Adaptive GISTs to Create a Cache for Retrieval Augmented Generation

Gautam Pradeep
Stanford University
gpradeep@stanford.edu

Arjun Karanam
Stanford University
akaranam@stanford.edu

Ronak Malde
Stanford University
rmalde@stanford.edu

1 Introduction

1.1 Motivation

Large language models, due to the nature of the underlying attention mechanisms[9], have limited context windows, and often we need to reference information that exceeds that window. As the use cases for language models veer beyond simple question-answering tasks, this problem occurs more and more. For example, we might want to ingest an entire codebase to ask questions about what it does or how it works. We may aim to ingest several Wikipedia articles for question answering or knowledge retrieval, where this external information well surpasses the context window.

We do see models with larger and larger context windows, such as GPT-4 [7] and Claude-2, but these come at a cost. It is difficult to train and run inference on LLMs with massive context windows because the runtime and memory requirements scale quadratically with the sequence length. Additionally, research has shown that not all information is equally attended to as context windows grow, casting doubt on their effectiveness in encoding large contexts[4].

With large models like ChatGPT being queried millions of times per day, encoding the base instruction prompt over and over with the quadratic self-attention mechanism proves infeasible. One strategy to get around this constraint is to attempt to fit a larger context into a model that only supports a smaller context. This can be achieved, for example, with Parallel Context Windows in Ratner et. al. [8], in which attention is computed in chunked windows of the large context and then attended together. One limitation of this approach, however, is that the runtime complexity during model inference still quadratically scales to the entire large context length. One can also fine-tune or distill the model to behave similarly to the original model without the prompt through parameter-efficient adaptation methods (Lester et. al. [3]). However, this strategy forces model re-training for each new prompt.

Another solution, one that reduces the model inference runtime complexity, is to selectively choose what information to include in the context window based on the current prompt. This can be done using Retrieval Augmented Generation (RAG), in which external information is stored in a vector database, and then parts of this information is selectively included in the context based on similarity to the current prompt. RAG models that combine pre-trained parametric and non-parametric memory have been shown as superior to parametric-only baselines (Lewis). However, the problem is that we are then still limited by the original context window and can only selectively include so much information. Our goal is to find a solution that solves both of these problems - one in which we can pass large amounts of information to the model (larger than the context window), while also not scaling quadratically with that amount of new information being added.

1.2 Related Works

As discussed, Retrieval Augmented Generation (RAG) combines pre-trained parametric and non-parametric memory for language generation and offers a solution to utilizing additional knowledge bases for more refined outputs compared to the standalone pre-trained model. Research has exper-

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

imented with RAG models in varying approaches such as using the same retrieved document to generate a complete sequence or drawing a different document for each target token. Besides the naive implementation of RAG, many popular strategies use agents that have access to RAG tools to reason over multiple steps and decide when and how to use these various tools. This methodology has been shown to allow agents to answer far more complex queries. The issue with an agent-powered RAG involves speed and cost where every reasoning step taken by an agent is a call to an LLM.

RAG requires an operation to determine the closest document to the queried prompt across all the documents for each task of text generation and proceeds to execute an inference whose runtime complexity scales quadratically with sequence length. This is extremely problematic for large sequence lengths, particularly for RAGs, given that many documents referenced will have long sequences that will be prepended to the context. This creates a challenge in that many models have context lengths much smaller than the length of a long retrieved document.

Techniques such as positional interpolation have been utilized to attempt extending the context window of RoPE-based pre-trained LLMs such as LLaMA to up to 32768 with minimal fine-tuning in Liu et. al[5]. Although larger context windows seem appealing, the RAG research demonstrates the best results arise when fewer, more relevant documents are given to the model in the context rather than a large number of unfiltered documents[5]. We predict we can further improve RAGs by implementing gist tokens to encode longer-sequence length passages (e.g., the retrieved documents) and applying a robust caching strategy to access relevant documents quicker and drastically reduce the length of document-specific information allocated in the context window.

2 Problem Statement

The issue of better utilizing the limited space of context length during RAG and increasing the spatial and temporal efficiency of knowledge base retrieval for a large language model is an open problem. Recent research investigates compressing prompts with gist tokens to encode textual instruction into a single representative token to reduce memory utilization[6] and appears promising at enabling larger contexts without a quadratic increase in computation.

In this project, we show that we can further improve RAGs by implementing gist tokens to encode longer-sequence length passages (e.g., the retrieved documents) and applying a robust caching strategy to access relevant documents quicker and drastically reduce the length of document-specific information allocated in the context window.

2.1 Dataset

We used the Wikipedia TriviaQA Dataset made by the University of Washington [2], which asks trivia questions that are obscure enough that a language model could not possibly contain all of the information in its weights. It contains over 650K question-answer-evidence triples, where the evidence is sourced from relevant Wikipedia articles. We used the existing HuggingFace datasets train test validation split for our procedures.

3 Approach

3.1 Gisting

We build much of our work on top of the recent "Learning to Compress Prompts with Gist Tokens" research by Mu, et. al[6] In this paper, an LLM is trained to compress prompts into smaller sets of what they call "gist" tokens. These gist tokens are computed, and the cached and reused for computational efficiency over multiple language model calls. The primary strength of gist tokens is that they are computed via a variation in the attention mask of a transformer, meaning that at their first iteration, that are just as computationally efficient as a regular finetuning step.

Mathematically, we can describe it as follows. Let's say we have an instruction-following dataset $D = \{(t_i, x_i, y_i)\}_{i=1}^{N}$, where t is a task encoded with a natural language prompt (e.g., "Translate this to French"), x is an (optional) input for the task (e.g., "The cat"), and y is the desired output (e.g., "Le chat"). Given a (usually pretrained) LM, the aim of instruction finetuning is to learn a distribution $p_{LM}(y \mid t, x)$, typically by concatenating t and x, then having the LM autoregressively predict y. At

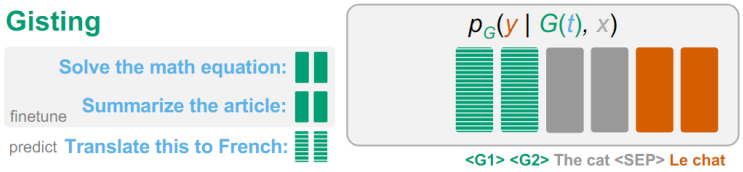

Figure 1: Compressing instructions into a gist token

inference time, we can *prompt* the model with a novel task t and input x, decoding from the model to obtain its prediction.

Gisting serves to reduce both the inference-time cost of prompting p_{LM} with t and the train-time cost of learning a new model. Gisting involves learning a compressed version of t, denoted as $G(t)$, which leads to faster inference from $p_{LM}(y \mid G(t), x)$ than from the original $p_{LM}(y \mid t, x)$. The compressed version $G(t)$ consists of *gist tokens* that are fewer in number than the tokens in t but still induce similar behavior from the LM. The transformer activations on top of $G(t)$ can be cached for computational efficiency. Gisting is designed to generalize to unseen tasks; given a new task t, one can predict and use the gist activations $G(t)$ without additional training.

3.2 Gisting for Retrieval Augmented Generation

We hypothesized that applying gisting to contextualize semantics from documents in concise packaging can further augment RAG. As we saw in Berchansky et al[1]. research, retrieval augmented reader models could be further optimized via token elimination. Similarly, learning a compressed version of the prompt and instruction using gist tokens may be able to maintain the quality of RAG but drastically reduce the bottleneck of context window size and increase efficiency.

This does present some challenges, however. In their original gisting paper, Mu et al.[6] found that instructions that contain exact information (i.e extract 3 words) did not gist very well. This is most analogous to our setting of Retrieval Augmented Generation, where users will often expect exact key word matches. As such, our main obstacle will be to employ the gisting strategy while achieving near-state-of-the-art results using RAG techniques techniques alone.

4 Experiments

4.1 Baseline

The baseline model we used was Google's flan-t5-base variant of the FLAN-T5 available on HuggingFace, trained on a large corpus of unfiltered and uncensored data. The model was based on the pre-trained T5 and fine-tuned with instructions for better zero-shot and few-shot performances. The baseline model has about 248M parameters with F32 precision. We fine-tuned the model on Google Cloud Platform when running experiments with instruction and contextualized data for retrieval and gisting.

4.2 Experimental Procedure

Using the Gisting objective described in the previous section, we finetuned upon the baseline model (flan-t5-base). All finetuning was done for 16000 steps, which was the recommended step size published for the model. Every 1000 steps, we evaluated on a validation set of data, and employed early stopping by taking the checkpoint that had the highest RougeL score on the validation set.

In our experiments, we also found that the Gist finetuning performance is very sensitive to the initial learning rate. As such, we first ran a hyperparameter search over learning rates and evaluated on the validation dataset, and found 1e-3 to be the optimal learning rate.

4.2.1 Different Model Strategies

We evaluated four distinct strategies to the question answering task.

Performance of Model Varying Fine-tuning Learning Rate		
Learning Rate	Cosine Similarity	% Exact Match
5e-5	0.398	4%
5e-4	0.441	7%
1e-3	0.478	13%

Table 1: Varying learning rate induced changes in the cosine similarity and exact matches of the model to ground-truth data

1. **Vanilla Flan T5 model**. Base pretrained model that is just asked the question, no additional context given.
2. **RAG Flan T5 model**. Base pretrained model that is given context from the relevant Wikipedia article, and also asked the question.
3. **Instruction Gist model**. Model finetuned on Alpaca instruction dataset, as in the original gist paper.
4. **RAG Gist model**. Model finetuned on Wikipedia Open QA, where the article context is compressed into the Gist token.

4.2.2 Varying Number of Gist tokens

The original Gist paper allows functionality to vary the number of Gist tokens used to encode information. The original Gist paper found that increasing the number of Gist tokens did not have a significant result in the model's ability to finetune on short instructions. Nevertheless, we chose to revisit this experiment in this project, since the tokens have to compress a larger amount of information than just a small instruction, so perhaps increasing the number of gist tokens will increase performance. We finetuned models with various numbers of Gist tokens in a row to assess performance by comparing the performance of the t5-flan-base model, with a learning rate of 1e-3, gisted with one, two three, and five tokens.

4.3 Evaluation Metrics

We evaluated all models on the test split of the Wikipedia QA dataset. We then used Microsoft's MiniLM L6 v2 sentence transformer model to generate an embedding of the predictions and the answer. We then used this embedding to create two metrics:

1. **Cosine Similarity** - The average cosine of the angle between the prediced and ground truth embeddings. Ranges between 0.0 and 1.0, where 1.0 is the best (exact match).
2. **Percent Exact Match** - The percentage of predicted answers that got a cosine similarity of 1.0, ie were exact matches.

4.4 Results

The first experiment evaluates the different model strategies outlined in section 4.2.1. We tested each set of models on different lengths of context input. Since the vanilla model receives no context and only the question, its input did not change. For the other three models, the Wikipedia article was truncated to the first n tokens to become the "context". The RAG model was then fed in the context and the question concatenated. For the two Gist models, the context was compressed into the gist token, and the model was given the gist token and the question. Due to constraints of the T5 base model, we were limited to 500 tokens for the context input. Table 2 reports both the cosine similarity and the percent exact match of these models on the test dataset.

The second experiment we ran is varying the number of gist tokens used in finetuning, as explained in Section 4.2.2. For all of these, we finetuned on the Wikipedia QA dataset and gave a context input

Comparing Model Strategies						
Context Input Length	25 tokens		100 tokens		500 tokens (max)	
Model Strategy	Cosine Sim	% EM	Cosine Sim	% EM	Cosine Sim	% EM
Vanilla	0.375	3%	0.375	3%	0.375	3%
RAG	0.463	11%	0.523	10%	0.577	16%
Instruction Gist	0.331	1%	0.337	1%	0.340	1%
RAG Gist	0.460	12%	0.467	12%	0.474	13%

Table 2: Evaluation of different model strategies for different context input lengths

of 500 tokens to be compressed, since this achieved the best results from the previous experiment. The cosine similarity and exact match scores for these different models are shown in table 3.

Performance of t5-flan-base Model Varying the Number of Gist Tokens		
	Cosine Similarity	Mean Exact Matches
1 gist token	0.474	13%
2 gist tokens	0.475	13%
3 gist tokens	0.458	15%
5 gist tokens	0.470	16%

Table 3: Varying number of gist tokens in finetuning

5 Analysis

Comparing the model strategies across the vanilla (baseline), RAG, gisting the instruction, and gisting with RAG, proved to us interesting results. As shown in Table 2, we analyzed the cosine similarity and percentage of exact matches comparing the generated outputs and ground-truth answers across the four models and three different context input lengths. The vanilla model is not provided with any context tokens so it is explainable why its performance remains the same across the input lengths. However, for the three other model strategies, each one improves with longer context inputs. This is reflected in both the cosine similarity and percent of exact matches, although the latter was less sensitive to change. Longer context inputs simply provide more context which augments the performance of the model while also increasing the computational cost of the operation. The data also shows that the exact match performance of the instruction gist model was the poorest consistently which is understandable as no additional context was provided and only the set of gist tokens was utilized to generate.

RAG consistently performed with higher cosine similarity compared to the other models as it is provided with "ungisted" documents for augmented retrieval. However, on 25 and 100 token context lengths, the RAG Gist achieved a higher percentage of exact matches than the default RAG. This suggests that gisting does well at capturing context in shorter lengths (similar to the context lengths that were used in the original Gist paper [6], but it struggles as the context windows grow larger and larger.

Importantly, the data may at first indicate a high performance of RAG across the board, but we can also notice that RAG gisting performs better or about the same in relation to the percent of exact matches and is only slightly below (6%) the default RAG. This is impressive as the RAG gisting is able to accomplish similar performance with only one additional token (or number of gist tokens) rather than an entire Wikipedia article in context as in RAG. This relates to a much lower computational cost and runtime barrier with gisting.

From Table 3, we examine how increasing the number of gist tokens may affect the performance of the t5-flan-base model. As mentioned earlier, the original gisting paper suggested altering the number of gist tokens did not affect their performance but we hoped to revisit this as we are applying gisting

to a very different task. Increasing the number of gist tokens from 1 to 2 did not make a substantial change, and adding another gist token weakened the performance of the model (cosine similarity dropped over 2%). Finally, we see having five gist tokens increased the performance from the prior run but still was sub-optimal to only having one or two gist tokens. The interesting result arises from the maximum percent of exact matches in our experiments was 16%, coming from the 5 gist token model, beating the others. The percent of exact matches appears to be trending as an increase as we include more gist tokens, but the results are based on too few samples to make critical conclusions. We agree that given our current level of data, we stand with the original paper's determination that the number of gist tokens does not play a significant role in the performance of the model.

6 Conclusion

In conclusion, in this research, we found that the gisting framework introduced in Mu et. al.[6], which was originally intended to summarize and cache prompt instructions, can somewhat be extended to Retrieval Augmented Generation as well. Gisting outperforms a model with no context given at all, but doesn't perform as well as a model that is given the full context, as is to be expected. If this were to be integrated into a Retrieval Augmented System, one would need to make the tradeoff between decreased accuracy in exchange for increased speed (as our method skips the retrieval step of retrieval augmented generation via caching). However, there are still many ways to improve and extend our work.

6.1 Limitations

In this project, due to compute and cloud resource constraints, we were only able to train the Flan-T5-Base model, which has about 248 million parameters. However, in the original gisting paper, they achieved far better results overall using the Flan-T5-XXL model and Llama 7B, which have 11 billion and 7 billion parameters, respectively. We would like to assess our approach on these larger, more capable models, to see what patterns still hold for larger models, and if certain weaknesses can be overcome via emergent properties of larger models.

6.2 Future Work

Now that we've been able to show that gisting contexts is somewhat effective at helping question-answer systems work with long context, future work will revolve around making this better, and implementing it into RAG systems.

6.2.1 Direction 1 - Pairing with Retrieval Augmented Generation

Here, we can leverage the RAG methods that we outlined in the first part of the paper. We can store certain documents as gists within the context of the model, and use RAG to query for the rest. This then brings up the question though - how will the model know when it should just rely on its gist, and when should it query externally using RAG? For this, we hope to look towards the paper "ToolFormer: Language Models Can Teach Themselves to Use Tools", by Schick et. al. (2023). In this paper, they show how one can fine-tune a language model to use "Tools," i.e insert tokens corresponding to an API call in order to fetch external information. We would use this method to finetune our model to output a RAG token instead, so that it can perform RAG when it needs to, and solely utilize the Gist otherwise.

6.2.2 Direction 2 - Dynamic Updating of Gists

Additionally, it's highly unlikely that the same documents that are gisted end up being the documents that actually need to be accessed repeatedly. Can we dynamically update the set of documents in the gist based on which parts are accessed the most? Here, we would set an interval at which the model is unchanged (as changing requires re-gisting the documents, and doing this process at every inference would be very expensive). At the end of that interval, we would identify the information that is accessed the most via simple counting statistics, and gist those to ensure the maximal speed-up. The exact parameters of this algorithm can be tuned over time.

6.3 Code

Code can be found at this Github Link: https://github.com/QuantumArjun/rag_gisting

References

[1] Moshe Berchansky, Peter Izsak, Avi Caciularu, Ido Dagan, and Moshe Wasserblat. Optimizing retrieval-augmented reader models via token elimination. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1506–1524, Singapore, December 2023. Association for Computational Linguistics.

[2] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[3] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021.

[4] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023.

[5] Xiaoran Liu, Hang Yan, Shuo Zhang, Chenxin An, Xipeng Qiu, and Dahua Lin. Scaling laws of rope-based extrapolation, 2023.

[6] Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens, 2023.

[7] OpenAI. Gpt-4 technical report, 2023.

[8] Nir Ratner, Yoav Levine, Yonatan Belinkov, Ori Ram, Inbal Magar, Omri Abend, Ehud Karpas, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. Parallel context windows for large language models, 2023.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.