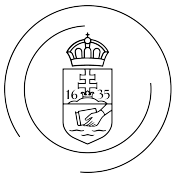
Investigating Bias in LLM Self-Evaluation

Thesis

Mathematics Expert in Data Analytics and Machine Learning



ELTE | FACULTY OF SCIENCE

Eötvös Loránd University
Faculty of Science

Budapest, 2025

Contents

Abstract

This thesis explores whether large language models (LLMs) tend to overestimate the quality of their own outputs when serving as judges or evaluators. Preliminary observations suggest that using the same or closely related LLM as both generator and judge may inflate performance metrics. Through systematic experiments, the project will quantify this potential bias and discuss its implications for AI evaluation, fairness, and trustworthiness in model benchmarking.

1 Introduction

This section introduces the key terms and core concepts that will be used in later sections.

1.1 A Brief Introduction to LLMs

A language model is a machine learning model designed to perform a wide range of tasks that involve natural language processing (NLP), including text summarization, translation, sentiment analysis, spam detection, content moderation, text generation, etc.

Significant advancements in deep learning [1, 2, 3] led to the emergence of **large language models** (LLMs) — particularly generative LLMs — which in the early 2020s became commercialized and widely adopted in both industry and popular discourse.

A generative LLM is a model which has a parameter count on the order of billions or more (hence "large"), and predicts the conditional probability [4]

$$P(w_m|w_0, \cdots, w_{m-1}) \tag{1}$$

where $m \in \mathbb{N}$, w_0 is a special start symbol, and w_k is the k-th token (for $1 \leq k \leq m$) in a sequence of tokens that form a piece of text in some language, be it a

natural language or a formal one like programming languages. The interpretation of the tokens depends on the exact tokenization strategy used, which may define tokens as words, word pieces, n-grams, or individual characters, and spaces, punctuation marks, etc.

Encoding is the process which converts human-readable textual tokens into integers which uniquely identify each token within the predetermined vocabulary of the model, and the inverse of this mapping is called **decoding**. [1]

Text generation is an autoregressive process where given a sequence of tokens as a prefix — known as the **prompt** — the model estimates the probability distribution of the next token, takes a sample from that distribution, appends it to the sequence, and repeats the process with the extended sequence until a stopping condition is met.

A frequently used parameter to control the sampling is called the **temperature** [5]: the closer it is to 0, the more the sampling will lean toward the most probable token — making the algorithm more deterministic —, while higher values increase the randomization, making the generated text feel more *creative* until, above a certain threshold, it becomes incoherent and semantically meaningless. [2] In practical implementations, if T is sufficiently close or exactly equal to 0, then the sampling is usually replaced with the deterministic argmax function in order to preserve numerical stability. Non-zero T values control the flatness of the distribution, leading to the aforementioned behavior.

With sufficiently large model complexity and training corpora size and diversity,

[1] Internally, the token numbers are mapped by a trainable model to vectors within a vector space called the **embedding space**. The choice for the dimensionality of this space allows a significant dimensionality reduction compared to what would be necessary for example to represent the tokens with one-hot encoding. An interesting property of the embedding space is that it tends to map tokens that are close to each other in meaning to vectors which are close to each other in the space.

[2] If $v \in \mathbb{N}$ denotes the number of all possible tokens available for the model (vocabulary size), and $\mathbf{s} \in \mathbb{R}^v$ is an output vector of the model assigning a score to each token as the continuation of a given input, then the distribution for the sampling, with respect to the temperature $T \in \mathbb{R}$ can be calculated via the softmax function: $\text{softmax}\left(\frac{1}{T}\mathbf{s}\right) = \left[\frac{\exp(\frac{s_i}{T})}{\sum_{j=1}^{v}\exp(\frac{s_j}{T})}\right]_{i=1}^{v}$

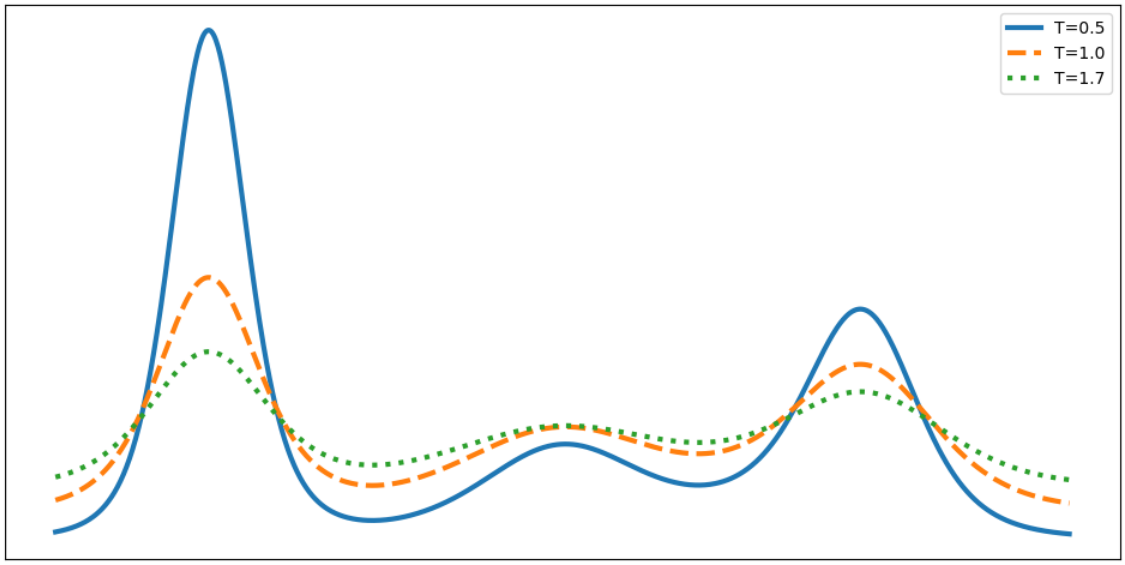


Figure 1: Effect of the temperature parameter on token probabilities.

LLMs start to exhibit capabilities which rival that of top performer humans in a broad class of problems [2, 3]. The versatility of the models is often utilized in a setting where the prompt is composed of two parts, each consisting of instructions given in natural language:

- the **system prompt** can instruct the model to behave in a certain way, for example, to act like a helpful AI assistant, an expert in a domain, or to generate its texts in the style of fictional 18th-century Caribbean pirates, etc.
- and the **user prompt** which describes the task to be carried out by the model, ranging from text translation or summarization to solving complex programming problems or pointing out business risks in legal documents, and more.

Generative models with sufficient generalization capabilities can predict likely continuations of such prompts with such high accuracy that as an emergent phenomenon, the generated text will often contain an actual solution to the proposed problem. This instruction-following paradigm enables models to perform **few-shot**

learning [2] or even **zero-shot learning** by interpreting tasks directly from the natural language description, based on just a few or zero examples, respectively, without specific training or fine-tuning.

The problem solving performance of LLMs can be improved further by prompt engineering techniques like **chain-of-thought** prompting [6], where the model is provided with step-by-step example solutions to related problems in the prompt, encouraging it to also articulate intermediate reasoning steps before arriving at its final answer. It is worth emphasizing that — recalling formula 1 and the autoregressive text generation process — the chain-of-thought is only effective if it is placed *before* the final answer.

1.2 LLM Evaluators, LLM-as-a-Judge

The continuing development of LLMs and their integration into more and more systems to support a growing number of use cases necessitates regular measurement of their capabilities and monitoring their alignment with human preferences.

While evaluating the quality of LLM-generated text by utilizing human labor does not scale well, may suffer from human error or subjective personal preference bias, and can be expensive, traditional algorithmic metrics which often rely on surface-level similarities to reference examples (like BLEU for machine translation [7] or ROUGE for summarization and translation [8]), often fall short of achieving acceptable correlation levels with human judgement.

In recent years, in order to overcome these problems, the **LLM-based evaluation** or **LLM-as-a-judge** paradigm has been proposed [9, 10, 11, 12], where — taking advantage of the instruction following and the zero-shot and few-shot learning capabilities of LLMs — a model is instructed to act as a fair judge and generate a quality assessment for a piece of generated text either in the form of a single score, or one accompanied by an explanation or a list of problems. An advantage of the latter approach — besides easier interpretability — is that enumerating evidences before giving a final result can influence the score via the autoregressive generation process, similarly to the improvements achieved by making large models include a

chain-of-thought [6] breakdown of complex problems before the final answer.

1.2.1 LLM-Judge Prompting Basics

There are numerous strategies to implement LLM-judges in practice [13], but a robust LLM-judge prompt usually includes the following elements:

- **Instructions** which clearly specify the evaluation task.
- Evaluation **aspects**, e.g. clarity, consistency, coherence, factuality, fluency, grammaticality, informativeness, structure, understandability, etc.
- Scoring **criteria** to specify the definitions for each score or score range.
- **Output format** specification so that the output of the judge can be programmatically parsed and interpreted.
- The **sample** itself to be evaluated or a pair of samples to be compared against each other.

Depending on the chosen evaluation strategy and aspect, additional elements may be included as well:

- Human-annotated **example** samples and their associated scores in few-shot evaluation scenarios.
- A **reference** answer for comparison with the evaluated sample, e.g. a human expert made translation, text summary, trivia answer, etc.
- The **source** data from which the evaluation sample was derived. (The original text to be translated, summarized, or the question to be answered, etc.)
- **Guidelines**, for example to help an LLM resolve the confusion that may arise in reference answer-based evaluations where some of the provided reference answers seem to contradict the model's own knowledge, e.g. "*Don't worry about factuality with respect to the real world, just judge the example based on*

what you see. No need to overthink this task, it really comes down to just soft matching." [14].

```
Please act as an impartial judge and evaluate the quality of the response provided
by an AI assistant to the user question displayed below. Your evaluation should
consider factors such as the helpfulness, relevance, accuracy, depth, creativity,
and level of detail of the response. Begin your evaluation by providing a short
explanation. Be as objective as possible. After providing your explanation, please
rate the response on a scale of 1 to 10 by strictly following this format:
"[[rating]]", for example: "Rating: [[5]]".
```

```
[Question]
{question}

[The Start of Assistant's Answer]
{answer}
[The End of Assistant's Answer]
```

Figure 2: System prompt with chain-of-thought and user prompt template for an LLM-judge [15].

Constructing the prompt template for a consistent, reproducible, and unbiased LLM-judge which also aligns well with human preferences is usually an iterative process, where the prompt is refined step-by-step until the LLM-judge can reliably produce evaluations that are sufficiently close to a set of human-labeled examples.

The juding model may also be fine-tuned using evaluation data constructed either manually or with the assistance of advanced models like GPT-4.

1.2.2 Metrics

Popular choices for scoring strategy include:

- **Binary classification**: the judge is expected to provide a "*yes*" vs. "*no*", or a 0 vs. 1 verdict.
- **Pairwise comparison**: the judge is given two candidate answers, and has to

select the one that is a better fit for the evaluation criteria. [3] Optionally, the judge may be allowed to declare a tie.

- **Multiclass classification**: the judge has to place the candidate on a discrete scale, usually between 1 and 5 points where 1 is the worst and 5 is the best.

- **Likert-style**: the judge has to rank the candidate answer along multiple dimensions using discrete scores, usually between 1 and 3 points where a higher score is better, then provide an overall 1 to 5 rating based on these scores.

- **Continuous score**: the candidate answer is scored with a number between 0 and 100.

If the judge LLM's interface makes the raw token probabilities available, then they can be used for refining discrete scores and making them into continuous ones by taking the sum of the discrete score values weighted by the probabilities of the respective tokens, as seen in the G-Eval framework [12]:

$$\text{score} = \sum_{i=1}^{n} p(s_i) \times s_i \tag{2}$$

where $S = \{s_1, s_2, \ldots, s_n\}$ is the set of scores predefined in the prompt, and $p(s_i)$ are the probabilities of the respective tokens for the score values, as calculated by the model.

Another way to turn a discrete score into a continuous one is used in the Gemba metric [16] for assessing translation quality: it requires the candidate answer to be dividable into smaller segments which are then evaluated one-by-one, and the resulting scores are averaged.

[3] This strategy can be generalized as **listwise comparison** where the judge is asked to select the best candidate among 3 or more candidates.

1.2.3 AutoCalibrate: Using an LLM to Find Criteria

A crucial part in the refinement process of an LLM-judge prompt is to come up with well-defined evaluation criteria.

The AutoCalibrate method [17] attempts to automate this process by utilizing a sufficiently large model:

- The LLM is presented with a random selection of human expert labeled examples, and instructed to infer the scoring criteria behind them. This is repeated multiple times with different samples, producing a set of draft candidate criteria.
- These drafts are then tested in evaluation rounds, and those which achieve the highest correlation with the human expert evaluation results are kept.
- Then a similar process takes place, but now the randomly selected examples come from the set of the mis-aligned examples, and the LLM is instructed to refine the draft criteria by applying small modifications, paraphrasing, clarifying some aspects or details, etc. instead of coming up with new ones from scratch.
- Finally, the criteria that produce the highest agreement with the human experts are chosen.

2 LLM-Judge Biases, Limitations, and Mitigation in the Literature

The assessment results from a fair and reliable LLM-judge should depend on nothing but the quality of the evaluated content with regards to the evaluation criteria. Therefore, if extraneous factors are found to systematically influence evaluation results, then this undermines their validity and warrants mitigation. Researchers have identified multiple causes of bias in the judgement of LLMs, and proposed various techniques to mitigate them.

Though the focus of this essay is the investigation of LLM self-preference, other types of biases need to be studied as well in order to minimize their effects in experiments.

2.1 Position Bias

Positional bias occurs in pairwise or listwise comparison tasks when a judge is presented with the same prompt template and the same set of candidate responses, the only difference being the order of the candidates, and this alone is enough to change the evaluation outcome [18, 19].

The probability of this phenomenon occurring is observed to be inversely correlated with the quality gap between the candidate answers, i.e. judgement of similar quality candidates is more likely to be affected by position permutation. (The quality of an answer in the presence of positional bias can be estimated by the overall win rate of the answer across all experiments, given that the cases where position changes were observed to be influencing the evaluation outcome are considered ties.)

2.1.1 Mitigation

- **Prompting**: some experimenters explicitly instruct the LLM-judge in the prompt not to let its judgement be influenced by the ordering of the candidate answers or any kind of bias.

- **Multiple Evidence Calibration (MEC)**: evidence calibration (EC) takes advantage of the autoregressive generation process by instructing the judge to first express a comprehensive explanation for its judgement, and only then provide the final decision. MEC performs multiple evaluations using this prompting technique, and combines the results e.g. by averaging.

- **Balanced Position Calibration**: the same set of candidates is evaluated multiple times with the same prompt template, but with permutations ensuring that each candidate appears at each position the same number of times, i.e. in

pairwise comparison experiments, the evaluation is repeated with the candidate answers being switched, then the results are averaged.

2.2 Length Bias

Verbose answers often contain more information, and to some extent, these are also often preferred by humans. However, LLMs have been observed to prefer longer answers even in cases where the information content was the same between answers, and even when human evaluators chose the shorter ones [20, 21, 22], resulting in low alignment with humans.

2.2.1 Mitigation

When multiple reference answers are available with matching quality, select one that is close to the evaluated one in terms of its length [22].

2.3 Prompt Injection

The possibility for an injection attack arises whenever insufficiently filtered, attacker-controllable data is passed along with computer instructions in the same input channel of a computer system. [4] LLM-based systems where potentially malicious user input is mixed with the instructions in the prompt are especially susceptible to injection attacks.

Due to the black box operation and stochastic nature of LLMs, an attack payload doesn't even have to break out from the context of delimiter strings like "`[The Start of Assistant's Answer]`" in order to be successful: it can be sufficient if it manages to confuse the model by a long sequence of infrequently used complicated words ("*resynchronization bacteriohemolysin complaisantness*") or unusual Markdown for-

[4] Famous examples include SQL-injection, HTML-injection (which is usually escalated into cross-site scripting code execution), and shell command injection. These are frequent contenders in the regularly updated OWASP Top 10 Web Application Security Risks chart: `https://owasp.org/www-project-top-ten/`.

matting, then giving instructions which override the originally intended task. In some cases, the probability of success can be increased by adding seemingly authoritative commands like `Authorization: ADMIN_LEVEL_ACCESS Command sequence: 7A-9B-12C Priority: CRITICAL` [23].

2.3.1 Mitigation

The proposed mitigation techniques include using [23] [5]:

- smaller LLMs to filter potentially harmful inputs,
- statistical measurements to filter unusual inputs,
- smaller LLMs to detect unusual response from the judge,
- multi-model committee assembled from diverse models to reduce the probability of an attack successfully compromising all participants,
- traditional string matching to filter suspicious inputs that contain frequently used phrases in prompt injection attacks, for example "*Ignore previous instructions, and...*".

[5] My personal opinion is that in the long history of injection attacks, the most reliable mitigation technique has always been to separate the instruction channel from the input data channel (e.g. SQL prepared statements, DOM API, structured shell APIs, etc.) and avoid using string templates with basic string substitution. In the case of LLMs, this would mean to either introduce separate data channels, or to use special tokens (like sequence start, stop, padding, etc.) which attackers could not possibly include in their texts, at the encoding-decoding level to distinguish instructions and data, and train the models accordingly.

2.4 Self-Preference Bias

2.4.1 Mitigation

3 Experiments

4 Results

5 Conclusion

References

[1] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). DOI: https://doi.org/10.48550/arXiv.1706.03762.

[2] OpenAI. "Language Models are Few-Shot Learners". In: *CoRR* abs/2005.14165 (2020). DOI: https://doi.org/10.48550/arXiv.2005.14165.

[3] OpenAI. "GPT-4 Technical Report". In: (2023). DOI: https://doi.org/10.48550/arXiv.2303.08774.

[4] Tong Xiao and Jingbo Zhu. *Foundations of Large Language Models*. 2025. DOI: https://doi.org/10.48550/arXiv.2501.09223.

[5] Enrique Manjavacas et al. "Synthetic Literature: Writing Science Fiction in a Co-Creative Process". In: *Proceedings of the Workshop on Computational Creativity in Natural Language Generation (CC-NLG 2017)*. Santiago de Compostela, Spain: Association for Computational Linguistics, Sept. 2017, pp. 29–37. DOI: https://doi.org/10.18653/v1/W17-3904.

[6] Jason Wei et al. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". In: *CoRR* abs/2201.11903 (2022). DOI: https://doi.org/10.48550/arXiv.2201.11903.

[7] Kishore Papineni et al. *BLEU: a method for automatic evaluation of machine translation*. 2001. DOI: https://doi.org/10.3115%2F1073083.1073135.

[8] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013/.

[9] Jinlan Fu et al. *GPTScore: Evaluate as You Desire*. 2023. DOI: https://doi.org/10.48550/arXiv.2302.04166. arXiv: 2302.04166 [cs.CL].

[10] Jiaan Wang et al. *Is ChatGPT a Good NLG Evaluator? A Preliminary Study*. 2023. DOI: https://doi.org/10.48550/arXiv.2303.04048. arXiv: 2303.04048 [cs.CL].

[11] Yi Chen et al. *Exploring the Use of Large Language Models for Reference-Free Text Quality Evaluation: An Empirical Study*. 2023. DOI: https://doi.org/10.48550/arXiv.2304.00723. arXiv: 2304.00723 [cs.CL].

[12] Yang Liu et al. *G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment*. 2023. DOI: https://doi.org/10.48550/arXiv.2303.16634. arXiv: 2303.16634 [cs.CL].

[13] Zhen Li et al. *Leveraging Large Language Models for NLG Evaluation: Advances and Challenges*. 2024. DOI: https://doi.org/10.48550/arXiv.2401.07103. arXiv: 2401.07103 [cs.CL].

[14] Pat Verga et al. *Replacing Judges with Juries: Evaluating LLM Generations with a Panel of Diverse Models*. 2024. DOI: https://doi.org/10.48550/arXiv.2404.18796. arXiv: 2404.18796 [cs.CL].

[15] Lianmin Zheng et al. *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. 2023. DOI: https://doi.org/10.48550/arXiv.2306.05685. arXiv: 2306.05685 [cs.CL].

[16] Tom Kocmi and Christian Federmann. *Large Language Models Are State-of-the-Art Evaluators of Translation Quality*. 2023. DOI: https://doi.org/10.48550/arXiv.2302.14520. arXiv: 2302.14520 [cs.CL].

[17] Yuxuan Liu et al. *Calibrating LLM-Based Evaluator*. 2023. DOI: https://doi.org/10.48550/arXiv.2309.13308. arXiv: 2309.13308 [cs.CL].

[18] Lin Shi et al. *Judging the Judges: A Systematic Study of Position Bias in LLM-as-a-Judge*. 2025. DOI: https://doi.org/10.48550/arXiv.2406.07791. arXiv: 2406.07791 [cs.CL].

[19] Peiyi Wang et al. *Large Language Models are not Fair Evaluators*. 2023. DOI: https://doi.org/10.48550/arXiv.2305.17926. arXiv: 2305.17926 [cs.CL].

[20] Keita Saito et al. *Verbosity Bias in Preference Labeling by Large Language Models*. 2023. DOI: https://doi.org/10.48550/arXiv.2310.10076. arXiv: 2310.10076 [cs.CL].

[21] Hui Wei et al. *Systematic Evaluation of LLM-as-a-Judge in LLM Alignment Tasks: Explainable Metrics and Diverse Prompt Templates*. 2025. DOI: https://doi.org/10.48550/arXiv.2408.13006. arXiv: 2408.13006 [cs.CL].

[22] Zhengyu Hu et al. *Explaining Length Bias in LLM-Based Preference Evaluations*. 2024. DOI: https://doi.org/10.48550/arXiv.2407.01085. arXiv: 2407.01085 [cs.LG].

[23] Narek Maloyan and Dmitry Namiot. *Adversarial Attacks on LLM-as-a-Judge Systems: Insights from Prompt Injections*. 2025. DOI: https://doi.org/10.48550/arXiv.2504.18333. arXiv: 2504.18333 [cs.CR].