Uncertainty Quantification for Code Generation with Large Language Models

AUTHOR
Soren Dunn

Github Link: https://github.com/sorendunn/LLM-Calibrated-Uncertainty-for-Code

Introduction:

Large language models are ever-more-frequently being used by individuals in applications ranging from writing essays to debugging code. In fact, ChatGPT is reported to be the fastest growing app in the history of web applications (1). Amid all this hype however language models still exhibit glaring deficiencies. Notably, they still do not often admit ignorance of a topic and provide a convincing answer even when they are completely ignorant. These hallucinations not only greatly limit the usefulness of these models but also can lead to dissemination of false information and other negative societal effects. These concerns are even more pronounced due to the tendency of language models to go along with assumptions stated in their prompts (they are trying to perform next word prediction and a piece of text is unlikely to contradict its strongly-stated assumptions). Language models confidently stating false information is a failure of communcation between the individual who trained the model and the user. It also loses all information about the quality of the data and forces the end user to judge the response quality almost entirely in the dark. Failing to provide information about the model's uncertainty makes end users final decisions based off of the model worse informed since it is harder for them to judge when to trust the model's outputs.

To help combat this solution, I propose adding a measure of uncertaintly along with a lanugage model's response to a question. Ideally this would look something like the following:

User: "What is the capital of France?"

Language Model: "Paris" Confidence: 98%

User: "What is 93382 * 39383?"

Language Model: "367766330" Confidence: 3%

Having the model output an uncertainty metric along with its answer helps combat the problem of language models disseminating false information. If the model outputs a low confidence in its response, the human will simply disregard the answer. This approach is also particularly effective since it leaves the choice to the user to determine what confidence level they require for their particularly domain. For example, if the user is using the model in the medical domain then they might require a higher confidence threshold than if they are debugging code for a python visualization.

It is undoubtedly possible to come up with *some* measure of uncertainty for models answers but the most important component is coming up with a measure of uncertainty that does a good job of reflecting the real probability of the provided answer being correct. In statistics the term for outputted probabilites matching the underlying probabillity for an answer is called calibration and a model where its outputted probabilities match well with the underlying probabilities for its answers can be said to be well calibrated.

There have been several previous attempts at achieving calibration in large language models, but there is no clear way to extend these methods to perform well on free-form question answering on the current most powerful large language models such as ChatGPT and GPT-4. Here, I propose a novel (to the best of my knowledge) approach to outputting calibrated outputted model probabilities for large language models (explained later) My research question is to test if this novel approach to uncertainty quantification can acheive similar calibration on ChatGPT and GPT-4 as previous methods evaluated on smaller models.

There are several common methods for measuring calibration, but they are all fundamentally attempting to measure how different the model ouputted probabilities for a response are to the true underlying values for that response. One obvious problem with such comparisons is that we only know whether a response is actually correct or incorrect - we rarely have a simple way to know what the underlying probability was for a response to be correct or not. However, if the model predicts a 90% probability for 1000 different questions then if the model was well-calibrated we would strongly expect around 900 of those answers to be correct. Thus we can group answers by the probability that the model assigns of those answers being correct and check the model's calibration based on how frequently answers in those groups were correct vs. the probability the model assigned to those answers being correct. One additional problem arises with this approach: it is not uncommon that a model may assign most answers slightly different probabilities (for example assigning 17.7% to one answer, 18% to another, and 19.2% to a third). To combat this problem what we do in practice is compare the percent of model answers that were correct in buckets of probability (for example combining all model predictions between 15% and 20%).

We can check calibration visually by plotting the outputted probabilites for the model against the real proportion of those answers being correct. If a model is well-calibrated, its points plotted on such a plot would fall along the line y=x (examples of such plots and further descriptions of such can be seen in the results section). Another method for measuring calibration is the expected calibration error (ECE) which is a weighted average of the absolute difference between the proportion of answers in a group that are correct and the average probability of being correct that the model assigned to that group. The last method of calibration I used is the mean squared error between whether an answer was correct and the probability that was assigned to that answer being correct. It has the advantage of not requiring grouping of the outputted probabilities but is arguably less interpretable than the ECE so I report both metrics for all results.

Previous Work

There have been several papers on eliciting calibrated probabilites from large language models, each approaching the problem with a different method. The most natural of these methods is to look at the probabilities that models assign to tokens for True/False or multiple choice questions. This is the approach taken by Kaplan et al. [2] are able to achieve almost perfect calibration (ECE around 0.07) on True/False and multiple choice questions presented to models with up to 10^10 parameters (around an order of magnitude smaller than GPT-3). Although these results are quite promising they do not solve the problem of calibration for the free-form question answering setting. This same paper attempted to extend their method to acheive calibration with free-form responses by first having the model generate its response and then looking at the probability that it assigned to the token that the above answer was true. Unfortunately, using this approach lead to significantly poorer model calibration (with brier scores around 0.18). This same paper also identified a further issue: reinforcement learning with human feedback (RLHF) a large component of the process that made ChatGPT signficantly more useful to people than GPT-3 also degrades model calibration. This makes it particularly difficult to acheive model calibration on newer, RLHF trained models such as GPT-3.5 (ChatGPT) and GPT-4.

In addition to the above approach, Si et al. [3] also try sampling the answer to a question from the model 10 times and taking the frequency of the most outputted answer as its probability. Using these methods they mostly acheive ECE values around 15 to 25 on simple question-answering tasks where there is a fixed correct answer. I thought this approach was promising, but had to adapt it for the code-generation domain since there is no one, fixed set of tokens that is the right answer for this setting.

The last approach in the literature worth mentioning is that Hilton et al. [4] tried to fine-tune models to verbally output their probabilities of their answers being correct. While this approach was able to acheive good calibration on the exact benchmark the models were fine-tuned on, it tended to generalize poorly to other domains.

While not directly an approach for uncertainty quantification, the approach taken here was also inspired by Codet [5], which increases language model performance on coding benchmarks by having the system generate test cases without looking at the ground truth test cases (I term these "internal" test cases), generate a bunch of potential function, evaluate those functions on the proposed test cases, and finally choose the model answer which performed best when evaluated on its internal test cases. This approach implicitly assumes that the answers which evaluate correctly on the most number of test cases have the highest probability of being correct. Since this approach performed well, I tried to adapt it to attempt to quantify the uncertainty the model assigned to each of its answers. Specifically, I followed the Codet approach of generating several internal tests for a given question and then its subsequent answers on those proposed test cases. I took the percentage of test cases correctly passed as the model's measure of uncertainty. The benefits of this approach is that it works for free-form code generation with no simple answer (instead of only working for true/false questions like some previous approaches). It does not require fine-tuning on any specific dataset so it would be more likely to generalize than methods based on fine-tuning.

One last baseline that it is worth comparing the subsequent results to is the multiple-choice question-answer calibration acheived in the GPT-4 Technical Report [6]. There they acheive ECE of 0.007 before RLHF and ECE of 0.074 after RLHF (again showing how RLHF degrades model calibration). My results achieve lower calibration than this but as previously shown are on a much harder task for calibration: free-form code generation.

Methods

Though I had fixed a general method for calibration (evaluating on internal test cases), I still had several degrees of freedom to experiment with to acheive calibration including temperature adjustment, method for generating additional functions, and whether to make post-hoc adjustments to the determined probabilities. The most straightforward of these is model temperature: I experimented with different temperature settings for the model (think amount of randomness). In terms of the method for generating additional functions, I already went over the basic method I used (simply generating several different possible functions for a fixed set of prompt and test cases). However I also experimented with an additional prompting technique called Reflexion to see how it would effect model accuracy [7]. With this approach, after generating a function and evaluating it on test cases the error message for any test cases that weren't passed are fed back into the model and the model is asked to regenerate the function fixing the test cases that it failed to pass. I also experimented with increasing the number of iteration of the reflection for the Reflexion approach, but this did not end up improving results. Lastly, one could imagine that although the raw number of test cases passed by a function wasn't a good method of measuring accuracy there may be a transformation one could apply to those probabilities that would lead to calibrated probabilities (which I refer to as ad-hoc adjustment).

I also needed a dataset upon which to measure the performance of these various methods. For simplicity I chose HumanEval, one of the most widely used code generation benchmarks which also has the advantage of only containing around 150 questions so it is computationally feasible to evaluate on and I was able to find a previous code implementation I could adapt for this project [7]. On HumanEval, the model is given the start of a function in the following format

```
def digits(n):
"""Given a positive integer n, return the product of the odd digits."""
```

and is expected to correctly complete the function. Whether the function is written correctly is checked by evaluating the returned function on a series of test cases which are not shown to the model.

To avoid overfitting on the dataset I tried the various degrees of freedom I had on a random subset of 50 problems from the HumanEval dataset. I then chose the best-performing of my approaches on those problems and applied the same method at the remaining problems from the dataset. I also primarily experimented with GPT-3.5 and only used GPT-4 to evaluate the effectiveness of the final selected approach on the test set (due to computational costs). I still considered it worth testing the approach with both GPT-3.5 and GPT-4 to test whether the approach scaled with model size and performance. Note that the difference in performance of GPT-3.5 and GPT-4 on this dataset is actually quite substantial. Using a Codet-like approach the accuracy acheived for GPT-3.5 and GPT-4 on my runs were 72% and 92%, respectively. This substantial difference in performance between the two models made me consider it worthwhile to test my approach for uncertainty quantification on both of them.

Results

I found that a temperature around 1 tended to maximize the calibration of the models on the training dataset, with temperature above 2 being mostly useless to the language models starting to generate complete nonsense rather than properly-running functions. Post-hoc adjustments applied using calibration results from one subset of the data didn't generalize to other prompts in the data. Additionally, calibration using a Reflexion-type approach for generating functions was worse than the calibration for a Codet-like approach (calibration plots shown in the appendix). Therefore, the main final results simply generated the test functions separately using temperature 1. Using this approach I was able to acheive ECE under 10 for prompts in the experimental set, but only ECE around 17 on the testing set questions. Note that here I am reporting ECE out of 100 while the papers I mentioned early report ECE out of 1 so for these results to be comparable one should divide these reported ECE values by 100.

In the rest of this section I will display representative calibration plots showing ECE and Brier score (here equivalent to MSE) for various test runs. The red dashed line in these plots shows perfect calibration. Each of the blue bars represent a group of answers where the model predicted a similar probability for them to be true. The number above each bar is the number of points which contributed to that bar and the height of the bar is the percent of the questions within that bar that the model actually answered correctly. The x-axis shows the average percent of internal test (tests generated by the model itself) which were passed by the answers making up each bar. Additionally, each plot may show more answers than the total number of questions since 5 potential answers were generated for each question, each of which is shown here.

▶ Code

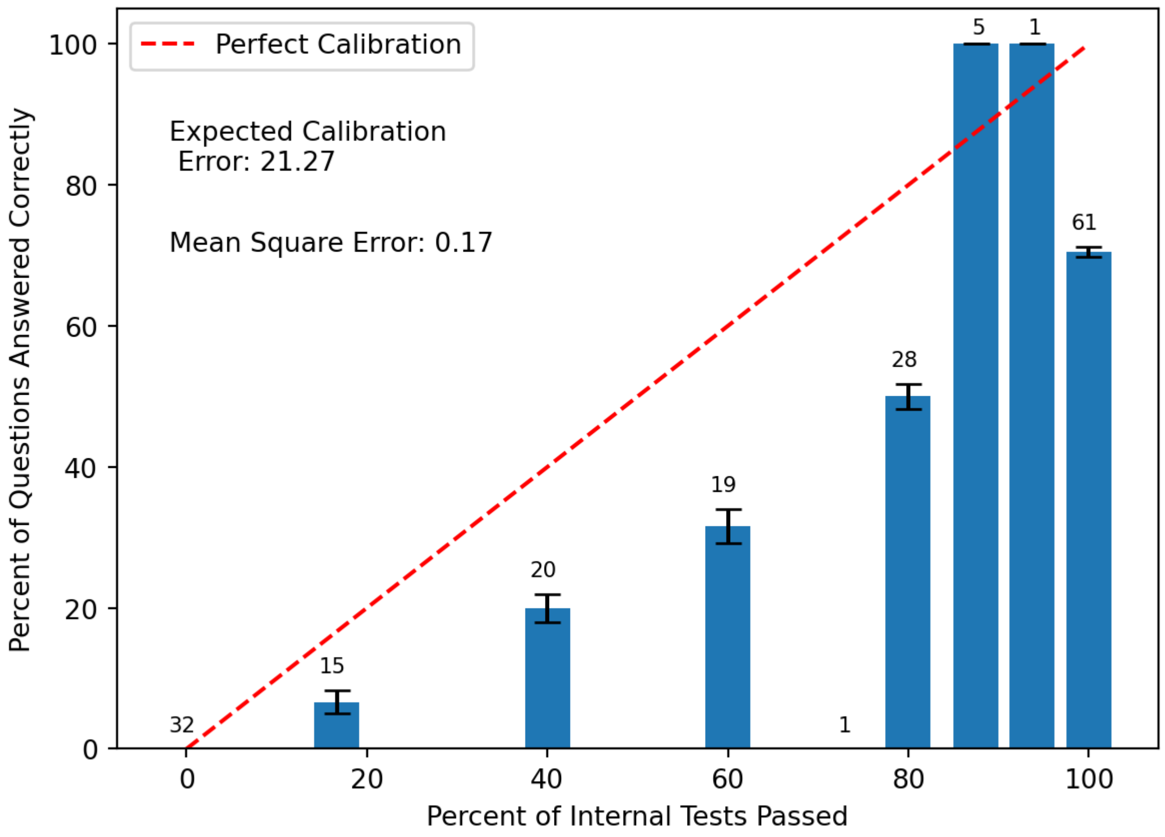

Figure 1: Calibration plot for GPT-3.5 on experimental cases using Reflexion

▶ Code

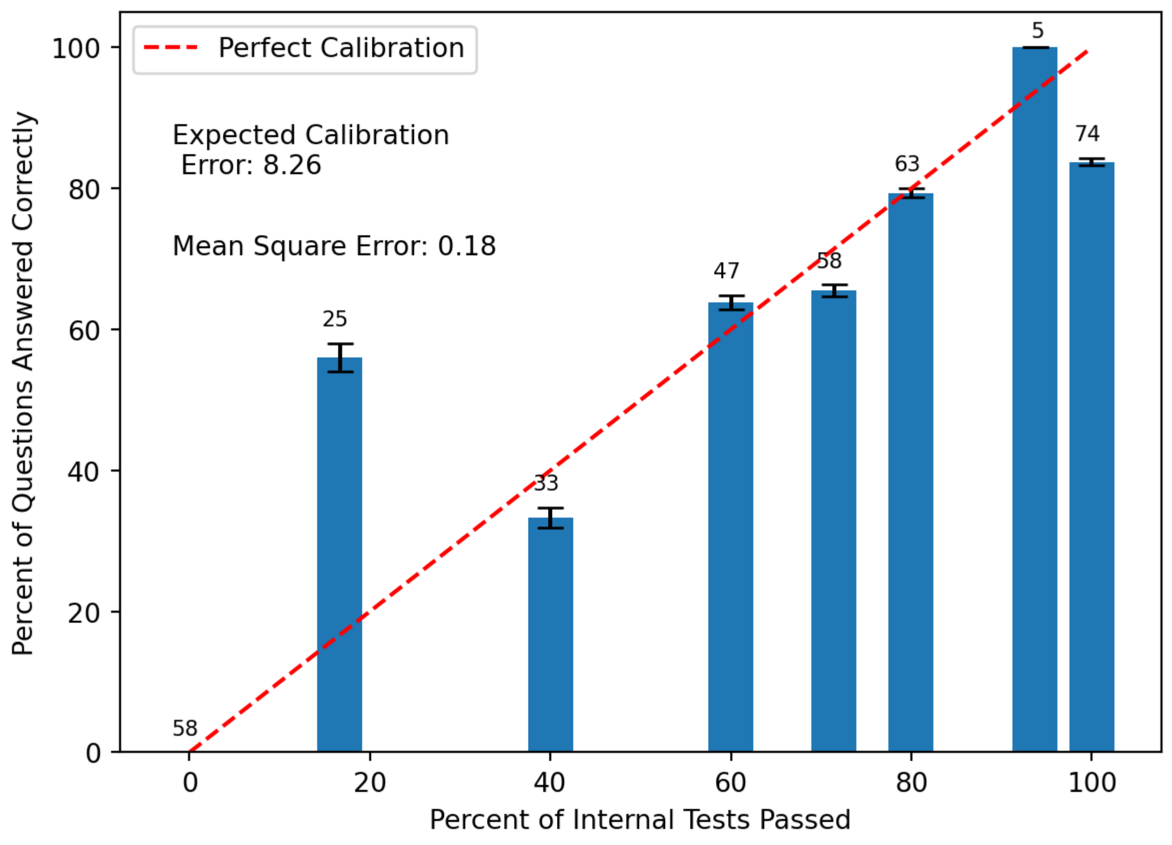

Figure 2: Calibration plot for GPT-3.5 on experimental cases using separate function generation

▶ Code

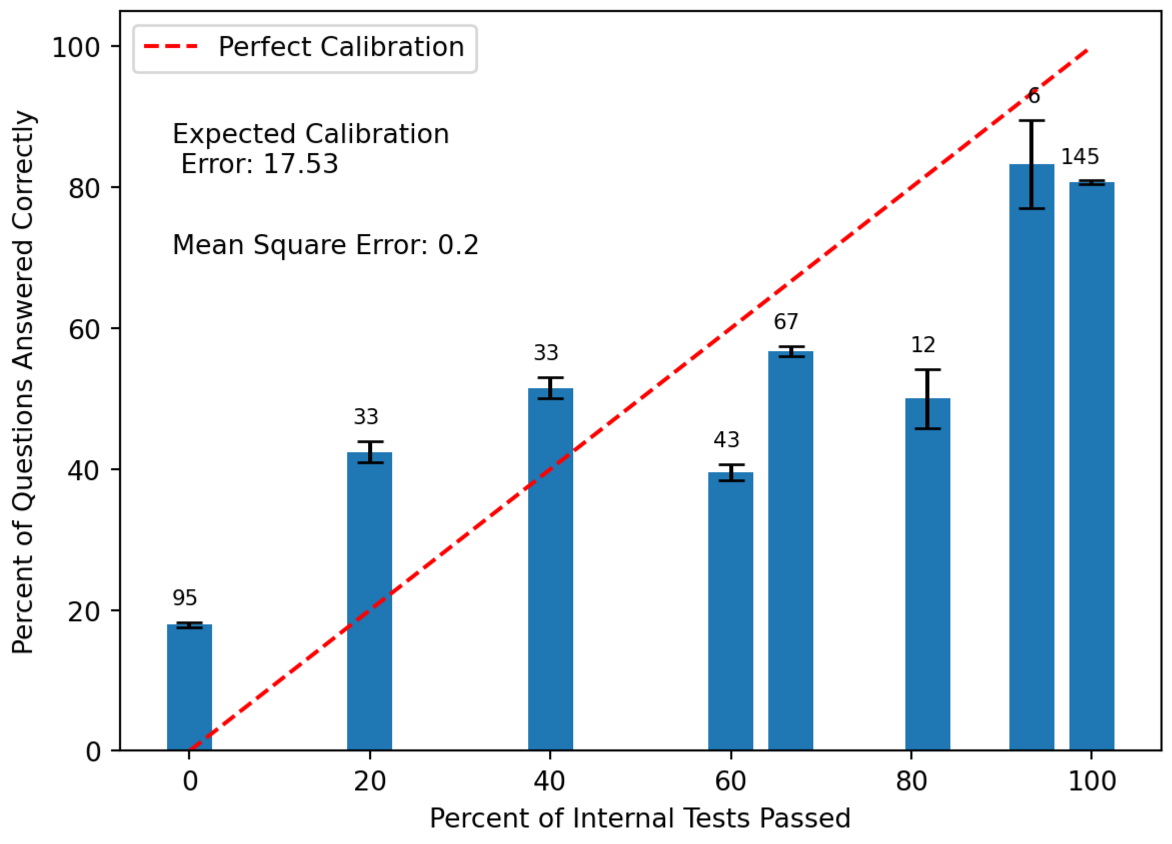

Figure 3: Calibration plot for GPT-3.5 on test cases

▶ Code

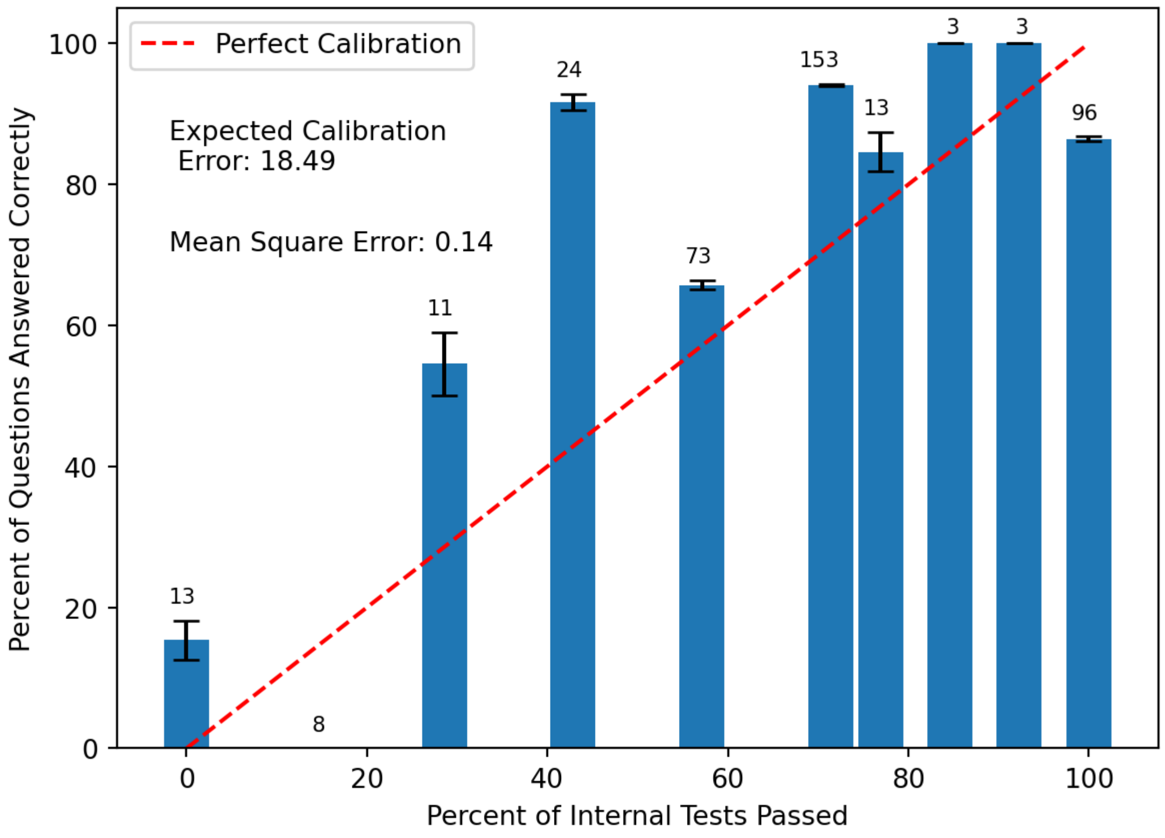

Figure 4: Calibration plot for GPT-4 on test cases

Unfortunately, the ECE values I was able to get on the experimental cases declined when evaluated on the held-out test cases. Conterintuitively, the Brier score slightly declined at the same time (indicating better calibration) between the experimental data and the evaluation data (potentially due to the average probabilities being further off but by minimizing some of the worst calibration errors (since Brier score/MSE involve the square of the error so put a higher penalty on outliers)). Additionally, the calibration of the test cases in the Reflexion setting seemed to consistently overconfident (a higher percent of internal tests passed than the percent of answers which were actually correct). This is likely due to the Reflexion frame resulting in the models optimizing for performing well on the internal test cases over correcting the underlying problems in the generated function.

Conclusions and Further Work

Using internal test cases, I was able to achieve a brier score of 0.20 and 0.14 for GPT-3.5 and GPT-4, respectively corresponding to an ECE of 17.5 and 18.5, respectively on over 100 held-out code generation questions from HumanEval. This model performance is significantly worse than previous calibration on True/False and multiple choice questions but comparable with previous results using model token probabilities for calibration on free-form question answering. These results show that internal test cases can perform as well as previous calibration methods for free-form question answering.

Promising directions for future work would be to evaluate some of the uncertainty quantification methods discussed in the “Previous Work” session on the HumanEval dataset to have a more direct comparison between this and other methods for uncertainty quantification. Another direction for work would be to try to expand the domain of questions being asked beyond questions in HumanEval or even beyond strictly programming questions. For example I am currently attempting to reformat the well-know MATH dataset into a form where I could test this method of uncertainty quantification. I would be excited to collaborate with others on further related work.

References:

1. Gordon, C. (2023, February 3). Chatgpt is the fastest growing app in the history of web applications. Forbes. https://www.forbes.com/sites/cindygordon/2023/02/02/chatgpt-is-the-fastest-growing-ap-in-the-history-of-web-applications/?sh=4f804a97678c

2. Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, & Jared Kaplan. (2022). Language Models (Mostly) Know What They Know.

3. Chenglei Si, Zhe Gan, Zhengyuan Yang, Shuohang Wang, Jianfeng Wang, Jordan Boyd-Graber, & Lijuan Wang. (2023). Prompting GPT-3 To Be Reliable.

4. Stephanie Lin, Jacob Hilton, & Owain Evans. (2022). Teaching Models to Express Their Uncertainty in Words.

5. Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, & Weizhu Chen. (2022). CodeT: Code Generation with Generated Tests.

6. OpenAI. (2023). GPT-4 Technical Report.

7. Shinn, N., Labash, B., & Gopinath, A. (2023). Reflexion: an autonomous agent with dynamic memory and self-reflection. arXiv preprint arXiv:2303.11366.