An Alternative to Perplexity for the Evaluation of Long-context Large Language Models

Chiel de Jong - 7639112
June 2024

1st supervisor: Denis Paperno
2nd supervisor: Yupei Du

Bachelor Thesis Artificial Intelligence

Universiteit Utrecht

Abstract

With the advent of Large Language Models and increasingly longer context windows, comes the need for new evaluations that can capture model performance at these lengths. As an alternative to perplexity, the Rank Based Perplexity is proposed which calculates the scores of tokens based on their position in an an array of possible tokens, ranked by their probabilities. The temperature of an LLM, which directly influences the perplexity of a model, does not affect the Rank Based Perplexity. The metric was tested on the four latest OpenAI GPT models (4-o, 4-turbo, 4 and 3.5-turbo) at increasing context lengths with a supplementary perplexity approximation. It successfully captured GPT-3.5-turbo being the worst model in the series and both metrics showed an increase in performance as context length grew, where GPT-4-o had the greatest increase. Repeated runs on the same context length but at different starting positions reveal large variability in the scores and highlight the need for more extensive evaluations to determine the true efficacy of the proposed metric. The code and saved data can be found here.

Contents

1 Introduction

Large Language Models (LLMs) like ChatGPT, have been at the forefront of AI research due to their exceptional performance in a wide range of natural language tasks. The models excel at, among other things, coding, reasoning and summarization. As the models have grown larger and have been trained on more data, their performance has improved substantially. Large companies like OpenAI, Google and Meta are spending striking sums of money on training and running these models. Modern day LLMs like OpenAI's GPT-4-o and Google's Gemini 1.5 pro have even been upgraded to process more modalities than just text, extending to images, video and audio as input, paving the way for further applications of the models.

1.1 Long-Context Large Language Models

The context window of an LLM is the number of tokens that it can process as input in a single turn. A larger context window allows models to process large quantities of data that it has not been previously trained on and understand and reason about it. As the models have advanced in size, performance, efficiency and modality, great advances have similarly been made in increasing the size of context windows. Anthropic's Claude 3 models[6] and Google's Gemini 1.5[20] already boast an impressive 1M and 10M tokens as input respectively. Both models are currently only available at the longest context lengths to a select group of developers.

The models are trained on vast amounts of data in the form of text. This gives us the pre-trained model which is then further fine-tuned for specific use cases like being a chat assistant. When such a model is given new data in the form of input, they show remarkable learning abilities and comprehension. A longer context window then allows us to give more examples, documents and instructions to a model. The ability to feed a model large quantities of new data and have it grasp it without needing more training, represents a valuable avenue possibly applications. Take for example, a model that is given every scientific paper in a specific field of research as input. Subsequent summarization, reasoning and elaboration over the data could significantly aid human research objectives.

Another phenomenon promoted by larger context windows is in-context learning (ICL). This is where a model learns a certain task or skill just from the input prompt. This can be done by giving a few demonstration examples followed by a query question. ICL does not require any additional training or parameter updates like few-shot learning[8]. This in combination with the fact

that LLMs are not often open sourced or cannot be run locally by average people, means that ICL provides a way to deliver a Model-as-a-Service[19], where the models can be used effectively by clever prompting techniques without adjustment.

Researchers at Google DeepMind also found gains in performance when going from few-shot ICL to many-shot ICL[1]. The same team testing the same model, Gemini 1.5 pro, found that the model was able to learn a language, kalamang, that it had not been trained on from an input prompt of 250k tokens[20]. It subsequently achieved results that were comparable to a human learner, highlighting an additional use case of long-context windows in LLMs.

However, researchers found that 'extended context models are not necessarily better at using input context.'[14] They saw a rapid degradation of performance when the models were asked to reason about information around the middle of their input context. The performance was best when the required information was at the beginning or the end of the input, similar to the serial position effect witnessed in humans. The increasingly larger context windows must therefore be evaluated to determine whether they effectively comprehend the entirety of the input.

1.1.1 Existing Evaluations

The evaluation of models is a critical component used both during and after development processes. An evaluation for a language model consists of a task that the model must complete and a way to score the corresponding output. LLM output evaluation methods can be split into three broad categories; in ground-truth-based evaluation a model performs a task for which there is a defined correct answer. This type of task allows for an objective final evaluation. With LLM-as-judge evaluation, a model's output is evaluated by another language model. Finally, user-facing evaluation employs humans to evaluate the outputs of models.

The perplexity (PPL) is a common example of a ground-truth-based evaluation. It is regularly used to evaluate how well a model performs based on its token probability predictions. However, a recent study[11] found a discrepancy between better (lower) PPL scores and other long-context evaluation benchmarks. Here, a comparison between three long-context models (YARN-7B-128K, Yi-6B-200K and LongLoRA 7B-100K) found that the model with the best PPL (YARN) had the lowest scores on almost every other benchmark. They subsequently posed that the PPL might only reflect a model's local language abilities rather than its full-text understanding. In any case, a call is made towards the creation of additional evaluation metrics that provide more accurate insight into model performance at long contexts.

Another prevalent ground-truth-based evaluation metric is a retrieval task often referred to as needle in a haystack[4] (NIAH). Here a model must retrieve one or multiple *keys* with corresponding *values*. These 'needles' are inserted into long distractor texts, 'haystacks', followed by a query that asks the model to retrieve the *keys* and their *values*. The task is iterated over multiple depths, i.e. locations in the text, and multiple context lengths. The synthetic nature of this tasks allows it to be easily expandable to long-contexts. Recent work showed that despite models achieving perfect results for needle in a haystack tasks, they failed to maintain their performance in other tasks like question answering, aggregation and multi-hop tracing[10]. The tests were done at context-lengths ranging from 4k-128k tokens; they show that, although NIAH is an effective and scalable ground-truth-based task, it fails to capture more nuanced text understanding abilities.

Several longer context benchmarks have been proposed like LongBench[5] and L-Eval[3]. These are fixed up to 32k tokens however, and so researchers proposed Ada-LEval[21] which extends this up to 128k tokens with TSort and BestAnswer benchmarks. Researchers have also created ∞Bench[25] which consists of tasks that are, on average, longer than 100k tokens. ZeroSCROLLS[18], which improves upon the previous SCROLLS benchmark, is designed to evaluate long text understanding and does this with 10 different tasks, each requiring reasoning. LongICLBench[13], a benchmark for long in-context-learning evaluation, was also designed after the need for more comprehensive evaluations was recognized. This represents a more sophisticated long-context evaluation that requires models to actually reason over the retrieved data.

When using an LLM as a judge, it is important to note the biases that can occur. LLMs have been shown to prefer their own generation of models when evaluating outputs[16]. The researchers state that this is due, in part, to the models exhibiting self-recognition. As can be seen in table 4 of Gemini 1.5 pro's testing[20], the outputs of Gemini models and Claude 2.1 were evaluated by both Gemini 1.5 pro and humans. The Gemini model as an evaluator consistently over-scored itself and under-scored Claude compared to the human evaluation. When taking this into account, an LLM-as-a-judge can still be an effective and relatively cost-efficient method for task output evaluation.

Human evaluation of outputs can give an accurate picture of an model's performance. An example is the LYMSYS Chatbot Arena where users ask questions and are told to choose between two (unnamed) outputs. This is however a very time-consuming method and requires a diverse group of unbiased participants to manually give input questions and rate the outputs.

Especially at long-context window tasks, to achieve qualitative results this method approaches infeasibility. For example, a 128k token input length equates to reading roughly 90k words i.e. a book of average length. Therefore, despite human evaluation representing a reliable evaluation method, the long-context tasks would require humans to digest prohibiting quantities of text.

1.1.2 Existing Long-Context LLMs

OpenAI's GPT-4-o and GPT-4-turbo are widely used proprietary LLMs that now support a context window of 128k tokens. Kimi-Chat developed by Moonshot AI[2] has a context window of up to 200k tokens. Nous Research's Mistral 7b has been extended using YaRN[17] to support a context window of 128k tokens. Anthropic's Claude 3.5 sonnet has a context-window of 200k tokens and its Claude 3 models can process up to 1M tokens. Google's Gemini 1.5 pro can take 10M tokens as input. These last two models are both proprietary and are not yet broadly available.

Both the Claude 3 models[6] and Gemini 1.5 pro[20] claim near perfect accuracy when performing a needle in the haystack evaluation at 10M and 200k tokens respectively. It is a trivial evaluation, one that Claude even identified as being synthetic in nature, but an evaluation nonetheless. In Gemini's testing it also seemed to be successful in long-context, i.e. 710k tokens, question answering where it needed to use both retrieval and reasoning to answer correctly.

1.2 A New Evaluation

This paper aims to create a new evaluation metric for LLMs that can be used to supplement other benchmarks for the testing of long context performance. This metric will then be tested on an array of LLMs, each at different context-lengths. Specifically, the models to be tested will be increasingly state of the art OpenAI GPT models as their API allows for the requesting of up to 20 logits. The metric, named Rank-based Perplexity (RBP), will be similar to PPL but a different scoring function will be used that removes the temperature as a variable. This will be done by calculating token scores based on the ranking of tokens according to their probabilities and not the probabilities themselves. A sample of 3 scoring algorithms, with a supplementary PPL approximation, will be used to calculate and compare the scores of a model at a given context length. The models will be tested at increasingly larger context lengths up to and including 16k tokens. This will allow for a comparison between a model's performance at different context

lengths as well as an inter-model comparison.

After the relevant theoretical background has been discussed, the scoring functions are described in detail. The algorithm has been implemented in python and parameters and specific implementations, like an efficiency solution, are also described in the methodology. The models will be tested on two sets of data, with 30 samples at each increasing context length, doubling from 64 up to 4096 tokens. The RBP was able to capture increasingly better scores at longer lengths, as was the PPL approximate. Both the PPL and the RBP reflected GPT-3.5-turbo's expected worst performance and indicated that GPT-4-o had the best performance increases with context length. The small sample size due to API inference costs call for larger, longer and more diverse testing of the RBP to determine its candidacy as a long-context evaluation.

2 Theoretical Background

2.1 Generative Language Models

Pre-trained Language Models (PLMs) with model sizes usually exceeding 6-10B parameters are typically denoted as Large Language Models (LLMs)[15]. The transformer architecture has been the backbone of nearly all modern generative text models, like the Generative Pre-trained Transformer (GPT). As these models scaled up with parameter size and training data, they exhibited "emergence", characterized by huge leaps in quality of text generation, -reasoning and -learning.

Transformers are based on deep neural networks and, as with all deep-learning, the data must first be converted to vectors i.e. arrays of numbers. For these models to effectively process text, the words are first tokenized into smaller parts using tokenizers. These tokens are then mapped to high-dimensional vector embeddings that capture the meaning of individual tokens. A dataset of text can in this way be processed and represented as an array of vectors.

The transformer uses an attention mechanism to achieve the superior performance and parallelizability that makes LLMs dominant in language tasks. Specifically, a self-attention mechanism updates the vector representation of a sequence of words by relating different words in the sequence. The attention mechanism takes a full sentence or sequence into view, and changes the values of this array of vectors, based on the tokens (i.e. their vectors) and their location and relation. A well-trained attention block takes the standard vector representation of a word and calculates what you need to add, based on the context, to arrive at a more accurate vector representation. This process is then repeated such that the updated vectors are updated again, based on the other previously updated vectors. This method takes the rich and varied context that is inherent in language into account to create a more accurate vector representation of a sequence of words.

The network is trained on huge quantities of text by constantly trying to predict the next token based on the previous ones. Back-propagation is used to update and improve the model's parameters. Once trained, the models are made to generate the next most likely token of a sequence that is fed as input. They do this by generating logits, that all have different probabilities corresponding to different tokens. After various functions have been applied to these probabilites, i.e. temperature and softmax, a token is chosen based on its probability. High probability tokens are more likely to be chosen as the final token. The choices that the model makes are decoded

by the same tokenizer back into a text string that is returned to us as the final output.

2.2 Temperature

The temperature is a variable that affects the distribution of logits, i.e. the token probabilities, before the softmax function is applied to them. A low temperature means the logits will be sharpened and the final distribution will have a more peaked form. A higher temperature leads to a flattening of the probabilities which results in a model being more likely to choose from tokens that have lower probabilities. The temperature is a variable that can be adjusted and always has to be set to something, yielding a more concise or 'creative' output depending on its value. After the temperature has updated the logits as described above, a softmax function is used to create a probability distribution by making the probabilities sum to 1. Below the temperature t can be seen to influence the logits z:

$$\text{Softmax}(z)i = \frac{\exp(\frac{z_i}{t})}{\sum j^n \exp(\frac{z_j}{t})} \tag{1}$$

2.3 Perplexity

A common measure of LLM performance is the perplexity (PPL), defined as the exponentiated average negative log-likelihood of a sequence. This gives an estimate of how well a model can predict the next token, based on the preceding tokens. The perplexity PPL of a sequence of tokens $T = [t_0, t_1, t_2, ..., t_n]$ can be calculated by using the model's generated probabilities according to the following formula:

$$\text{PPL}(T) = \exp\{-\frac{1}{n}\sum_{i}^{n} \log p(t_i|t_{<i})\} \tag{2}$$

Where $\log p(t_i|t_{<i})$ is the log-probability of seeing token t_i given the preceding tokens $t_{<i}$. As we saw above, the temperature directly influences the final probabilities that we use to calculate the perplexity. This is important to take into consideration when evaluating LLMs.

2.4 Long-Context Alternatives

The self-attention mechanism that is inherent to LLMs induces a quadratic time complexity for a linear increase in the input length[12]. This means that

both the compute and memory footprint of a model increase quadratically with respect to the input. For this reason, a number of alternatives have been proposed to reduce this increase in complexity. Some of these include sparse attention mechanisms[23] and linear self-attention approximations[22].

Most modern LLMs including Gemini 1.5 use a Mixture of Experts (MoE) architecture to delegate tasks to smaller expert networks, leading to gains in efficiency in both training and inference. A gating network chooses what expert networks to delegate parts of the prompt to. This means that only a subset of the total larger network required to perform computations. MoE is often used in combination with the previously mentioned techniques to be more computationally efficient, this being more pronounced at large context lengths.

A practice that is common is fine-tuning a model for a specific task. Fine tuning is when a pre-trained LLM is trained further on more task-specific data. This has been shown to increase the performance on these downstream tasks, but also decrease the model's ability to generalize[24]. Fine-tuning is costly but can be used as a way to extend LLM context-windows by fine-tuning the models on long texts[7]. This is also prohibitive as the number of such extremely long texts is limited, and the performance degrades due to the dispersion of the attention heads. Methods like LongRoPE[7] and YaRN[17] have made progress towards removing these limitations.

Retrieval Augmented Generation (RAG) is a technique that involves creating a database in which large quantities of factual information are stored. The database is then semantically searched and the relevant information is passed on to the LLM to generate its final response. It is a cost-effective method of providing the model with relevant information which it can cite, and it was proposed as a way to reduce hallucinations and give real-time information to models without the need for more training[9]. RAG is difficult to implement but once it is in place it could be an effective way to, for example, reduce the total context input length by storing all relevant data in its database. However, in Gemini's testing[20] it was found that, unless the model was trained on the data specifically, a long context input outperformed RAG when it had to reason across long range dependencies. While RAG is therefore good for retrieval and final output of desired information, it is currently limited by an absence of inter-information reasoning capabilities which a successfully implemented long-context can provide.

3 Methodology

3.1 Rank-Based Perplexity

When a model generates a token distribution it assigns probabilities to each token. Ranking each token by its probability then gives us a rank of tokens that are increasingly less likely to occur according to the model. Given a reference dataset of text, we can compare the reference token to the predicted token ranking and determine its place in the ranking. According to this place in the ranking a score can be calculated where a higher rank will result in a higher score. The scores will be in the range $[0, 1]$ with 1 being the highest score. The resulting scores will then be averaged over the amount of comparisons. There are multiple ways to score an item in a ranking and these will be discussed below, together with the algorithm and its python implementation.

3.1.1 Scoring

There are multiple ways to score a ranked item and we will use a combination of several alternative methods. The linear score provides a simple score that decreases linearly in relation to a decreasing rank, normalised by the length of the ranking, i.e. the number of logits. The reciprocal score sees a steeper decrease in scores that converges to the linear score. It does not take the total number of logits into account, and punishes high-ranked but incorrect tokens faster than the linear score. The non-linear score is calculated using a hyperparameter α that determines the slope of the scores. A lower value of 0.1 gives overall better scores with decreasing ranks. When $\alpha = 0.3$, lower rankings are punished to a greater extent. Only the reciprocal score punishes higher ranks faster but then gets passed by the $\alpha = 0.3$ non-linear score.

In this analysis an average of the aforementioned scores will be used in combination with the perplexity approximate. The single scores for all the runs will be available in the appendix and the entire dataset of scores, tokens and probabilities can be found in the GitHub. Using a combination of the scores could give a more balanced final evaluation that captures multiple aspects of ranking performance. Any score can then also be emphasized or ignored post hoc.

Given is a sequence of tokens $T = [t_0, t_1, t_2, ..., t_n]$ where n is the amount of tokens in the sequence. Take l to be the number of logits or logprobabities that we receive from the model for each token. Given a $ranking = \{1, 2, 3, ..., l\}$ and r_i to be the rank of token i.

1. **Linear Score**

   The linear score S_L of a sequence of tokens T can be calculated as such:

$$S_L = \frac{1}{n} \sum_{i}^{n} s_i \tag{3}$$

   Where s_i is calculated as:

$$s_i = \begin{cases} 0 & \text{if } r_i \notin ranking \\ \frac{l - r_i + 1}{l} & \text{else} \end{cases} \tag{4}$$

2. **Reciprocal Score**

   The reciprocal score S_R of a sequence if tokens T with length n can be calculated using the following formulae:

$$S_R = \frac{1}{n} \sum_{i}^{n} s_i \tag{5}$$

   Where s_i is calculated as:

$$s_i = \begin{cases} 0 & \text{if } r_i \notin ranking \\ \frac{1}{r_i} & \text{else} \end{cases} \tag{6}$$

3. **Non Linear Score**

   The non linear score S_{NL} of a sequence if tokens T with length n can be calculated using a hyperparameter α which determines how strong a lower ranking is punished. A higher value for α means that tokens that are ranked lower will have relatively lower scores.

$$S_{NL} = \frac{1}{n} \sum_{i}^{n} s_i \tag{7}$$

   Where s_i is calculated as:

$$s_i = \begin{cases} 0 & \text{if } r_i \notin ranking \\ e^{-\alpha(r_i - 1)} & \text{else} \end{cases} \tag{8}$$

4. **Perplexity Approximation**

   To calculate the perplexity for a sequence of tokens T, the probability of seeing each token is needed, take P to be the sequence of corresponding

probabilities $[p_1, p_2, p_3, ..., p_l]$. Since the API only allows the top 20 or top 5 (for the older legacy models) probabilities to be requested, the PPL will not be entirely representative when a token is not present in the ranking. Therefore, this is only an approximation of the true perplexity that serves as a supplementary comparison between models and lengths. The approximate perplexity $\widetilde{PPL}$ can be calculated using a score penalty when a token is not present, as seen below.

$$\widetilde{PPL} = e^{-\frac{1}{n}\sum_i^n \log p(t_i|t_{t<i})} \tag{9}$$

Where $\log p(t_i|t_{t<i})$ is the log-probability of seeing token t_i given the sequence of preceding tokens $t_{t<i}$ and can be calculated as such:

$$\log p(t_i|t_{t<i}) = \begin{cases} \log p_l - 3 & \text{if } r_i \notin ranking \\ \log p_i & \text{else} \end{cases} \tag{10}$$

The valuation when the token is not in the ranking, i.e. $r_i \notin ranking$, is calculated as the lowest probability $p_l \in P$ minus a penalty of 3. This penalty was chosen after running some tests from which it became apparent that there were large differences in the value of the last log probability $\log p_l$ that could be requested. So, in this manner, the penalty is not fixed but varying relative to the probabilities of the other, more likely, tokens. This aims to capture the fact that some tokens are generally more or less probable than others. The penalty value of 3 remains arbitrary however, and receiving all logits removes the need for any such approximations.

3.1.2 Algorithmic Implementation

In the analysis the models are not just being compared to each other, but they are also being evaluated at different context lengths. Factoring in the expenses of completion requests at large context lengths, this research will only test models up to and including 16384 tokens. Running cost analysis led us to set the number of comparisons n_c to 30. With this $n_c = 30$, we achieve a maximum price of about \$12 per text with the token lengths [32, 64, 128, 256, 512, 1024, 2048, 4096]. At each length a random starting position is chosen and a sliding window model is used. This means that the algorithm will run an evaluation with the current amount of tokens as input, and then slide the window up such that that amount is kept the same for the $n_c = 30$ evaluations.

When calling the API for a text completion, there are gains to be made in efficiency. Namely, when requesting a completion for a model it makes sense, especially at long-contexts, to request more than just a single token completion. Despite the output tokens being three times more expensive than the input tokens, we are talking about requesting thousands of input tokens and relatively few output tokens. In the event that the model gets the first token correct, and also chooses this as the token that it will use in the final completion, that previously requested completion can be referred to for the calculation of the next score. This means that the same request can be reused every time the model predicts the correct completion token. As soon as a model does not choose the correct token, that score will be calculated and a new completion will be requested.

This method allows for two possibilities: an exhaustive and non-exhaustive run. In the exhaustive case, the algorithm will take the efficiency gains it can get but continue until it has made $n_c = 30$ requests to the API. Therefore, the residual comparisons might exceed n_c depending on how well the model predicted and decided on tokens. The non-exhaustive variant will stop when it has made a total of $n_c = 30$ comparisons. The problem with the exhaustive variant is twofold; firstly the models are likely to return different amounts of total comparisons leading to a skewered total score. Secondly, it reincorporates the temperature as a variable due to the fact that the models indirectly rely on the temperature when making their final token choice. The final token choices in turn determine whether the requested completion can be reused in the case of it being correct.

In the code this parameter, 'max_tokens', determines how many tokens, and corresponding distributions, to request each time. At the amount of tests that will be done here, i.e. 30, it makes sense for these to be equal as the largest costs are incurred from loading the input context. All potentially correct tokens are worth their small cost by obviating the need to reload the entire input context.

3.1.3 Intermediate Data Saving

During the final run the SaveData variable was set to True meaning that relevant data was saved in excel sheets, for the purpose of potential post hoc analysis. The data saved were the reference tokens, the generated tokens, the token rankings, the log probability rankings and of course all of the corresponding scores. These data were saved per text, context length and model. The scores were also automatically averaged per context length and saved into excel files. In the appendix these average scores per context length can be found. The entire set of intermediate data can be found in the

GitHub linked in the abstract.

3.2 Models

The four main models used in the analysis are OpenAI's GPT-4-o, GPT-4-turbo, GPT-4 and GPT-3.5-turbo. These models have context windows of 128k, 128k, 8192 and 16385 tokens respectively. Other models from Gemini and Anthropic were not available in our region or did now allow for the logit distributions to be requested. Open source models would allow for a more complete insight into the distributions but running these models is extremely compute intensive, especially at such large context inputs. Therefore it was decided to go with the GPT family of models through the OpenAI API. These instruction-tuned models require a system prompt that usually tells them they are to be useful assistants, in this case the system prompt will be set to: "Complete the following text:". The user prompt will then be the text that the model must complete.

OpenAI also provides access to the base models for GPT-3 that have not been instruction tuned. These older models, babbage-002 and davinci-002, follow the legacy Completions API which means that there is no need for a system prompt as the model is simply completing the text it is given. Both these models support a context-window of 16384 tokens. An additional legacy Completions API model is GPT-3.5-turbo-instruct, which only supports up to 4096 tokens as input. Since these models are relatively cheap and not instruction tuned, they might provide insights into the possible effects of this tuning on text completion performance.

The Rank-Based Perplexity requires a model's token distribution and not just the token it decides upon. OpenAI's API allows for up to 20 log-probabilities of tokens to be requested for the newer Chat Completion models, and 5 log-probabilities for the older Completion models. When testing with different models, it is important to consider that different model tokenizers can lead to different starting token positions. It was therefore necessary to align the starting tokens of each model at each context length to ensure starting equality.

3.3 Data

When calculating the (rank-based) perplexity of models it is necessary to have a reference text to calculate the scores for. Importantly this data must be data that no model has been trained on, as this would bias them towards better scores. The most recently trained model, GPT-4-o, has training data of up to October 2023. The data used consisted of two publicly available

stories written by humans. The first text is 'Lessons From a Mass Shooter's Mother' by Mark Follman, published on Longreads. It was taken from the library of short stories. The other text is a short story called 'An Adventure in Futurity' written by Clark Ashton Smith, in the genres science-fiction, adventure and mystery.

These stories were selected for their length, as they both exceed the 16k tokens that the original testing would extend to, but also for their not being present in the training data. Both texts were filtered for any characters that the tokenizer did not recognize. These were not essential parts of the stories. Images were also removed from the second text. Multiple texts were used to increase diversity and reduce potential biases. Another option for obtaining long reference texts is to generate them using a different LLM. This would have to be from a distinct model family, and the prompt would need to induce the model to generate a novel piece of text.

The data sent to the OpenAI API will not be used to train or improve their models unless a user specifically opts in. In this case it is advantageous that, while testing the data and algorithms, the data was not used to train their models. No opt in was made to remove this potential bias from the models being trained on the data during pre-testing.

4 Results

The average scores for the runs at different lengths with $N_c = 30$ can be seen below in **Table 1**. The average score is the mean of all scores, i.e. the linear, reciprocal and non-linear with $\alpha = \{0.1, 0.3\}$. For each context length test at $N_c = 30$, the scores were generated for both texts. **Table 2** shows the approximated perplexity $\widetilde{PPL}$ scores. The individual scores averaged over the lengths can be found in the appendices in section **7** for text 1 and **8** for text 2.

Table 1. *Average scores of GPT models (t = turbo) at different lengths for both texts*

Text 1	**Length**	**3.5-t**	**4**	**4-t**	**4o**
	32	0.33	0.43	**0.50**	0.46
	64	0.51	**0.61**	**0.61**	0.52
	128	0.40	**0.49**	0.47	0.47
	256	0.51	**0.59**	0.57	0.55
	512	0.53	**0.72**	0.58	0.58
	1024	0.51	**0.70**	0.63	0.53
	2048	0.31	**0.61**	0.57	0.55
	4096	0.28	**0.52**	0.48	0.45
Text 2					
	32	0.37	**0.46**	0.45	0.45
	64	0.48	0.49	**0.52**	0.45
	128	0.37	**0.45**	0.43	**0.45**
	256	0.35	0.50	**0.53**	0.41
	512	0.52	**0.67**	0.62	0.65
	1024	0.54	**0.76**	0.70	0.70
	2048	0.50	0.57	**0.61**	0.54
	4096	0.52	**0.66**	0.65	**0.66**

From both the average scores and the approximated perplexity, GPT-3.5-turbo is seen to perform the worst. The best model according to the RBP is GPT-4 with 4-turbo coming in second. The perplexity estimate does not consistently align with the best scores. A linear regression analysis showed no significant regression fits, but showed interesting results for the average scores. Each model had a negative slope for text 1 and a positive slope for text 2, the full resuls of which can be found in the Appendix C in section 9. A negative slope indicates decreasing scores with context length and thus worse performance, at least according to the RBP. Both slopes were small as they were fit to the doubling length values. The slopes of both texts combined

Table 2. *Approximated PPL scores for GPT models (t = turbo) at different lengths for both texts*

Text 1	Length	3.5-t	4	4-t	4o
	32	696.57	606.08	278.40	**194.43**
	64	106.17	78.31	**61.28**	155.90
	128	179.40	**133.72**	318.39	138.41
	256	112.75	121.70	106.08	**80.68**
	512	77.59	**9.38**	36.64	30.15
	1024	66.28	**12.49**	33.79	67.16
	2048	407.93	**33.86**	78.79	47.24
	4096	341.37	242.64	194.29	**107.70**
Text 2					
	32	273.44	693.65	406.06	**219.54**
	64	**88.62**	288.40	290.51	127.05
	128	258.43	479.54	611.22	**248.72**
	256	379.70	232.90	**159.94**	214.31
	512	90.66	29.80	62.88	**24.02**
	1024	109.12	**14.39**	24.60	21.22
	2048	82.55	70.73	**37.14**	57.39
	4096	60.60	**16.22**	17.92	20.33

fit to the averages scores for each model were GPT-3.5-turbo: -7.75e-006 ± 1.77e-005 (with p = 0.67), GPT-4: 1.92e-005 ± 1.97e-005 (with p = 0.35), GPT-4-turbo: 1.50e-005 ± 1.47e-005 (with p = 0.32) and GPT-4-o: 1.90e-005 ± 1.61e-005 (with p = 0.26).

When fitting a linear regression model on the PPL approximate per separate text, the fits were once again all not significant. Each model, except 3.5-turbo in text 1, had a negative slope, with text 2 having steeper (more negative) slopes and lower probabilities all-round. The slopes and probabilities can be found in section 9. Fitting linear regression models to the PPL approximation was also done using the average of both text's PPL scores. The slopes were all not deemed significant either. The slopes were GPT-3.5-turbo: -0.0090 ± 0.0343 (with p = 0.80), GPT-4: -0.0558 ± 0.0406 (with p = 0.19), GPT-4-turbo: -0.0471 ± 0.0307 (with p = 0.15) and GPT-4-o: -0.0276 ± 0.0134 (with p = 0.06).

To determine the accuracy of these results at a relatively small sample size of 30, the scores were repeatedly generated for a fixed context length of 128 tokens a total of 5 times for each text. This allows for a comparison of scores at different locations in the text, with a fixed context length. The results can be seen in **Table 3** and **Table 4** below.

Table 3. *Average scores repeated 5 times at length 128*

Text 1	**gpt-3.5-turbo**	**gpt-4**	**gpt-4-turbo**	**gpt-4o**
	0.51	0.54	0.55	0.47
	0.57	0.56	0.54	0.51
	0.46	0.56	0.55	0.50
	0.45	0.46	0.41	0.41
	0.33	0.52	0.51	0.47
Text 2	0.49	0.56	0.53	0.50
	0.56	0.58	0.61	0.54
	0.47	0.53	0.53	0.51
	0.43	0.56	0.54	0.56
	0.51	0.56	0.55	0.52

Table 4. *Approximated PPL repeated 5 times at length 128*

Text 1	**gpt-3.5-turbo**	**gpt-4**	**gpt-4-turbo**	**gpt-4o**
	107.99	64.85	68.97	218.77
	97.37	172.01	153.76	107.55
	165.30	94.43	154.43	118.06
	233.05	276.70	394.40	196.17
	550.23	211.12	232.29	98.39
Text 2	174.61	49.00	73.39	133.70
	75.97	59.43	68.01	153.69
	219.05	161.52	129.81	57.52
	177.61	158.39	117.72	65.96
	75.27	68.14	85.14	61.83

Both repeated measurement scores at length 128 show a substantial degree of variance between the models. For the average scores the two texts have a combined average standard deviation of 0.049 and 93.00 for the $\widetilde{PPL}$. When taking the previous original measures at 128 into account we get an increased average SD of 0.053 and 119.36, for the RBP and and for the $\widetilde{PPL}$, respectively. Given the average per model and their SD (taking each measurement at 128 into account) over both texts, we can determine which values were likely to be outliers in the original measurement. For both the average score and for the $\widetilde{PPL}$, 3 out of the 8 values measured were outliers. This means that from **Table 1** and **Table 2**, at length 128 there were a total of 6/16 outliers.

5 Conclusion

It is not a surprise that GPT-3.5-turbo performed the worst on all scores; it is the oldest model in the series. This observation does however signal the RBP's efficacy in capturing this qualitative increase in the model's language abilities. The RBP scores GPT-4 as the best performing model with GPT-4-turbo coming in second. Although each linear regression slope for the combined texts was not considered significant with $p < 0.05$, all of the slopes, except that of GPT-3.5-turbo, were positive in relation to the context length. This means that, alteast to a small degree, scores improved with context length in the aforementioned models. Here, GPT-4-o showed the best increase with context length when taking p into account.

The perplexity approximate does not initially show clear indications of a model performing better than another, except for GPT-3.5-turbo being the worst. When taking the average PPL of each model, the order and corresponding scores are: GPT-4-o, 4-turbo, 4 and 3.5-turbo with 109.64, 169.87, 191.49 and 208.20 respectively. The linear regression slopes show a negative correlation between score and increasing context length; a negative correlation meaning a lower PPL and therefore better scores. Although these values were not significant according to $p < 0.05$, the slope of GPT-4-o of -0.0276 came close with $p = 0.06$. The case can be made that GPT-4-o showed the best decrease in PPL in relation to increasing context lengths.

When looking at the history of OpenAI models these results might be able to be put into context. After 3.5-turbo came GPT-4, a model that accepted both images and text as input. GPT-4 was much more compute expensive but showed better scores on every evaluation. Thereafter came GPT-4-turbo, a faster and cheaper model that also increased the context length from 8192 to 128k tokens. The most recent model, GPT-4-o, has been upgraded to process both video and audio in addition to images and text. It could be posed that due to an increase in compute efficiency and further multi-modality, the models exhibit worse performance, where the final ranking is GPT-4, 4-turbo, 4-o and 3.5-turbo.

However, GPT-4-o currently stands in first place on the LYMSYS leader boards, with 4-turbo following in second place (4th on the leaderboard) and GPT-4 and 3.5-turbo are seen increasingly further down the leader boards. This leader board is a subjective but effective evaluation of a model's ability to answer user prompts. These tasks are however not long-context evaluations of the models, but serve as a qualitative reference which the PPL reflected, and the RBP reflected only partially.

When comparing the two texts and their individual slopes, an interest-

ing pattern can be observed. The RBP sees all negative slopes for text 1 and all positive slopes for text 2. A negative slope means that the models performed worse as the context lengths increased, and vice-versa. When looking at the slopes of the PPL, we can see that for text 1, all slopes were negative except for GPT-3.5-turbo. Text 2 similarly has all negative slopes, although these are higher and closer to being significant. A higher negative slope in the case of PPL means that a model is performing better at increasing context lengths. This seems to be indicative of differences between the properties of the two texts. When looking at the averages per text of both the RBP and the PPL there is not a large difference: text 1 has 0.52 and 161.11 and text 2 has 0.53 and 178.49. This can lead us to pose that while the texts were both equally difficult to predict, text 2 allowed for better usage of increasing context windows. Whether this is due to one of the texts being anomalous or it being a statistical insignificance, is something that more research into the topic can illuminate.

This research aims to probe into a method for evaluation of long-contexts after it was shown that the perplexity is not always a representative measure of performance, which could be due to the temperature playing a direct role in its calculation. The RBP is not affected by this potentially confounding variable. The temperature matters insofar as one wants a model to be creative, but we can raise the question whether this is something that should be incorporated into the evaluation of LLMs.

Furthermore, when a model is given more context as input, this is likely to be reflected by a change in the probabilities it assigns to tokens. Whether this results in more or less confident predictions is dependent on the text, as prior results show, and likely a multitude of other variables as well. What is important here is that the RBP will not be influenced by this change in confidence, as the ranking of tokens stays the same. Further research with more models, texts, context lengths, samples and scoring combinations (Linear, Reciprocal, ...) is needed to corroborate the effectiveness of a Rank Based Perplexity scoring algorithm.

Another area where perplexity evaluations might suffer is the fact that different tokenizers are used by different models. Take model A with tokenizer A that splits a word into its prefix and suffix while model and tokenizer B turns the entire word into a single token. The suffix is statistically easier to predict than the entire word at once as there are always multiple word combinations that can follow from the suffix (if not it wouldn't be split in the first place). Therefore, model A has a higher degree of confidence in predicting the suffix, while model B has more options to choose from and thus a less confident probability distribution. Model A subsequently faces another

high confidence prediction as the token options that follow a particular prefix are limited. The PPL will be affected by this difference in tokenization, and at long-contexts this might accumulate into evaluational disparities. The RBP is not influenced by the difference in token confidence that results from differences in tokenizers.

In conclusion, both the RBP and $\widetilde{PPL}$ are seen to have GPT-4-turbo in second place with GPT-3.5-turbo in last place. The RBP scores GPT-4 as the best and the perplexity approximate scores GPT-4-o as the best performing model. Most models are seen to improve in both perplexity and RBP as the context size increases, although the difference is not statistically significant. Furthermore, GPT-4-o seems to make the most use of increasing context lengths and the PPL approximate coincides with the ranking of the LYMSYS leaderboard. Despite this, the limited number of samples and context lengths mean that the RBP did not have a chance to strongly prove or disprove itself as effective. It conferred the expected trivial results which demonstrate a foundational scoring ability.

The analysis into the variation of scores at the same context length brings the validity of the results into question. The high SD that was present in the sample is only likely to be larger in the population. Therefore, the constrained parameters like N_c and the maximum context length, make this particular set of results less trustworthy and it is important to recognize these limitations. Nevertheless, the RBP proved to be an indicator of model performance. More research with larger datasets, samples and more models is needed. This, in combination with unrestricted logits being returned and applying different scoring algorithm combinations, could yield more accurate scores.

6 Future Work

6.1 Scoring

The scoring algorithms presented in this paper each have unique characteristics. There are surely more ways to score the tokens based on their ranking, and perhaps there is a single method that works well to capture long-context performance. The methods might be tested separately or in combination to determine their strengths. There might be specific use cases to which a scoring combination can be tuned.

6.2 Token Analysis

The comparison data (tokens, rankings, probabilities and scores) have been saved and can be found in the Appendix or Github together with the code. From this and future data, there are a few options for deeper analysis. One of these is a prefix analysis. When a tokenizer splits words into tokens, it will occasionally take the prefix of a word as a token, followed by the rest of the word. Due to different models using different tokenizers, discrepancies can occur. An analysis of the token rankings that incorporates prefix tokenization between models might give a more accurate score.

This leads us into another possibility of post-hoc token analysis. Some tokens are likely to be easier to predict than other tokens. The question then is if this degree of likelihood is influenced by different context lengths. Further research into differences between tokenizers and their effects on evaluation metrics like the perplexity can create better benchmarks, from which better models will follow.

6.3 Further Evaluations

An additional possibility for a new long-context evaluation will be briefly described here. In an effort to capture long-context reasoning performance, previously discussed benchmarks have given entire books as input and asked questions about these books that require the retrieval of- and reasoning about pieces of information spread out among the text. This extends the straightforward retrieval task to requiring actual reasoning across long range dependencies.

In this spirit, consider a formal system wherein axioms allow for the generation and derivation of theorems through rules of inference. When taking the rules and starting axioms of such a system, we can give them as

input to the LLM to see how it performs on a series of tests. The idea is to then make this evaluation of modular length by inserting text as noise in between the rules. The rules can in this case be spread out to any length and an evaluation without noise can provide a baseline for a model's ability to reason with the rules in a local setting. Such an evaluation could capture a model's ability to reason over rules nested in a long-context input.

References

Agarwal, R., Singh, A., Zhang, L. M., Bohnet, B., Rosias, L., Chan, S., ... Larochelle, H. (2024). *Many-shot in-context learning.* Retrieved from https://arxiv.org/abs/2404.11018

AI, M. (2023). *Kimi chat.* Retrieved from https://kimi.moonshot.cn/

An, C., Gong, S., Zhong, M., Zhao, X., Li, M., Zhang, J., ... Qiu, X. (2023). *L-eval: Instituting standardized evaluation for long context language models.*

arkadyark cohere. (2024). *Needle in a haystack - pressure testing llms.* Retrieved from https://github.com/gkamradt/LLMTest_NeedleInAHaystack/blob/main/README.md

Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., ... Li, J. (2023). *Longbench: A bilingual, multitask benchmark for long context understanding.*

The claude 3 model family: Opus, sonnet, haiku. (n.d.). Retrieved from https://api.semanticscholar.org/CorpusID:268232499

Ding, Y., Zhang, L. L., Zhang, C., Xu, Y., Shang, N., Xu, J., ... Yang, M. (2024). *Longrope: Extending llm context window beyond 2 million tokens.*

Dong, Q., Li, L., Dai, D., Zheng, C., Ma, J., Li, R., ... Sui, Z. (2024). *A survey on in-context learning.* Retrieved from https://arxiv.org/abs/2301.00234

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., ... Wang, H. (2024). *Retrieval-augmented generation for large language models: A survey.* Retrieved from https://arxiv.org/abs/2312.10997

Hsieh, C.-P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., ... Ginsburg, B. (2024). *Ruler: What's the real context size of your long-context language models?* Retrieved from https://arxiv.org/abs/2404.06654

Hu, Y., Huang, Q., Tao, M., Zhang, C., & Feng, Y. (2024). *Can perplexity reflect large language model's ability in long text understanding?*

Keles, F. D., Wijewardena, P. M., & Hegde, C. (2022). *On the computational complexity of self-attention.*

Li, T., Zhang, G., Do, Q. D., Yue, X., & Chen, W. (2024). *Long-context llms struggle with long in-context learning.*

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2023). *Lost in the middle: How language models use long contexts.*

Liu, Y., He, H., Han, T., Zhang, X., Liu, M., Tian, J., ... Ge, B. (2024). *Understanding llms: A comprehensive overview from training to inference.*

Panickssery, A., Bowman, S. R., & Feng, S. (2024). *Llm evaluators recognize and favor their own generations.*

Peng, B., Quesnelle, J., Fan, H., & Shippole, E. (2023). *Yarn: Efficient context window extension of large language models.*

Shaham, U., Ivgi, M., Efrat, A., Berant, J., & Levy, O. (2023). *Zero-scrolls: A zero-shot benchmark for long text understanding.* Retrieved from `https://arxiv.org/abs/2305.14196`

Sun, T., Shao, Y., Qian, H., Huang, X., & Qiu, X. (2022). *Black-box tuning for language-model-as-a-service.* Retrieved from `https://arxiv.org/abs/2201.03514`

Team, G., Reid, M., Savinov, N., Teplyashin, D., Dmitry, Lepikhin, ... Vinyals, O. (2024). *Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context.*

Wang, C., Duan, H., Zhang, S., Lin, D., & Chen, K. (2024). *Ada-leval: Evaluating long-context llms with length-adaptable benchmarks.*

Yorsh, U., & Kovalenko, A. (2022). Linear self-attention approximation via trainable feedforward kernel. In *Artificial neural networks and machine learning – icann 2022* (p. 807–810). Springer Nature Switzerland. Retrieved from `http://dx.doi.org/10.1007/978-3-031-15934-3_67` doi: 10.1007/978-3-031-15934-3_67

Zaheer, M., Guruganesh, G., Dubey, A., Ainslie, J., Alberti, C., Ontanon, S., ... Ahmed, A. (2021). *Big bird: Transformers for longer sequences.* Retrieved from `https://arxiv.org/abs/2007.14062`

Zhang, B., Liu, Z., Cherry, C., & Firat, O. (2024). *When scaling meets llm finetuning: The effect of data, model and finetuning method.*

Zhang, X., Chen, Y., Hu, S., Xu, Z., Chen, J., Hao, M. K., ... Sun, M. (2024). ∞*bench: Extending long context evaluation beyond 100k tokens.*

7 Appendix A

All the scores averaged over each length for text 1.

Length	Metric	gpt-3.5-turbo	gpt-4	gpt-4-turbo	gpt-4o
32	PPL	696.57	606.08	278.40	**194.43**
	LIN	0.42	0.57	**0.63**	0.56
	RP	0.25	0.30	**0.36**	0.35
	A01	0.38	0.51	**0.58**	0.52
	A03	0.27	0.36	**0.44**	0.39
	AVG	0.33	0.43	**0.50**	0.46
64	PPL	106.17	78.31	**61.28**	155.90
	LIN	0.59	0.65	**0.66**	0.58
	RP	0.43	**0.56**	0.54	0.47
	A01	0.56	0.64	**0.65**	0.56
	A03	0.47	0.59	**0.60**	0.49
	AVG	0.51	**0.61**	**0.61**	0.52
128	PPL	179.40	**133.72**	318.39	138.41
	LIN	0.49	**0.59**	0.58	0.57
	RP	0.31	**0.37**	0.36	0.36
	A01	0.45	**0.56**	0.54	0.53
	A03	0.33	**0.45**	0.41	0.41
	AVG	0.40	**0.49**	0.47	0.47
256	PPL	112.75	121.70	106.08	**80.68**
	LIN	0.61	0.69	**0.71**	0.65
	RP	0.39	**0.49**	0.43	0.44
	A01	0.57	**0.66**	**0.66**	0.61
	A03	0.45	**0.54**	0.49	0.50
	AVG	0.51	**0.59**	0.57	0.55
512	PPL	77.59	**9.38**	36.64	30.15
	LIN	0.61	**0.83**	0.70	0.70
	RP	0.45	**0.61**	0.47	0.45
	A01	0.58	**0.78**	0.65	0.66
	A03	0.49	**0.65**	0.52	0.52
	AVG	0.53	**0.72**	0.58	0.58
1024	PPL	66.28	**12.49**	33.79	67.16
	LIN	0.60	**0.77**	0.71	0.61
	RP	66.28	**12.49**	33.79	67.16
	A01	0.57	**0.74**	0.68	0.57
	A03	0.47	**0.66**	0.59	0.49
	AVG	0.51	**0.70**	0.63	0.53
2048	PPL	407.93	**33.86**	78.79	47.24
	LIN	0.41	**0.68**	0.63	0.64
	RP	0.21	**0.52**	0.50	0.45
	A01	0.37	**0.66**	0.61	0.61
	A03	0.24	**0.58**	0.53	0.50
	AVG	0.31	**0.61**	0.57	0.55
4096	PPL	341.37	242.64	194.29	**107.70**
	LIN	0.35	**0.60**	0.58	0.52
	RP	0.21	**0.44**	0.38	0.38
	A01	0.32	**0.57**	0.54	0.49
	A03	0.22	**0.48**	0.43	0.41
	AVG	0.28	**0.52**	0.48	0.45

8 Appendix B

All the scores averages over each length for text 2.

Length	Metric	gpt-3.5-turbo	gpt-4	gpt-4-turbo	gpt-4o
32	PPL	273.44	693.65	406.06	**219.54**
	LIN	0.46	0.56	**0.58**	0.57
	RP	0.27	**0.33**	0.31	0.32
	A01	0.43	**0.53**	**0.53**	0.52
	A03	0.32	**0.41**	0.37	0.38
	AVG	0.37	**0.46**	0.45	0.45
64	PPL	**88.62**	288.40	290.51	127.05
	LIN	0.55	0.59	**0.60**	0.55
	RP	0.40	0.39	**0.44**	0.36
	A01	0.52	0.56	**0.57**	0.51
	A03	0.43	0.43	**0.49**	0.39
	AVG	0.48	0.49	**0.52**	0.45
128	PPL	258.43	479.54	611.22	**248.72**
	LIN	0.44	0.52	**0.53**	**0.53**
	RP	0.29	**0.38**	0.33	0.37
	A01	0.42	0.49	0.49	**0.50**
	A03	0.32	**0.41**	0.37	0.40
	AVG	0.37	**0.45**	0.43	**0.45**
256	PPL	379.70	232.90	**159.94**	214.31
	LIN	0.44	0.60	**0.65**	0.50
	RP	0.26	**0.39**	**0.39**	0.31
	A01	0.40	0.57	**0.60**	0.47
	A03	0.29	**0.46**	**0.46**	0.37
	AVG	0.35	0.50	**0.53**	0.41
512	PPL	90.66	29.80	62.88	**24.02**
	LIN	0.62	**0.76**	0.74	0.74
	RP	0.42	0.56	0.52	**0.57**
	A01	0.58	**0.73**	0.69	0.71
	A03	0.47	**0.62**	0.55	0.61
	AVG	0.52	**0.67**	0.62	0.65
1024	PPL	109.12	**14.39**	24.60	21.22
	LIN	0.59	**0.84**	0.78	0.76
	RP	0.50	**0.68**	0.62	0.64
	A01	0.57	**0.81**	0.74	0.73
	A03	0.51	**0.72**	0.66	0.66
	AVG	0.54	**0.76**	0.70	0.70
2048	PPL	82.55	70.73	**37.14**	57.39
	LIN	0.57	0.63	**0.69**	0.60
	RP	0.42	0.51	**0.52**	0.48
	A01	0.55	0.61	**0.66**	0.57
	A03	0.47	0.54	**0.58**	0.51
	AVG	0.50	0.57	**0.61**	0.54
4096	PPL	60.60	**16.22**	17.92	20.33
	LIN	0.60	0.73	**0.74**	0.72
	RP	0.45	**0.61**	0.56	0.60
	A01	0.57	**0.70**	**0.70**	0.69
	A03	0.48	0.62	0.61	**0.63**
	AVG	0.52	**0.66**	0.65	**0.66**

9 Appendix C

An overview of the linear regression slopes per text for the RBP and PPL. The first number is the slope and its standard deviation, followed by p.

9.1 RBP

Text	3.5t	4	4t	4o
1	-4.608e-5 ± 2.355e-5 0.10	-2.292e-6 ± 2.880e-5 0.94	-1.150e-5 ± 1.674e-5 0.52	-9.537e-6 ± 1.330e-5 0.50
2	3.059e-5 ± 1.907e-5 0.16	4.076e-5 ± 2.854e-5 0.20	4.159e-5 ± 2.203e-5 0.11	4.759e-5 ± 2.701e-5 0.13

9.2 PPL

Text	3.5t	4	4t	4o
1	3.5 0.02808 ± 0.06254 0.67	-0.007794 ± 0.05701 0.90	-0.002744 ± 0.03210 0.93	-0.01149 ± 0.01570 0.49
2	-0.04626 ± 0.02854 0.16	-0.1039 ± 0.05832 0.13	-0.09147 ± 0.05028 0.12	-0.04375 ± 0.02213 0.10