DiscordGPT: A GPT Language Model

Chris Bonner
Virginia Tech
Blacksburg, Virginia
chrisbonner@vt.edu

Molly Finley
Virginia Tech
Blacksburg, Virginia
cmfinley3@vt.edu

Daniel Palamarchuk
Virginia Tech
Blacksburg, Virginia
d4n1elp@vt.edu

ABSTRACT

Large Language Models (LLMs) have been highly researched and remarked over their ability to "understand" natural language and handle complex questions in varieties of fields. However, such models tend to generalize and produce a generic answer that minimizes bias. Here, we want to take advantage of a markov chain model and the open-source LLaMA and GPT-2 language models to mimic this capstone team's personal language patterns in a text autocompletion task. We will evaluate the models through seeing how often these models can correctly guess successive words of incomplete sentences.

1 INTRODUCTION

1.1 Initial Problem

Initially, we wanted to create a language model that is capable of chatting with a user while mimicking one of the project team members. Current state-of-the-art language models are trained on a broad data set of text, written by many different people, so it is impossible for them to mimic a specific person's language patterns unless that person is famous and has speech patterns somewhere on the internet.

1.2 Motivation

Our motivation for solving this problem is that the blandness of AI generated text makes it less useful. There are many instances in which it is important that a given piece of text sounds like it is coming from the the original author and so current language models can not be leveraged to make that task easier. If our project is a success, then language models become a more direct stand in for that person doing the writing themselves, allowing them to automate some of their work.

1.3 Initial Approach Limitations

Our goal was to work towards eliminating the stated problem was to create a language model that generates text that specifically sounds like its user. We ran into several limitations when working on this initial approach. First, we had issues with GPT-3 and being unable to train and use it except through an API provided by OpenAI which came with a higher cost than allotted for this project, so we tried switching to the open source LLaMA model. But that came with its own challenges. The biggest of which was that we did not have the computing resources necessary to train it on a large dataset for a large scope project, like generating whole pieces of text. To train the LLaMA model for the task we originally wanted would require in excess of 16GB of VRAM, which is all we had access to. We decided to pivot to a smaller problem, which would allow our training to fit within our resource limitations.

1.4 Modified Problem Description

We want to utilize open source large language transformers and Markov models in order to autocomplete words based on previous user messages. This way, we would not have to come up with a conversational dataset and can instead produce models that purely model our own speech patterns.

2 RELATED WORK

2.1 Literature Survey

The basic idea behind a Markov chain is that there exist a sequence of states, each of which is dependent solely on the state before it. Then, probabilities can be found for the likelihood that some specific state occurs directly after the current one. To use language as an example, we can break up a sentence into a sequence of words (states). The word "the," in the context of the sentences before this one in this paragraph, can be followed by "large," "model," "state," "basic," "likelihood," and "current" (The word "state" appears twice. Everything else appears once). As a result, the probability that the word "state" comes after "the" is $\frac{2}{7}$, while everything else appears $\frac{1}{7}$. We can then create an entire square matrix of such conditional probabilities "given word X what are the chances we get word Y." However, such a matrix would be computationally expensive to hold on to and to calculate the likeliest sequence of words (not to mention it would be very sparse), so we decided to use a hidden Markov chain.

Rather than holding the relationships between specific words, it would only hold on to the information about the types of words. For example, rather than calculating which specific word comes after "the," we can instead calculate that the probability of a noun occurring after "the" is $\frac{4}{7}$, while the probability of an adjective occurring is $\frac{3}{7}$. Then from there we can predict the actual word that is said given the type of word, and that prediction would constitute the "hidden" part of the chain. We do not know the relationship between the actual words, since that is hidden, but we can infer that relationship based on word types.

With regards to a more complicated language model, those require a more intense training process in terms of volume and density of data. Our group cannot gather the data required to train one from scratch in a reasonable time frame, so we decided to take advantage of existing language models. We initially looked at the GPT-3 language model and tried to adapt it to our own language [6] using the Process for Adapting Language Models to Society. It takes a more general language model and adapts it to specific situation using data specifically crafted to the situation. This general process is called [1] few-shot learning and boils down to the GPT-3 model being able to write the answer to a prompt in the context of prompt-answer input data that creates a bias towards the given topic. In this context, we can add on top of the GPT-3 model 100-200 of our own handcrafted questions/answers (which can be completed using the Markov model) and as a result create a language model that sounds like us.

An alternative state of the art to using OpenAI's LLM is Facebook's open source LLAMA language model [8], which has the same fundamental principal of being a massive model that is capable of recognizing language patterns. A model that is able to take advantage of this LLM is the ALPACA [7] instruction-following model that is capable of fulfilling a handful select general tasks such as writing a list for a grocery trip or New Year's resolutions.

2.2 Limitations of Existing Approaches

Some of the limitations of using Markov Chains comes with its unsupervised techniques. Since the model relies on the last generated word to decide its next step, it is limited in its ability to "think ahead" and heavily limits its response ability to answer more advanced or complex prompts. A limitation of the GPT-3 model is the inability to manually tune the hyperparameters. This makes training the model much less sensitive to changes and will require more data tuning in order to tune certain features of the model. In addition, it is a general-use LLM, so it is predisposed to give a general answer. A more recent limitation of GPT-3 was the fact that its API was prohibitively expensive, with no alternative way to run the model. This prevents us from being able to freely tune a model as needed. This was a major issue with our initial code session, which we'll elaborate on in the result analysis section.

Some limitations of the LLAMA model are similar to GPT-3. It is a general-purpose LLM that is predisposed to giving a general answer. The limitation of ALPACA is that it is fine tuned to fulfill instructions, so it is slightly unsuitable to being trained to mimic a person. It is more appropriate to use a model such as GPT-3 or LLAMA that has general pattern recognition and that has not been tuned to a specific task.

3 PROPOSED APPROACH

We propose a method of adapting a markov chain/large language transformer model to a user's personal data using little space. In our implementation, we train these large models on personal discord data to demonstrate a proof of concept.

3.1 Problem Definition

There is currently no sophisticated open source autocompletion option that can compare to phone keyboard autocomplete or Google's autocomplete in gmail/docs/other Google products. We plan on utilizing three different models, markov chains, GPT-2, and LLaMA, trained on personal discord data in order to create machine learning-assisted auto-completion akin to Figure 1.

3.2 Data Pre-Processing

The raw data provided by Discord on request comes in a complex library of files per interaction. In order to condense this, we created a bash script to extract the JSON message data from each interaction file and import it into a single csv file. Our next step involved cleaning the data from noise and spam. The process involved removing hyperlinks, tags, replacing Discord/unicode emotes with text representations, replacing special characters with ASCII, and

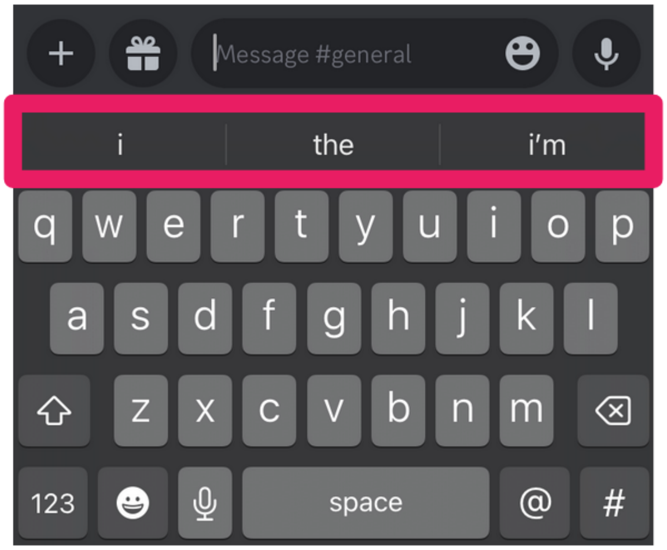

Figure 1: Autocompletion Example

separating punctuation. From there we split the processed data into a 80/20 training/test set for the Markov and LLaMA models, and a 60/20/20 training/validation/test set for the GPT-2 model.

3.3 Model Architecture Description

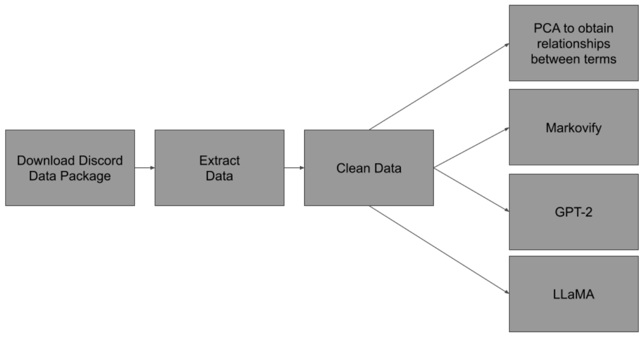
Download Discord Data Package
Extract Data
Clean Data
PCA to obtain relationships between terms
Markovify
GPT-2
LLaMA

Figure 2: Project Pipeline

4 EXPERIMENTAL EVALUATION

4.1 Exploration & Observation

Our exploratory data analysis was primarily composed of looking at the discord data that we pulled and coming up with ways to illustrate the speech that we have said. One of the simplest and most effective strategies to grasp at a corpus of data is to create a word cloud. For the sake of space, we only chose to include Daniel's results in this section, but they can be replicated for any one of us (see Figure 3).

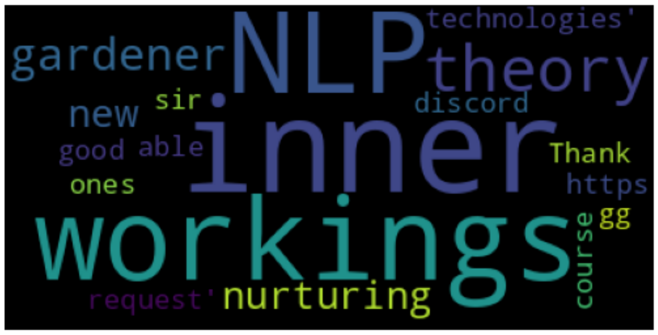

Figure 3: Word Cloud on Daniel's Discord Messages

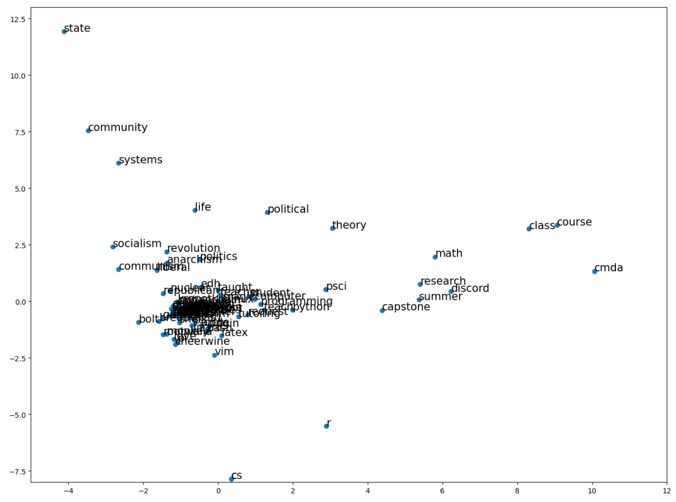

Figure 4: Principal Component Analysis of Terms in Daniel's Discord Messages

Wordclouds mostly look at frequency (and judging by the topic, frequency of more recent conversations), so we can easily see that Daniel's wordcloud that there is a discussion of gardening, NLP, theory, and the inner workings of it.

We next wanted to look at the relationships between words. To accomplish this, we performed a Principle Component Analysis (PCA) on the word embeddings of individual corpuses of messages. The embeddings were obtained using Gensim's Word2Vec, which [4] captures whether individual pairs of words appear within a specified word window and puts the results in a relatively low dimensional space. This allows us to move beyond simple word frequencies and observe contextual word similarity within our discord messages to see which words are "similar" to each other. These patterns will be what we will look for when we build language models that try to replicate our own speech. Daniel's results for his personal selection of words is illustrated in Figure 4.

Most terms are bunched up around $(0, 0)$, which demonstrates that there is no strong connection between most words with everything else. At around $(-2, 2)$, there lies a corpus of political terminology, which conceptually makes sense since they are all related to politics. On the other hand, at around $(5, 1)$, there lies a cluster of academic terms. Interestingly, the term "theory" lies somewhere in between those two, which goes in line with Daniel's reading of both mathematical machine learning theory in addition to political theory. "CS" being so far down was unexpected, we had hypothesized that it would be near "CMDA," although it might just be because the two terms are rarely used together and are instead used across different Discord Servers to speak with different audiences. A "mimic" model that we will later develop should be able to capture these clusters and interactions in forming its own speech. This PCA is insightful in showing the types of biases we should be including within our handcrafted datasets. Some words are more related to each other, so our new datasets should be reflective of that and not spew random garbage.

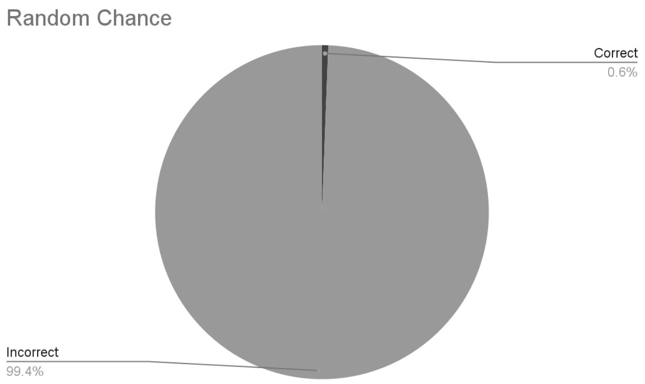

Figure 5: Random Weighted Word Selection

4.2 Result Analysis

As mentioned, we have developed three models with which to attempt autocompletion. As a simple exercise, we created a control model that randomly picks a word out of all possible test words in order to provide a baseline. Figure 5 demonstrates, random word selection predicts the correct word 0.6% of the time.

4.2.1 Chris-Markov Model. Our first model took advantage of Markov probabilities in order to find the most probable predictions. It must be noted [5] that hidden Markov models look at only the current word in determining the next word, while the present model used three words into account. According to the package we used [3], these three words compose a single state, so if a combination of three words is not observed by this model, it will turn up as invalid. As a result, 80% of our runs ended up being invalid because the specific test starting sentence could not be found in the initial states. Those that were found ended up being correct around 4% of the time (Per Figure 6), making it slightly better than random.

4.2.2 Molly-GPT-2 Model. Our second model was able to fine tune a simplified open-source GPT-2 model with 117 million parameters (the regular version has 1.5 billion parameters) in order to get an accuracy of 13.5% (or 17.1% if looking at the first 3 predictions), as

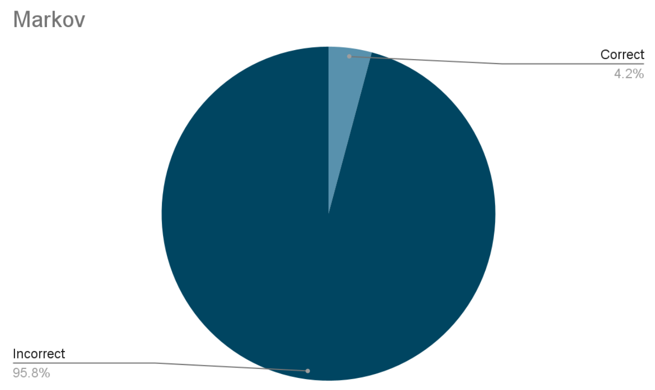

Figure 6: Markov-based Word Selection

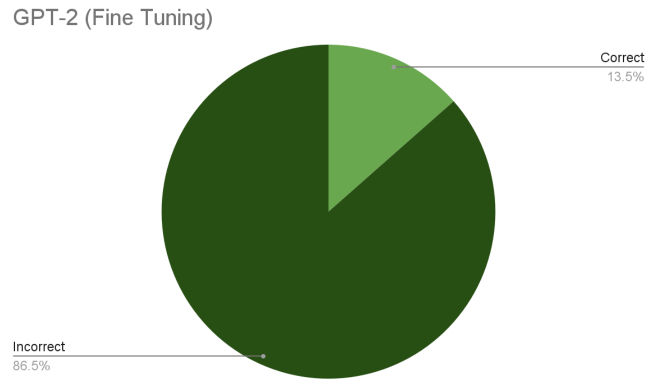

Figure 7: GPT-2 Word Selection

seen in Figure 7. This is a massive improvement over the Markov model in two ways:

(1) There are no invalid results; the model was able to piece together all the new data by itself.
(2) The model was able to capture the context of the given input text in order to figure out what word was wanted.

We did get an opportunity to extend this approach and get a glimpse of what one of our models might have looked like if we had the resources to do actual text generation. Below is a small excerpt from this experiment, censored to remove swear words:

> how are you doing? Im not sure if you're doing it for the money or not but i'm not really sure what you mean by that. i'm just trying to figure out what to do with my time im not a big fan of the whole "you're not allowed to be a part of this *****" thing you know what i mean

While the actual contents of the text were gibberish, the actual speech patterns did reflect Molly's typing style on discord.

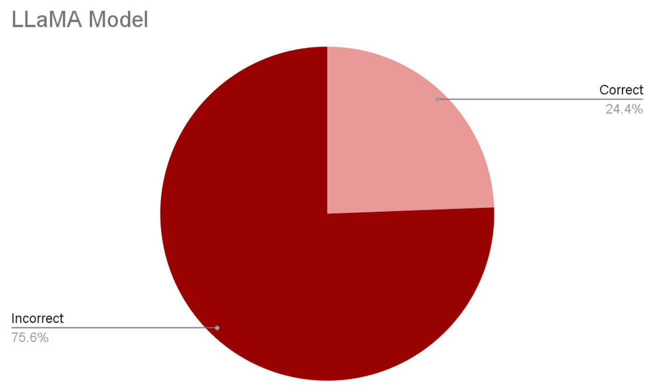

Figure 8: LLaMA word selection

4.2.3 Daniel-LoRA Model. This model was able to take the 7 billion parameter LLaMA model [8] and train on top of it LoRA [2] matrices that can be added to the original model in order to modify it. This model was several times bigger than the GPT-2 model, which, in addition to the LoRA matrices, allowed the model to achieve 24.4% accuracy (see: Figure 8). There were issues with loading up and training the model on our local machines, so this model was trained on Google Colab. This says something for the practicality of having a model that takes up so much space when all it's doing is autocompleting sentences, but that aside this has demonstrated to have the best results.

4.3 Ablation Study

We tested out the GPT-2 and LoRA LLaMA models without any parameter tuning. Without fine tuning, GPT-2 got the correct word 8.2% of the time (as opposed to the 13.5% with fine tuning), demonstrating the effectiveness of the fine tuning process very well. When given 3 chances to output a word, the untuned GPT-2 model got the correct answer 14.1% of the time, while the tuned version got it right 17.1% of the time. This is less of an accuracy boost, but still a significant one.

The untuned LLaMA model was able to guess the next word 21.5% of the time, giving praise to the base model's ability to predict a word given just its raw computing and inferential power. This is in comparison to when the LoRA-Daniel weights were added on top of the model, which was able to output the correct answer 24.4% of the time. This is not as impressive as the GPT-2 accuracy boost, but significant nevertheless, especially given our evaluation methodology. Both LoRA and regular fine tuning can thus be shown to improve a large language model's capacity to predict future words.

5 FUTURE WORK

5.1 Evaluation Methodology

We could have better evaluated our models. We would only count a test completion as correct if the model was able to exactly predict what the word was. This poses a problem, because we observed that the model would produce sensible results a large portion of

```
Prompt: There's always a ____
Expected: winner     Predicted: way
Prompt: With that _____
Expected: being      Predicted: many
Prompt: This high  _____
Expected: horse   Predicted: school
Prompt: but started imagining a scenario where _____
Expected: a       Predicted: you're
Prompt: would probably have to talk it through with _____
Expected: molly     Predicted: the
```

Figure 9: Predictions for the Daniel-LLaMA Model

the time (as shown by figure 9), but these sensible results had to be marked as incorrect on account of our evaluation methodology. Future work would test our models using methods described in the literature review [1] in a zero-shot context, where the model is allowed to pick between a handful of different answer choices. This way, the model does not get penalized for coming up with an answer that is very realistic but not the exact word that is expected.

5.2 AdaLoRA

Our current final model takes advantage of LoRA [2] in order to produce low rank weights to be added on to the LLaMA model in order to obtain a more personalized model. In March, 2023, there was released [9] an improvement on the LoRA model with AdaLoRA, which is an "adaptive budget allocation" LoRA capable of determining the most important weight matrices of a LLM rather than treating each matrix as equal. This allows more budget to be allocated to matrices deemed more important, thus making the low-rank adaptations have more impact. Future work would upgrade from the standard LoRA to this new state-of-the-art in low-rank fine tuning.

5.3 LoRA for more members

The LoRA fine-tuning process allows a person to obtain 8 MB worth of rank-reduced weights worth of personalized weights and add it onto the main model. As a result, it is very easy to story personalized weights for individual members and just add them on top of the LLaMA model. We would just need to store the base model in GPU RAM and we will then be able to add the weights of whichever person we want to autocomplete for. This LoRA process can be done for smaller LLM's, such as GPT-2, and it would produce similar results [2] to as if the weights were directly edited via fine tuning. So, we can store multiple LoRA's and take up much less storage in the end than if we had to store multiple fine-tuned LLM's for each person.

6 CONCLUSION

As a warning to future capstone teams that look to do something related to NLP: Do not try and adapt a raw large language model such as LLaMA unless you have access to powerful GPUs capable of holding and fine tuning the model in RAM. While we were able to produce results that shows promising possibilities for future work, working with these models are very computationally intensive and require a deep understanding of how these models were trained before exploring further. Until a sufficient model is developed that can be trained and explored using personal hardware, many parts of lower level research into the intricacies and capabilities of LLMs will be limited.

REFERENCES

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *CoRR* abs/2005.14165 (2020). arXiv:2005.14165 https://arxiv.org/abs/2005.14165.

[2] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685 [cs.CL]

[3] jsvine. 2022. Markovify. https://github.com/jsvine/markovify.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781 [cs.CL]

[5] Lawrence R. Rabiner and B. H. Juang. 1986. An introduction to hidden Markov models. *IEEE ASSP Magazine* 3 (1986), 4–16.

[6] Irene Solaiman and Christy Dennison. 2021. Process for Adapting Language Models to Society (PALMS) with Values-Targeted Datasets. *CoRR* abs/2106.10328 (2021). arXiv:2106.10328 https://arxiv.org/abs/2106.10328.

[7] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

[8] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA. (2023). arXiv:2302.13971 [cs.CL] https://arxiv.org/abs/2302.13971v1

[9] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning. (2023). arXiv:2303.10512 [cs.CL] https://arxiv.org/abs/2303.10512