning = prefix-tuning.** Freezes most of the pretrained parameters and modifies the pretrained model with small trainable modules. Stores 1000x fewer parameters than fine-tuning.
- AutoPrompt searches for a sequence of discrete trigger words and concatenates it with each input to elicit sentiment or factual knowledge from a masked LM. In contrast with AutoPrompt, prefix-tuning optimizes continuous prefixes, which are more expressive; moreover, it focuses on language generation tasks.
- Has a comparative advantage when the number of training examples is smaller. Can maintain a comparable performance in a full data setting and outperforms fine-tuning in both low-data and extrapolation settings.
- effective in table-to-text generation, not very effective in summarization. On both table-to-text and summarization, prefix-tuning has better **extrapolation** than fine-tuning
- advantageous when there are a large number of tasks that needs to be trained independently.

Role-Playing or Persona-Based Prompting

- In real-time applications, persona-based prompting helps LLMs take on various identities or roles, providing tailored responses based on the simulated persona.
- Persona-based prompting works in a similar way by using examples and role-specific instructions to guide how the model responds.
    - For instance, a model might be prompted to respond as a knowledgeable tutor or a compassionate therapist, improving user engagement by creating a sense of conversational authenticity.

Challenges in Maintaining Consistency in Long Dialogues

- One of the significant challenges in persona-based prompting is maintaining consistency and authenticity over long conversations.
- As dialogues extend, the model must remember key details about the persona and the ongoing conversation to avoid breaking character.
- Over time, inconsistencies can emerge, especially if the dialogue becomes complex or spans multiple topics, which can undermine the user’s trust in the AI's role.
    - **Memory and Contextual Drift**: A key limitation is the model's short-term memory,
        - Mishra et al. (2021) discussed cross-task generalization and the difficulty of retaining consistency across tasks in open-ended scenarios. Similarly, in persona-based interactions, maintaining the same voice or character across turns and tasks presents challenges, especially when the user brings in new or unexpected inputs.
    - **Balancing Flexibility and Stability**: Balancing the flexibility needed to respond to new contexts and the stability of adhering to the persona’s core traits can be difficult.
        - Zhao et al. (2021) suggested that calibrated prompting could improve task adherence by guiding the model to stay consistent with the persona’s predefined attributes, but achieving this balance requires careful prompt design and fine-tuning.

Vector Database

- Vector databases enable efficient, scalable, and real-time handling of high-dimensional embeddings in LLM applications. By leveraging these databases, LLM-powered systems can perform fast and accurate semantic searches, enhance user interactions, and provide context-aware responses

1. Efficient Similarity Search

LLMs often generate vector embeddings for text, and vector databases are optimized to store and retrieve these embeddings quickly based on similarity. The **goal** is to **retrieve vectors** that are "**close**" to a given query vector based on some **distance metric**, typically **cosine similarity** or **Euclidean distance**.

Example: Semantic Search (‘climate change effects’ => ‘global warming impacts’)

2. Handling High-Dimensional Data

Each embedding produced by an LLM represents rich contextual information and is typically a vector with thousands of dimensions.

Regular databases are not optimized for **storing** and **retrieving** such **high-dimensional data**. Vector databases, such as Pinecone, Milvus, and FAISS (Facebook AI Similarity Search), are designed specifically for handling these embeddings, supporting operations like nearest neighbor search (k-NN) in high-dimensional spaces.

Example: Document Retrieval (bookstore recommendation system)

RAG

1. **Analysis:** The input is analyzed and, if it is determined RAG should be used, transformed into a query.
2. **Retrieval:** The query is executed against a database, an API or other system.
3. **Augmentation:** The results of the query are used to create a new input, usually by combining the original input with the query results.
4. **Generation:** The enhanced input is provided to an AI to generate a response.

Most implementations of RAG today assume the analysis step encodes the user's input into an embedding that is used to query a vector database, whose results are snippets of knowledge injected into the user's input.

- more trustable and up-to-date data
- since the results are injected into the input, uses more space

Retrieval-Augmented Prompting

a type of RAG

1. **Analysis**: The input is analyzed to determine if the current prompt is still relevant to the chat with the user. If it is, the input is passed to the AI without any augmentation. If not, it is transformed into a query.
2. **Retrieval**: The query is used to return a single rule: a new prompt that provides instructions to the GPT on how to alter its interaction with the user.
3. **Augmentation**: The new prompt is injected into the original input.
4. **Generation**: The GPT switches into a new “mode” based on the instructions in the prompt and then replies to the user’s input.