Harrison Stein

Exam 2 Report

GitHub: https://github.com/Harrison-Flax/Exam2

Building Ollama Models: https://github.com/ollama/ollama/blob/main/docs/modelfile.md

Overview of Models and Explanation

I have chosen the models TinyLLama, TinyDolphin, and Reader-lm as the first two are less computationally expensive and smaller in size than their standard models, which is useful for running the script with quick execution. I also chose Reader-lm which is different from TinyLLama and TinyDolphin as it is much better with reading and looking up information from websites and generating its own content and displaying it to the user with clear formatting which the other two models don't do. I wanted to use models that had less space than the standard ones especially those that weren't more computationally expensive, so I could preserve space and ensure a quick output on my computer with Ollama and the local application.

I could have chosen the standard Ollama models or those with specializations like with coding or language capabilities that were trained on datasets, but I chose to find models that would be the most efficient, least computationally expensive, and providing the most fulfilling answers. Even though TinyLLama and TinyDolphin are similar in size, they are different in training and architecture as TinyDolphin is trained off of the Dolphin dataset which is based of Microsoft's Orca paper which TinyLLama does not entail in its trained data.

I have also created my own LLM, DropAI which is based off of Microsoft's Orca2 model which can process prompts with reasoning from Orca's dataset. This model is the most computationally expensive out of the three and takes the most time to display an output. I decided to create my own LLM to expand upon the three LLM's I have already tested and to see whether I could change any of the standard models online for my own spin.

TinyLLama:

- ~700 MB
- Lighter version of Ollama model

TinyDolphin:

- ~700 MB
- Lighter version of Ollama model and trained on Dolphin dataset
- Dolphin dataset is based off of data from Microsoft's AI development

Reader-lm:

- ~3 GB
- Made for processing text and fetching URL's

DropAI:

- ~3 GB
- My own custom LLM based off of Microsoft's Orca2 model that has custom inputs in the Modelfile

Basic Exploration and Focused Experimentation

Questions (handwritten by me):

1) What is the computational power needed to solve the hardest math problem in the world?
2) What is the most advanced math problem a calculator can solve?
3) Can computers read math symbols effectively and easily without assistance?

Text Input for Summary (handwritten by me):

"Jewish delis have long been an emblem of symbol of kosher cuisine, but also one of Ashkenazi roots in Eastern Europe and particularly in Europe. Delis like Katz's and Second Avenue have been around in New York City since Jewish immigrants started coming in droves to the United States in Ellis Island before and after WWII. These delis have served as a place for Jewish culture and cuisine to thrive while keeping the hardworking stories of Jewish immigrants alive in NYC. However, these eateries are now under more threat than ever as rents in Manhattan increase dramatically, food prices rise, and maintenance for these delis are exhausting and ever more expensive as the years go by. Some of the latest victims have been the Carnegie Deli and Lindy's which closed under financial and landlord pressure. It is rather unfortunate that these delis have had to undertake such a crisis, but their preservation is more important than ever to keep the culture and history alive of the Jewish people and their immigrant backgrounds in NYC."

Prompts (handwritten by me):

Text Summarization: "Summarize the following text about Jewish delis in NYC, focusing on their history, culture, and current challenges"

Simple Code Generation: “Generate code based off of the prompt. Generate a simple function that can pull the latest information about the Dollar General stock”

Creative Writing: “Generate a short story based off of a unique prompt. Generate a short story that follows the adventures of Sonic the Hedgehog as he falls down from grace and Shadow ends up replacing him. Make it funny too and don't be afraid to get wacky with it!”

Multilingual: “Generate paragraphs in 3 SPECIFIED languages and translate them all to English. Show both original language and translation to English. Generate a paragraph in Hebrew, Japanese, and French and then translate it to English. Show both the paragraph in the original language and translation in English.”

Evaluations:

f"As a grader, rate the following {task} response for its quality and accuracy of truthful information."

f"Prompt: {prompt}\n\nResponse: {response}\n\nProvide a letter grade from F to A and a explanation behind decision."

f"Only grade on the F to A scale and do not use numerical values. Please grade accordingly like a human."

For the basic exploration tasks and the focused experimentation with the multilingual capabilities of the LLM models, I found all three models to work and to be effective albeit, differences in response quality, resource management, and evaluation metrics. The general questions, defined by Q&A, were all answered in detail, but TinyDolphin has the worst quality answer out of the three models with Reader-lm having the best quality answer. TinyLLama is in the middle as it does provide a quality answer, but Reader-lm still includes more detail and pulls from actual internet sources, so the user can read about additional information.

For the text summarization, TinyDolphin surprisingly performs the best as it has the shortest summary and does not copy everything word for word which is sufficient for a realistic summary. TinyLLama performs well, but the summary is longer than TinyDolphin and reads more from the text I hard coded in than producing a realistic summary. Reader-lm performs the worst as it fails to summarize the text and only copies it down as a summary which is bad and doesn’t constitute a realistic summary.

For the simple code generation, TinyLLama produces the best output with the code it produces to track the stock price of Dollar General as it is functional and provides a URL to get the request from for the stock price itself. TinyDolphin and Reader-lm produce worse code as TinyDolphin

does not check if the response is successful from the URL and does not use a dictionary for efficient code. Reader-lm's output is incorrect and incomplete as there must be other steps before loading the .json including writing the output to it and does not have error checks like if the response is successful which can produce or fail to produce an output showing the Dollar General stock price.

For creative writing, TinyLLama produces the most complete and compelling story as it follows the prompt with being unique and wacky while longest in length. TinyDolphin produces a good story however, it is not as long or interesting as the TinyLLama version and feels even more generic. Reader-lm failed to produce a story and did not write anything informative or with substance which shows that the model can't generate a story or at least from a prompt.

In terms of the performance of the models, TinyLLama performed the best as it took the quickest to generate a response and used the least memory and CPU usage which makes it efficient and more workable for less powerful machines. Reader-lm performed the worst as it took the longest to generate a response and used the most amount of memory and CPU power which makes it inefficient and less workable for weaker machines, making it more usable on higher end machines.

In terms of the multilingual capabilities, I had a prompt where the models had to generate a paragraph in Hebrew, Japanese, and French and then had to translate it fully into English as the directions specified. TinyDolphin ended up accomplishing the task correctly while TinyLLama and Reader-lm couldn't complete it. TinyDolphin produced the best response but has errors and doesn't provide the correct translations which means that it struggles with language generation and detection. TinyLLama produced a full paragraph in Hebrew with an accurate English translation, but did not do Japanese and French as my prompt dictated. Reader-lm failed to recognize my prompt and generate an output which means that the model isn't configured to understand different languages and provide a translation for them.

DropAI developed creative and explanatory responses to the Q&A, creative writing story, text summarization, and multilingual capabilities. The only con with DropAI is that it cannot generate code for tracking the Dollar General stock which is simple to do. Everything but the code was informative, adhered to the prompts, and actively displayed three languages with their original text and English translation.

The evaluations provided were accurate and adhered to the format I requested which is a F to A scale although, they were lengthy and at times unclear which is inefficient and confusing for the user. A standard or more complex model might perform better at evaluations, but the response given is sufficient and clearly judges the models at their core capabilities.

Strengths and Weaknesses of Models

TinyLLama: Produces the quickest output, with the lowest memory and CPU usage, and the most substantive output in terms of information and adherence to prompt. Best overall as it is the most efficient in time and resources while giving informative and mostly complete responses.

TinyDolphin: Moderately timely in output, with mixed memory and CPU usage, and the most complete output in terms of adherence to prompt and information given. Good for quick answers and short descriptions but cannot handle more complex tasks and even gives inaccurate information more often than other models.

Reader-Lm: Produces the longest output, with the highest memory and CPU usage, and provides URL's along with information from both the Ollama model and web sources for more complete information to the prompts and provides examples as well to give the most quality output to the user. Fails to work with the creative story and the multilingual understanding and output. Gives a poor solution to the Dollar General stock price coding problem and is lacking in detail and clarity which makes it a bad model to use for understanding and using code. The best choice for Q&A as it gives detailed responses and references links that explain where users can go to learn more about the topic discussed or prompt.

DropAI: Most computationally expensive and takes the longest to produce an output. Best choice for Q&A but also has great multilingual abilities due to its deep reasoning and detailed output from the Microsoft Orca dataset.

TinyLLama:

- Time: ~1-3 seconds
- Avg. CPU Usage: ~2-5%
- Avg. Memory Usage: ~50%

TinyDolphin:

- Time: ~0.5-2 seconds
- Avg. CPU Usage: ~2-6%
- Avg. Memory Usage: ~50%

Reader-lm:

- Time: ~2-7 seconds
- Avg. CPU Usage: ~11-16%
- Avg. Memory Usage: ~50%

DropAI:

- Time: ~13-27 seconds
- Avg. CPU Usage: ~2-5%

- Avg. Memory Usage: ~50%

Insights About LLMs and Capabilities

From working with the three models and testing how they adhere to the prompts and their own information they generate, I have learned how LLMs work and how the lighter models are less advanced, less computationally expensive, and can't cover as many areas as the standard or complex models. TinyLLama does the best job for a light model aside from TinyDolphin as produces descriptive output to the questions, can summarize text effectively, produces detailed and correct code, writes a fun and longer story, and can write a paragraph in another language and translate it fully in English. Reader-lm is best for answering questions and providing answers especially with online sources as it is detailed and is made for scraping from the internet for tasks tuned to information and descriptive outputs. DropAI is best for Q&A and multilingual functions especially when needed in research as the Microsoft Orca2 model and dataset is specifically built for reasoning, research, and examining models and providing explanations.

Reflections

For real world applications, each of the models are unique in terms of their specializations and should be applied specifically to those instead of multiple areas where the models may not be sufficiently trained in which Reader-lm is an example of in terms of code generation, creative story writing, and multilingual abilities. The most effective example proven from the outputs of each of the models is question and answering as all three models gave substantive responses and provided guidance for the user. Question and answering prompts can be most useful in terms of support assistance and tutoring to users as the models provide clarity and examples with the information, they post which makes it useful in learning with a real-world application.

The other tasks, while useful, including code generation, are better suited for standard or more advanced models that know how to generate requirements directly from prompts and can be trained in a multitude of items including multilingual writing and translations. Reader-lm and TinyDolphin should not be used in these areas as they most often fail to read and produce responses to the prompts and generate either incomplete or inaccurate information. While they still can be used in real world situations, it is best to avoid these areas or just use them for basic questions if the user wants information on these more advanced tasks.

Creative writing and multilingual capabilities were the most severe instances of lacking quality across the three models as each of the outputs remained incomplete, filled with mistakes, or incorrect grammar in both the language and translation. The most effective way to test these tasks appropriately would be to use a model that is trained in languages and can provide accurate writing and translations per the prompt.

In terms of energy and timely outputs, I would pick the most effective model while having one that is informative and answering prompts correctly which would be TinyLLama. TinyLLama is the most accurate and moderate out of the three models and can be used in real world situations since its responses are informative and adhere to the prompts, ensuring that users can expect successful results.

It is important to note that these AI models should be used as a check or a partner in terms of answering prompts or seeking clarity as they lack accuracy in terms of their trained data and having human judgement can provide corrections and realistic solutions that are not robotic sounding.

Challenges

I faced a few challenges in figuring out how to organize the code, provide evaluations, log information to a .txt file, and measure resources for memory, CPU, and time taken for responses. I was able to overcome these challenges by figuring out the imports to use and then implementing the functions themselves by reading online documentation and then applying each of the functions to the task methods and then writing everything to the log .txt file at the end to capture the entire output sequence. In addition, I solved the incorrect CPU usage percent calculation by setting the interval to 0.1 from 0.5 and creating a for-loop with a try block to measure each Ollama process and track the CPU usage there. It takes the average by taking the CPU calculation outside of the for-loop and adding it to the Ollama CPU measurement and divides it by 2 and returns both the memory and CPU. I also added an except statement for an error check, so if the code cannot measure the CPU process, then it will print out that no such process exists, and access denied.

Each of the things like response quality, time taken, resource usage, and evaluations are made possible by calling these methods within task methods with the parameters model, question, response, task, result_time, avg_memory, avg_cpu, and evaluation. This relies on object-oriented programming and inheritance as the measurement methods are in a separate class called Benchmarks which Tasks inherits from, so the task functions can call on the methods to work in tandem.

I had to reorganize the main method a bit to import or inherit the Tasks and Benchmark classes with their respective methods while encapsulating the main initiative code in main() and calling main itself.

When developing the Modelfile for DropAI, I had to figure out the correct formatting by looking at the documentation online for setting up the temperature (scale of creativity in response) and what the prompt for the agent would look like. Once I finished the Modelfile, I built the LLM using the command “ollama create DropAI -f ./Modelfile” in terminal.