Assignment 7

# Problem 1:
While they are great for text generation, reasoning is often challenging for LLMs. Why? What's being done to address that? Provide your answer in one or two paragraphs and provide appropriate citations. [2 points]
# Solution 1:
LLMs often struggle with reasoning because they primarily learn patterns from data rather than understanding underlying logic. They excel in generating coherent text based on context but may falter in tasks requiring complex, multi-step reasoning or understanding abstract concepts. This limitation arises because LLMs are trained on vast amounts of data to predict the next word, not necessarily to reason logically or deduce. To address these challenges, researchers are exploring techniques such as:

1) Prompt Engineering: Prompt engineering involves crafting specific prompts or input structures to guide the LLM in generating more accurate and relevant responses. Techniques like few-shot or zero-shot prompting help models better understand the context and nuances of the tasks. This approach is explored in "Language Models are Few-Shot Learners" by Brown et al. (2020), which highlights how carefully designed prompts can elicit improved performance in various tasks .

2) Chain of Thought (CoT) Prompting: This technique encourages the model to generate intermediate reasoning steps before arriving at the final answer. By breaking down complex reasoning tasks into smaller, manageable steps, CoT prompting helps the model produce more logical and accurate results. The paper "Chain of Thought Prompting Elicits Reasoning in Large Language Models" by Wei et al. (2022) provides insights into how this method enhances reasoning capabilities in LLMs .

3) Retrieval-Augmented Generation (RAG): RAG combines LLMs with information retrieval systems to fetch relevant documents or knowledge pieces during the generation process. This method allows the model to access external information dynamically, improving its ability to handle questions requiring specific knowledge. The paper "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" by Lewis et al. (2020) describes how integrating retrieval mechanisms can enhance the performance of LLMs on knowledge-intensive tasks .

# Problem 2:
What are emergent properties w.r.t. an LLM? Why/how does it happen or how do you show that with an LLM? [2 points]
# Solution 2:
Emergent properties refer to complex behaviors or capabilities that arise in LLMs as a result of training on large datasets, even though these capabilities were not explicitly programmed. These properties manifest as surprising abilities, such as language translation, summarization,

or understanding context beyond the training data. Emergent properties occur due to the scale and diversity of data, which enable LLMs to generalize patterns and infer relationships implicitly. Demonstrating these properties involves testing the LLM on tasks it wasn't specifically trained for, revealing capabilities that emerge from its extensive learning process.

# Problem 3:
We are all about different types of experimentations for this assignment question, but it's up to you what those experiments look like! Expand and modify the class code at least three different ways and report your results for loss (quantitative) and generation (qualitative). For example, you can train a model using a different (ideally, larger) dataset, change some of the hyper-parameters (block size, batch size, iterations, number of heads, etc.), extend bigram model to trigram, use a bigger dataset to train, etc. There are no right or wrong answers here and you shouldn't shy away from trying something interesting (and not just something easy). Present your experiments with description (e.g., what did you change or do differently), results (training/validation performance and generation quality), and your thoughts (1-2 sentences) about what happened here. [6 points]
# Solution 3:

- You can change *number of iterations*
- Change *Hyperparameters* like *Batch size* (32 to 8 for example) or *Block size* (8 to 16 eg) or make *embedding size* larger (eg to 512)
- *Change architecture*, eg from Bigram to Trigram
- *Modify the dataset* and make it larger / more suitable