GaS – Graph and Sequence Modeling for Web Agent Pathing

Stanford CS224N Custom Project

Marcel Rød
Department of Computer Science
Stanford University
roed@stanford.edu

Arantzazu Ramos
Department of Aeronautics and Astronautics
Stanford University
aramosv@stanford.edu

Matthew Noto
Department of Computer Science
Stanford University
mattnoto@stanford.edu

Abstract

Modern retrieval mechanisms rely on ordered, regular datastructures to find information relevant to a given query. The web is a massive repository of information, yet it is organized with few indices and overviews to serve as handles for a retrieval model. The growth in NLP capabilities demonstrated by modern LLMs have shown that for short to medium-length snippets of text, transformer models can match or even exceed human capabilities. We show that one can use the content and structure of the web to train few-shot models for navigating to the information in a given prompt. Our method finetunes a Mistral-7B generative decoder causal language model to follow links on the web, with Wikipedia as our primary domain, and shows that small amounts of finetuning can make a difference in performance for this task. We propose metrics for performance in the task of browsing using web links, and show how our model performs on them before and after training. We also contribute a novel data processing pipeline for Wikipedia dumps, with flexible, parallel text processing, as well as a method to simply define traversal of the underlying graph in C++.

1 Key Information

- Mentor: Tathagat Verma
- External Collaborators (if you have any): N/A
- Sharing project: N/A
- Team Contributions:
  - **Marcel Rød**: Implementations, data preprocessing pipeline, graph algorithms, model training, evaluation, reports, poster
  - **Arantzazu Ramos**: Literature review, loss function design, prompting, reports, poster, evaluations
  - **Matthew Noto**: Implementation, model studies, evaluations, preparing datasets, graph algorithms, reports

2 Introduction

Large Language Models (LLMs) have shown much promise as general problem solvers in the past few years. GPT-style language models (Radford and Narasimhan, 2018) are trained to perform next-token prediction. In order to exceed at this objective, they have to learn to understand the structure and semantics of language. With further training, it has been observed that some level of reasoning ability and planning (Valmeekam et al., 2023) also can be learned through the simple yet rich task of next-token prediction. Furthermore, LLMs can answer factual questions and inform their writing using knowledge about the world. This capability requires that either knowledge be latent in the model weights (Burns et al., 2024), or for the user or framework surrounding the model to present relevant information within the model's context window (Dong et al., 2023). State of the art models show remarkable ability for lossy compression of knowledge into a relatively small set of weights, but don't have an explicit way to verify or cite the source of the information they provide (Delétang et al., 2023) without "hallucinating" them. There are many approaches to augment LLMs with retrieval capabilities, yet most rely on a method to compress each article into an embedding vector database that supports lookup during retrieval (Gao et al., 2024), either for looking up a textual unit, or looking up a subgraph of connected units of knowledge, as in Larson and Truitt (2024).

The breath of knowledge gathered by humanity is in large part readily available across the web in human-readable format, a format that is quite familiar to language models. When looking for information on the web, humans have two key methods of navigation: keyword search and hyperlinks. Navigation by hyperlinks is a form of graph traversal of the web where pages are nodes, and each link is a labeled edge. By the nature of graph traversal, in a well-behaved graph with a constant number of edges per node, the expected distance between two nodes is logarithmic in the total number of nodes (Newman et al., 2006). For Wikipedia, the number of connections one must traverse to get from any connected article to any other article is at most 8, and even for this seemingly short distance very few pairs of pages have been found with this graph distance between them (Wikipedia,

2024c). Similar results are true for the truly massive graph of the entire internet, where the average degree of separation between any two web pages was found to be 19 as of 2013 (Barabási, 2013).

Inspired by "The Wikipedia Game" (Unknown, 2022), we aim to study the ability of modern open-source large language models to traverse the web, specifically in order to find the shortest hyperlink path between two articles. Using our zero-shot performance as a starting point, we fine-tune our language model to become better at the skill of predicting the next token.

3 Related Work

"The Wikipedia Game" (Unknown, 2022; Wikipedia, 2024d) is a popular online game where the player is asked to find a path from a given starting page to a target page. The goal is to find such a path within a given time frame and in as few clicks as possible. Previous research has studied the Wikipedia Game from a different perspective, documenting how humans act when trying to solve this problem (see West et al. (2009) and West (2010)). Interestingly, the end goal of this line of research was to better understand the semantic relationships between pages given the paths measured from online players, ultimately to advance language models and knowledge retrieval. Our approach is the inverse of this – we aim to use the understanding of an existing pretrained language model to solve the problem of navigating between pages on Wikipedia the way that a human would, or perhaps even better.

Significant research has been dedicated to understanding and predicting the hyperlink structure of Wikipedia. Mihalcea and Csomai (2007) leverage classic NLP techniques to predict which words in an article are missing hyperlinks. Similarly, Milne and Witten (2008) use machine learning to detect links in an article and classify the type of relation they represent between the articles. These projects aim to take the unstructured information found in the text of Wikipedia pages and order them into a more complete knowledge graph, which can be more readily used by methods that are designed for knowledge graphs.

In Scaria et al. (2014), the authors investigate what leads to users backing out of a graph search, building graph structures that explain the different strategies employed and patterns that show up in people's browsing histories. The study shows that humans are very inefficient at browsing when looking for something specific, but are good at finding local distance minima according to heuristics that the authors approximate using tf-idf distances.

The Mistral-7B language model (Jiang et al., 2023) is a new, highly capable decoder-only GPT-style causal language model with a simple architecture that is simple to replicate. Modern finetuning techniques can be employed to train it, including Low-Rank Adaptation (Hu et al., 2021), the AdamW optimizer (Loshchilov and Hutter, 2019) and LLM.int8() (Dettmers et al., 2022) quantization. Dettmers et al. (2022) also introduces a quantized version of the AdamW optimizer, suitable for training models at low precision. These methods quantize weights to 8-bits and sometimes as low as 4-bits of precision in order to keep memory usage low, but still perform computations and accumulations in `bfloat16` (Kalamkar et al., 2019).

4 Approach

Our method can be broken down into a few parts:

1. Collecting a dataset of Wikipedia articles and metadata
2. Cleaning the dataset and extracting links
3. Assembling the dataset with links into a graph and backward graph for traversal
4. Sampling and traversal methods for the assembled graph
5. Building an NLP model for next-token prediction
6. Quantization of the model to get it ready for training
7. Metrics and a loss function for evaluating and training the model

4.1 Collecting our Dataset

We quickly found that there was no way to source a dataset of Wikipedia in the format and with the metadata our model needs. Additionally, we want to be able to use up-to-date datasets so that we can investigate our model's performance by browsing current Wikipedia. Thus we elected to build our own dataset from raw data sources, which turned out to be a time-consuming and difficult operation. The raw data inputted into our processing pipeline was downloaded directly from Wikipedia's official bi-monthly data dumps.

4.2 Cleaning the Dataset

We attempt to extract the parts of each Wikipedia page that correspond to raw text, identifying and erasing nested custom structures contained in the special Wikitext template tags we find on each page. Figure 1 shows a simple example of this process for the Wikipedia page about Anarchism. See 5.1 for details on this process works. We evaluate our cleaned dataset by manual inspection of hundreds of pages, finding elements that ended up as false positives or false negatives in the cleaning process. Further, we employ the Mistral tokenizer to small sections of the text to determine which parts are problematic for the total sequence length and should be filtered out. The original plan for cleaning was to build our own custom parser, as we greatly

underestimated the complexity of the worst-case Wikipedia page. This approach failed on many edge cases, and we found that it was an untenable approach to the problem, so we eventually decided on parsing using the nearly complete C-library in `mwparserfromhell` (Earwig, 2013). Even with this relatively robust parser, we have continued to find bugs in the parsed documents. Since this took us a long time to do, and we couldn't find a similarly processed dataset on the internet, we intend to publish our methods, and perhaps maintain an up-to-date (updated for every new Wikipedia dump) dataset on Hugging Face (HuggingFace, 2024).

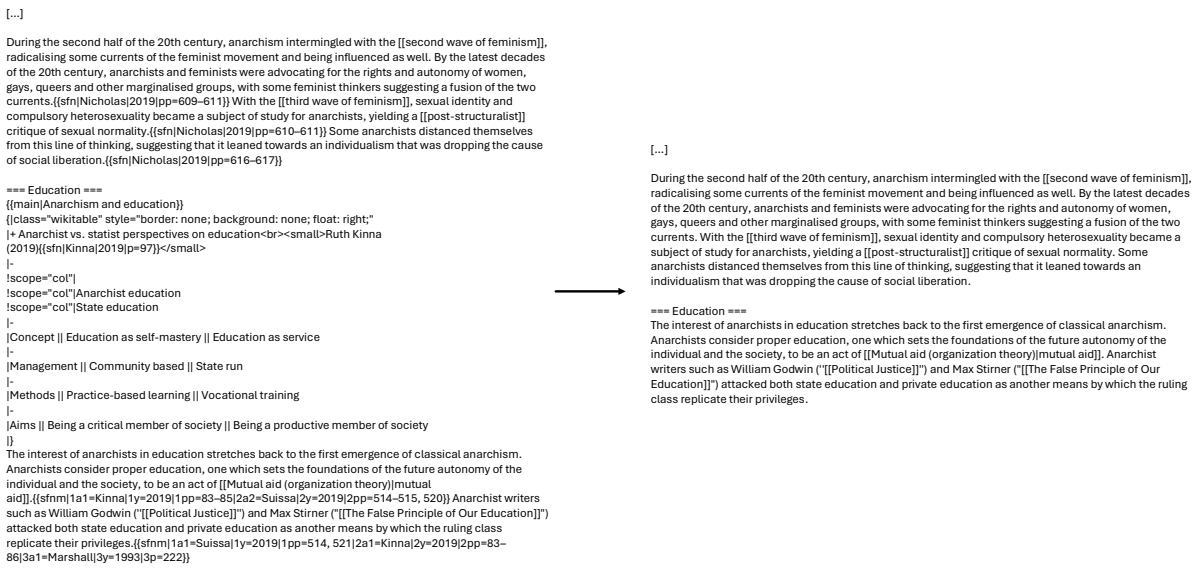


Figure 1: A small section of the Wikipedia page for "Anarchism," before and after cleaning. The cleaned up version only includes plaintext and links along with their displayed text in a way that is simple to parse later in the pipeline. The cleaned up text tokenizes to 274 tokens, down from 592 tokens for the source text. Our tests showed that Mistral-7B was able to understand the meaning of the formatted links in the text, so we opt to retain them.

4.3 Assembling the Graph

Once we have cleaned and parsed our dataset, we need to organize it for traversal. Since we are dealing with a massive dataset that cannot fit into memory, we assemble the operations we need to run using `LazyDataFrames` from the `polars` library (Polars, 2023). Due to this setup, all the operations we perform to construct the graph must be operations on such a `LazyDataFrame`. We take the link texts, canonicalize the titles that they link to, and identify which page IDs those links correspond to. We also reverse these links to figure out which pages are linked to by which other pages. This will be important for the graph traversal algorithms.

4.4 Sampling and Traversal Methods

Our training and evaluation flow requires pages to be sampled randomly, both for the starting page and the target page. The candidates for sampling the target page are chosen based on the page views. We uniformly sample the top 3 million pages ordered by page views to find the target page. To find the starting page, we start at the target page and traverse the back edges to find candidates for sampling. A starting page candidate is any traversed page which has at least 5 links on it, and is at least 2 links away from the target page. We sample these uniformly to get the starting page.

After we have determined a good starting and target page, we need to figure out which links from the starting page are part of a shortest path to the target. We again do breadth-first traversal over the back-edges starting from the target page, keeping a mapping of how far away each traversed node is from the target page. Once we discover and traverse the starting node, we know that each of the linked-to pages must have been already traversed (since this is an invariant of the BFS algorithm). We then iterate through all the neighbors of the starting node and check their recorded distances. If a neighboring node's distance to the target node is saved as being equal to the minimal distance to the target node, we record the corresponding link as an optimal link, otherwise it's sub-optimal. Figure 2 shows an example of which links would be considered optimal. The green colored nodes are the neighbors of the start page that are a part of a shortest path to the target page.

4.5 NLP Model for Next-Token Prediction

We employ an open source large language model trained on the pretraining task of next-token prediction. The model is prompted using a preamble describing the task along with the title of the current article and the title of the target article. Then, the article text follows, with links included and highlighted. Finally, the model is prompted to generate the text corresponding to a link.

Our language model architecture is that of Mistral-7B (Jiang et al., 2023), which is a pretrained decoder-based GPT-style transformer model. Mistral-7B uses several techniques from newer open source language models, and innovates with the use of sliding-window attention with the FlashAttention v2 library (Dao, 2023) with rotary positional encodings (RoPE) (Su et al., 2023). It also uses the RMSNorm (Zhang and Sennrich, 2019) (also known as the T5Norm) for layer normalization, along with a gated MLP feed-forward layer with the SiLU activation function (Elfwing et al., 2017). The 7B parameter pretrained model has a vocabulary size of 32000, a hidden size of 4096, an up-projected MLP size of 14336, 32 layers, 32 attention heads and a sliding window attention size of 4096 tokens. The model is pretrained to do next-token prediction on a large set of textual data, using a tokenization scheme similar to that of Llama 2 (Touvron et al., 2023).

For finetuning, we use Low-Rank Adaptation (LoRA) (Hu et al., 2021), along with quantization techniques from the `bitsandbytes` library, using LLM.int8() (Dettmers et al., 2022) and 4/8-bit quantization. Additionally, we use the `transformers` library to enable gradient checkpointing and further decrease memory usage. This allows us to run the forward and backward passes significantly faster, and to not run out of GPU memory while training.

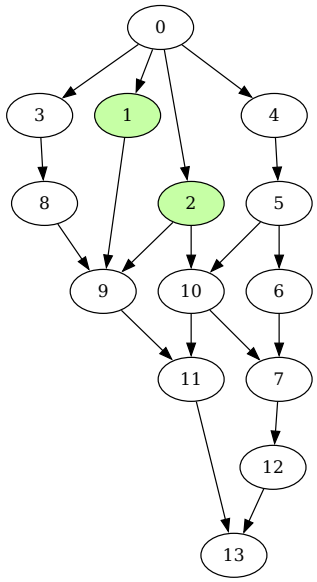


Figure 2: Given a start and target node in the graph, we mark the neighbors of the start node that are part of a shortest path from the start to the target. In this example, node 0 is the start node, 13 is the target node and [1, 3, 4] are all the neighbors of the start node. 1 and 2 are both part of shortest paths, since they both have distance 3 from the target node, while node 3 has distance 4 and node 4 has distance 5.

4.6 Training Loss Function and Evaluation Metrics

In order to unify the task of predicting which link to follow with the language model pretraining task, we use the prompts we generated, following with a single quotation mark '"'. When evaluating the model, we attempt to figure out

A problem that arises in this situation is for links with common token prefixes, for instance the links "Cornell University" and "Corn Tortilla" both start with the common token for "Corn," so if we attempt to make a choice based on the first token, there are two possibilities. In order to properly deal with this issue, one would have to follow multiple paths to find conditional probabilities and form them into marginal prediction probabilities for the specific groups of tokens. This requires a lot more computational cost than what we have available, so we approximate the result by considering any valid first token to be a valid link, but scaling the loss value for an incorrect prediction by the proportion of links with that prefix that are optimal.

The loss function for finetuning computes the cross-entropy loss for the predictions, where the probabilities computed for each valid link prefix token are combined into one element of the output distribution. Thus we get a mostly familiar crossentropy expression

$$\mathcal{L}(p, l) = -l_{\text{valid proportion}} \log \left(\sum_{i \in \text{optimal tokens according to } l} p_i \right), \tag{1}$$

where p is the distribution of tokens after applying the softmax to the output logits from our LLM, and $l_{\text{valid proportion}}$ is the proportion of the tokens with a common prefix to l that are optimal links.

We evaluate our model by a few metrics which we have introduced ourselves. The first is rather simple, and represents the total probability the language model assigns to valid link tokens, and we call it the ValidLink metric. We add up the probability assigned to all the tokens found in all the links, regardless of whether or not they are optimal. This metric shows us if the model has understood the basic task at hand, but not how well it's traversing the graph. The second is the TopK metric, a measurement of how many of the top k probabilities in the output distribution are the first token in an optimal link. k is determined by counting the number of unique first tokens for all the optimal links. This measures our ability to predict the correct path in the graph. Finally, we have a loss function used for training, computed based on the crossentropy of predicting one of the correct link tokens.

We had also planned to use a metric proposed in the project milestone – we called it the LinkMetric, and it would represent the total proportion of attention the model places on the tokens in each link in the text when predicting the first token of the outputted link. We expected this metric to show that the model learns to focus on the tokens that represent the links at first, but then over time learns to attend more to the rest of the sequence also. However, due to the specifics of FlashAttention v2.5.6 (Dao, 2023), getting the full attention map results in a buggy mess of outputs, and the fact that Mistral-7B uses sliding window attention makes it impractical to run this in a pure PyTorch implementation.

5 Experiments

5.1 Data

The complete set of Wikipedia articles is a massive dataset, and contains a wealth of pages and text data related to organization and formatting. For this to be usable, we need to prune entire pages, as well as irrelevant and noisy information stored in the formatting text of each page.

Our datapreprocessing pipeline is long and complex, and took a significant proportion of the time devoted to the project. This section will detail the process we undertook.

5.1.1 Filtering and Extracting Structured Data

The starting point for our data processing is a single 94GB XML file downloaded from the bi-monthly Wikipedia database dumps (Wikipedia, 2023). We parse the XML file using LXML (Behnel et al., 2005), which lets us stream and traverse the Wikipedia datastructure. Each entry in the document contains information about a single page, specifically tags containing the namespace, page id, redirect information, title and text of the article. Our processing eliminates pages not in the default namespace (namespace 0), as well as pages that have "disambiguation" in their titles. Otherwise, we keep every article, including articles with the sole purpose of redirecting to others. We store our data in a large `polars.DataFrame` (Polars, 2023) which we write as a Parquet dataset with 17.6M rows, totalling 21GB with columnwise compression.

5.1.2 Wikitext Parsing and Text Filtering

Wikipedia articles are written in the Wikitext format (Wikipedia, 2024b) (also known as MediaWiki), which is notoriously difficult to parse. Additionally, some pages have incorrect formatting, and will cause errors depending on the parser in use. The only remotely functional parser for the scale of data and specific information we need for our parsing and filtering is the aptly named `mwparserfromhell` (Earwig, 2013). We use `mwparserfromhell` to construct a recursive datastructure from the Wikipedia article, and traverse it to find elements that we need to track closely (e.g. links) as well as the ones we want to eliminate like tables, lists and other formatting structures. The output of our article processing pipeline is a set of new articles that have links that are in an easy to access format, and with most ancillary structures removed. This process is crucial for reducing the total number of tokens after tokenization for each article. Parsing and processing the entire dataset of 7 million articles with this parser would take more than two days without parallelization, and so we implemented a parallel approach using Python's multiprocessing library. Our method splits the dataset into chunks that are distributed to individual processes, one for each core on the machine. We process each row of the dataset using our text parsing pipeline, and return the processed, merged dataframe.

5.1.3 Additional Metadata

Wikipedia also hosts metadata that cannot be found in the main `pages` XML, which can be found in several individual raw data dumps which are organized by page ID. Of particular interest to us is the page view metadata, which signifies how many views a page got over the course of a month. This information is joined into the table of pages by Page ID.

5.1.4 Building a Graph

Once the dataset is gathered and parsed, we process each link on each page. Links in the Wikitext format are not by page id, but rather by the title of the page. However, there are some strange rules for matching links to pages, which we implement directly given their specifications (Wikipedia, 2024a) through a normalization scheme. We do another step of processing by shorting redirect page links. If a page link leads to a redirect, we instead record it as going to that redirect's target page.

We are left with the large dataframe consisting of all the pages on Wikipedia along with some metadata, as seen in Table 1. Using the links found in each article, we construct a second dataframe containing all the edges of the graph. This dataframe maps the page ID of the current page to a list of the page ids it links to in order, as seen in Table 2. We also store the list of link names for future use. Finally, we also need to create a table containing the reversed link edges (back edges), which we will use in 5.1.5.

Page ID	Canonical Title	Content	Formatted Title	Page Views (Monthly)	Link Count
12	"anarchism"	"Anarchism is a…"	"Anarchism"	25165	288
39	"albedo"	"Albedo is the fraction of…"	"Albedo"	12220	138
290	"a"	"A, or a is th…"	"A"	59271	137
303	"alabama"	"Alabama is a state in…"	"Alabama"	43548	617
⋮	⋮	⋮	⋮	⋮	⋮

Table 1: Head of the final Polars DataFrame used to store information about each of the pages. 6.6M rows.

Page ID	Links Text	Links Page ID
12	[“1872 Hague Congress”, “1999 Seattle WTO conference”, . . . , “workers’ self-management”]	[1915832, 312622, . . . , 40949353]
39	[“Aerosols”, “Almería, Spain”, . . . , “weather”]	[57763, 385256, . . . , 33978]
290	[“A (Cyrillic)”, “A-list”, . . . , “Ă”]	[748872, 301257, . . . , 2247682]
303	[“1974 Super Outbreak”, “1993 Storm of the Century”, . . . , “voter registration”]	[402969, 1078558, . . . , 994892]
⋮	⋮	⋮

Table 2: Links with corresponding link text for each article on each Wikipedia page. 6.6M rows, with 142M total entries.

5.1.5 Graph Traversal

Our model's objective will be to select the best link to get from a starting Wikipedia article to another terminal Wikipedia article. In order to both train and evaluate our model, we need some method of sampling such pairs from the graph, and a way to determine which links from the start page are in *a* best path to the terminal page. The process is a Markov process, meaning that the history of traversal is meaningless to the current problem at hand. Thus, sampling starting and target pages is all that is needed to train an end-to-end model that can do complete traversal.

We first randomly choose a target node with a view count higher than some arbitrary threshold, in our case filtering the pages down to the 3 million most viewed pages. We want this target to be something that the model can have knowledge about, and picking an article with a relatively high view count is one way to guarantee this. Then, we randomly pick a starting node by traversing away from the target node using the backward edges found in 5.1.4. We traverse using a breadth-first search (BFS) for a random number of iterations, then check if the node we've found has >5 links, and is at least a distance of two links away from the terminal node before we accept it.

Once we have a start and terminal node, we have another algorithm to determine which of the neighboring nodes of the start node (links from the current page) are in a shortest path to the terminal node. This is done by doing another BFS from the terminal node using the back edges, but keeping track of the distances for each node that has been seen during the traversal. As soon as we add the start node to the seen set, we must have also seen the all neighboring edges if they are in a shortest path to the terminal node. We then inspect the distances to each of the neighboring nodes of the start node, returning a 1 for every node that has a distance equal to the minimum distance, and a 0 for all other neighboring nodes. An example of the result of this algorithm can be seen in Figure 2. See `gas/graph_algorithms.py` for further details on our implementation.

This traversal algorithm is linear in complexity, yet the overhead of running it in Python means that sampling and scoring neighbors combined takes around 20 seconds per sample. To mitigate this performance issue, we re-implemented our graph algorithms in C++, using a custom efficient data structure through the `cppyy` (Lavrijsen and Dutta, 2016) library. Again, see `gas/graph_algorithms.py` for details. This implementation is consistently 100 times faster, which is more than fast enough to feed our NLP model and for us to perform further experiments on.

5.1.6 Data Ablation Experiments

It is not known what dataset Mistral AI used to train the Mistral-7B model (Jiang et al., 2023). For us to determine what tokens we should use to represent structures like links or references in our dataset, we needed understand how well the model responds to different formatting. A series of tests were conducted on a small portion of the “Albedo” Wikipedia article, which contained citations and various other Wikitext tags. The perplexity of the model at each token intuitively represents how many guesses it would take on average to guess the correct token given the model's output distribution. This is a measure then of how confident the model is in predicting the tokens in the sequence. We measure perplexity for each of the tokens in the input sequence to understand how similar the model's predictions were to the true text. Our hypothesis is that the model the model understands how links are formatted in Wikitext, as we think it should have been in the training dataset. Our experiments tested three different kinds of input formats: raw Wikitext, Wikitext with citations contained in `<ref></ref>` tags removed, and an entirely plain-text format with all Wikitext elements removed. These can be seen in Figure 3. In each of these three cases, the selected portion of the article text was tokenized and the model was fed the first ten tokens. From there, the masked language model generates the softmax probabilities for each token type for every element in the sequence, and we can compute the crossentropies given the true tokens. Generally, in the n^{th} iteration of the generation loop, the model was prompted with the first $10 + n - 1$ tokens of the Albedo article and was asked to predict the $(10 + n)^{\text{th}}$ token. A cumulative perplexity over the first n iterations was also calculated as $\exp\left(\frac{J_n}{n}\right)$ where J_n is the cumulative sum of the cross-entropies and n is the number of tokens that have been generated.

As seen in Figure 4a, the model is able to accurately predict the 11^{th} token, which is the word “of” following the word “ratio.” However, the token for “[[” in the formatting for the link that follows causes a massive spike in the perplexity. Interestingly, the closing token for “]]” has a very low perplexity, indicating that the model learns how to respond to the presence of formatted links after the first instance it sees. So despite having trouble generating link indicators the first time, it's conditional generation abilities greatly improve after understanding that the text contains these structures. Our hypothesis is then that this model has read plenty of Wikitext, since it after seeing the first indication of Wikitext understands responds well to structures in the rest of the page. The final average model perplexity was 3.403, which was the lowest among all three experiments. A possible

explanation for this result is that because citations and Wikitext tags follow a more rigid structure than human language, the model was able to have better performance in next-token prediction when it was working with a more rigid format.

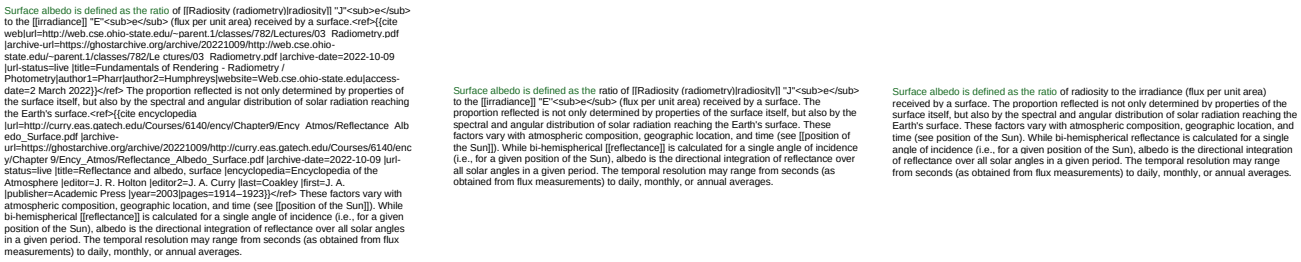
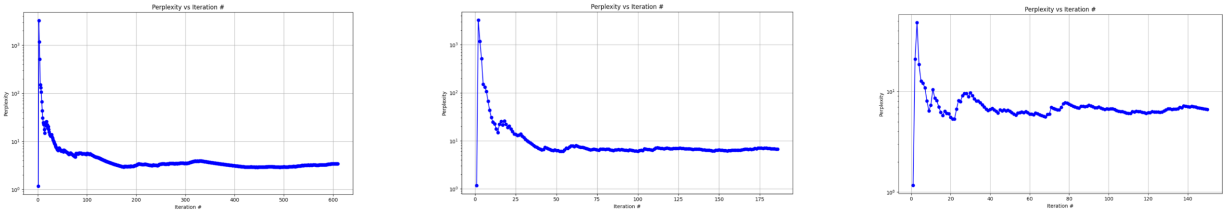


Figure 3: The first part of the text from the Wikipedia article about "Albedo" at various levels of processing. In our perplexity tests the green text is fed to the model before evaluation starts. The first version is the raw Wikitext taken straight from the Wikipedia dump. The second has references and other structures removed. The third has been simplified down to plaintext.

Figure 4b shows the cumulative perplexity over the sequence when citations were removed, where the model achieved a higher final perplexity of 6.634. A possible reason for this is that the absence of the predictable structures associated with citation links meant that the model had to make more difficult predictions since text is a more information-dense modality than links. Similar results were also obtained when the model was given a version of the Wikitext which only contained plaintext without any citations and any text in square brackets replaced with the plain text itself, as the final perplexity was 6.608.

In Figure 4a, the initial spike is due to the first encounter of the "[[" characters which the model didn't expect given that the first ten tokens of the selected portion of the Albedo article consisted of only text. In Figure 4b, the initial spike was due to an incorrect yet confident prediction of "radiant" instead of "radiosity" due to limited context.



(a) Full Wikitext (b) Citations removed (c) Plaintext only

Figure 4: Cumulative average perplexity for the Wikipedia article about "Albedo" at various levels of processing.

The results show us that the model is confident in predicting formatting tags after it's seen the corresponding opening tag, and understands that it's in a Wikitext formatted document. Thus we decide to include links in the format used by Wikipedia, with double brackets on either side, while stripping out other structures to greatly reduce the number of tokens in the input text.

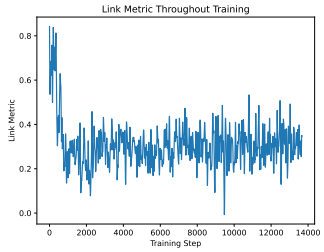
5.2 Experimental details

Our initial model is the pretrained Mistral-7B-v0.1 (Jiang et al., 2023), with vocabulary size 32,000, hidden dimension 4096, 32 hidden layers, 32 attention heads, 8 KV-heads, the SiLU activation function and a sliding window size of 4096 (which gives it a context length of 8092 tokens). Before training, we evaluate the base model using a single step of generation from a few text samples, as can be seen in Table 3. We train it with a batch size of 1 on 13671 samples on the train split of our dataset, which takes 24 hours. Since we have a massive pool to sample from, we can continue sampling without reuse, so there is no notion of epochs in our training setup. The learning rate scheduler is a CosineAnnealing method with a maximum learning rate 3×10^{-5} and annealing learning rate of 1×10^{-5}. After the model has been fine-tuned, we evaluate it on the same test set.

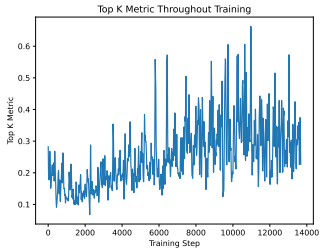
5.3 Results

The training process produces the plots in Figure 5. We notice that the ValidLink metric seems to drop quickly over the course of finetuning, before gradually improving over time. The TopK metric drops a little bit in the first few steps, but grows to be much better on average after a few thousand training samples. Further while the loss is very noisy, the moving average of its value drops significantly with training.

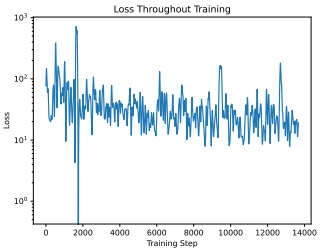
The final results in Table 3 show that the zero shot performance is better than random, but finetuning further improves the TopK metric.



(a) ValidLink Metric



(b) TopK Metric



(c) Finetuning Loss

Figure 5: Metrics over the course of training.

	ValidLink	TopK	Loss
Random Valid	1.0	0.17	N/A
Mistral-7B Zero-shot	0.761	0.22	76.75
Mistral-7B Finetuned	0.313	0.34	19.3

Table 3: Results before and after training compared to random choice.

6 Analysis

Our experimental evaluation setup shows decent performance for the zero-shot in-context learning version of Mistral-7B, and even better performance after a small amount of finetuning. However, a key issue with our training setup is that the gradient signals are very sparse compared to the total output and computation required by the language model. Despite having to build up representations and predictions for the entire sequence of text, which often is more than 12,000 tokens, only the final token in the sequence contributes to the loss function. This is in contrast to most finetuning tasks in causal language models, where the entire sequence of tokens is used to compute the model loss, and therefore also affect the weight gradients. We consider the TopK metric to be the ultimate metric of performance, since it best represents how the model is able to prioritize the right links while still being much more stable than checking if the highest probability link is optimal.

The finetuning process shows minor gains in performance, and the loss decreases as we expect. It is clear that the Mistral-7B model is quite capable, even when quantized and with quantized training. We found it to be very difficult to interpret the outputs of the model for each link, and that inspecting a single token of generation is not a great way to understand what's happening in the language model.

7 Conclusion and Further Work

We find that large language models are capable of doing better than random for the task of browsing Wikipedia, and that our finetuning objective was moderately successful in further improving this performance. Our proposed method defines a novel benchmark task for evaluating the capabilities of generative language models to navigate Wikipedia. We build a reusable dataset preprocessing pipeline alongside an algorithm with graph traversal implementations that can be used to train an LLM to perform the task of browsing Wikipedia with a specific topic in mind. Our model was able to improve by finetuning, but the training process took much longer than expected. Finetuning for only the final token in the sequence is very costly and provides a poor gradient signal.

If we had more time to work on this project, we would like to further experiment with the loss function, and to work more on the finetuning aspect of the model. A problem with the current formulation and model is that it takes a very long time to do the forward and backward pass of the model, especially for certain longer samples. In the future we would like to use more hardware or smaller models to study the asymptotic behavior of the finetuning instead of having to stop the training early. Further, we want to test the model end-to-end, deploying it for an entire search, and study if the LLM agent does better in expectation than humans. These evaluations were not possible for us to do due to time and resource limitations.

Our current model has no notion of history, and can get stuck in a loop if it is mistaken in its predictions. To overcome this issue, we would like to give the model information about its history, or to add a controller that uses the language model for traversal and keeps track of which pages it has seen before, avoiding cycles.

For examples that consist of multiple shortest paths, we could also potentially look at a rough metric for human preference alignment by designing a metric that assesses if the model puts more probability mass on the shortest path, which contains a greater number of total page views. This could work by considering $\log\left(\frac{p_1}{p_{1+n}}\right)$ where p_1 corresponds to the most relevant shortest path and p_{1+n} corresponds to the $(1+n)^{\text{th}}$ most relevant shortest path where we calculate this metric for however many other shortest paths there are between a given start and target node pair aside from the most relevant one. Some other possible weight functions could also be used as such a heuristic rather than summing the number of page views. This method might give a richer signal to the model about which links to choose compared to the binary "optimal" or "not-optimal" signals it currently receives.

References

Albert-László Barabási. 2013. Network science. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1987):20120375. Publisher: Royal Society.

Stefan Behnel, Martijn Faassen, and Ian Bicking. 2005. lxml: XML and HTML with Python.

Collin Burns, Haotian Ye, Dan Klein, and Jacob Steinhardt. 2024. Discovering Latent Knowledge in Language Models Without Supervision. ArXiv:2212.03827 [cs].

Tri Dao. 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. ArXiv:2307.08691 [cs].

Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. 2023. Language Modeling Is Compression. ArXiv:2309.10668 [cs, math].

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. ArXiv:2208.07339 [cs].

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. 2023. A Survey on In-context Learning. ArXiv:2301.00234 [cs].

Earwig. 2013. earwig/mwparserfromhell: A Python parser for MediaWiki wikicode.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. 2017. Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning. ArXiv:1702.03118 [cs] version: 3.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. ArXiv:2312.10997 [cs].

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].

HuggingFace. 2024. Hugging Face – The AI community building the future.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. ArXiv:2310.06825 [cs].

Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, Jiyan Yang, Jongsoo Park, Alexander Heinecke, Evangelos Georganas, Sudarshan Srinivasan, Abhisek Kundu, Misha Smelyanskiy, Bharat Kaul, and Pradeep Dubey. 2019. A Study of BFLOAT16 for Deep Learning Training. ArXiv:1905.12322 [cs, stat].

Jonathan Larson and Steven Truitt. 2024. GraphRAG: A new approach for discovery using complex information.

Wim T.L.P. Lavrijsen and Aditi Dutta. 2016. High-Performance Python-C++ Bindings with PyPy and Cling. In *2016 6th Workshop on Python for High-Performance and Scientific Computing (PyHPC)*, pages 27–35.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. ArXiv:1711.05101 [cs, math].

Rada Mihalcea and Andras Csomai. 2007. Wikify! linking documents to encyclopedic knowledge. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, CIKM '07, pages 233–242, New York, NY, USA. Association for Computing Machinery.

David Milne and Ian H. Witten. 2008. Learning to link with wikipedia. In *Proceedings of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 509–518, New York, NY, USA. Association for Computing Machinery.

Mark Newman, Albert-László Barabási, and Duncan J. Watts. 2006. *The Structure and Dynamics of Networks:*. Princeton University Press. Google-Books-ID: 6LvQIIP0TQ8C.

Polars. 2023. Polars.

Alec Radford and Karthik Narasimhan. 2018. Improving Language Understanding by Generative Pre-Training.

Aju Thalappillil Scaria, Rose Marie Philip, Robert West, and Jure Leskovec. 2014. The last click: why users give up information network navigation. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 213–222, New York New York USA. ACM.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. RoFormer: Enhanced Transformer with Rotary Position Embedding. ArXiv:2104.09864 [cs].

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. ArXiv:2307.09288 [cs].

Unknown. 2022. The Wikipedia Game.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. On the Planning Abilities of Large Language Models : A Critical Investigation. ArXiv:2305.15771 [cs].

Robert West. 2010. *Extracting semantic information from Wikipedia using human computation and dimensionality reduction*. Ph.D. thesis, McGill University, Montreal, Québec.

Robert West, Joelle Pineau, and Doina Precup. 2009. Wikispeedia: an online game for inferring semantic distances between concepts. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, pages 1598–1603, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Wikipedia. 2023. Wikipedia:Database download. Page Version ID: 1186807991.

Wikipedia. 2024a. Help:Link. Page Version ID: 1205116426.

Wikipedia. 2024b. Help:Wikitext. Page Version ID: 1212475524.

Wikipedia. 2024c. Wikipedia:Six degrees of Wikipedia. Page Version ID: 1203914048.

Wikipedia. 2024d. Wikipedia:Wiki Game. Page Version ID: 1199308715.

Biao Zhang and Rico Sennrich. 2019. Root Mean Square Layer Normalization. ArXiv:1910.07467 [cs, stat].