User-side Model Consistency Monitoring for Open Source Large Language Models Inference Services

Qijun Miao[1,2] and **Zhixuan Fang**[1,2*]
[1]Tsinghua University
[2]Shanghai Qi Zhi Institute
mqj24@mails.tsinghua.edu.cn, zfang@mail.tsinghua.edu.cn

Abstract

With the continuous advancement in the performance of open-source large language models (LLMs), their inference services have attracted a substantial user base by offering quality comparable to closed-source models at a significantly lower cost. However, it has also given rise to trust issues regarding model consistency between users and third-party service providers. Specifically, service providers can effortlessly degrade a model's parameter scale or precision for more margin profits, and although users may perceptibly experience differences in text quality, they often lack a reliable method for concrete monitoring. To address this problem, we propose a paradigm for model consistency monitoring on the user side. It constructs metrics based on the logits produced by LLMs to differentiate sequences generated by degraded models. Furthermore, by leveraging model offloading techniques, we demonstrate that the proposed method is implementable on consumer-grade devices. Metric evaluations conducted on three widely used LLMs series (OPT, Llama 3.1 and Qwen 2.5) along with system prototype efficiency tests on a consumer device (RTX 3080 TI) confirm both the effectiveness and feasibility of the proposed approach.

1 Introduction

Recent advancements in pre-trained language models based on the Transformer architecture (Vaswani, 2017) have demonstrated remarkable capabilities across various NLP tasks, including machine translation, question answering, creative writing, etc., which significantly boost operational efficiency in industry and enhance everyday user experiences. Consequently, LLMs inference services possess a large and continuously growing user base.

Among the numerous LLMs series, while closed-source powerful models like ChatGPT (OpenAI, 2025) and Claude (Anthropic, 2025) have gained significant attention, the rapid development of open-source models has also yielded a number of outstanding examples, such as the Llama (Dubey et al., 2024), Qwen (Yang et al., 2024), and DeepSeek (Guo et al., 2025) series, which demonstrate remarkable performance across various domains. Consequently, a growing number of third-party service providers (OpenRouter, 2025; together.ai, 2025; Groq, 2025) are dedicated to offering inference services for these open-source models. These providers have garnered substantial user preference due to their more accessible pricing structures compared to their closed-source counterparts.

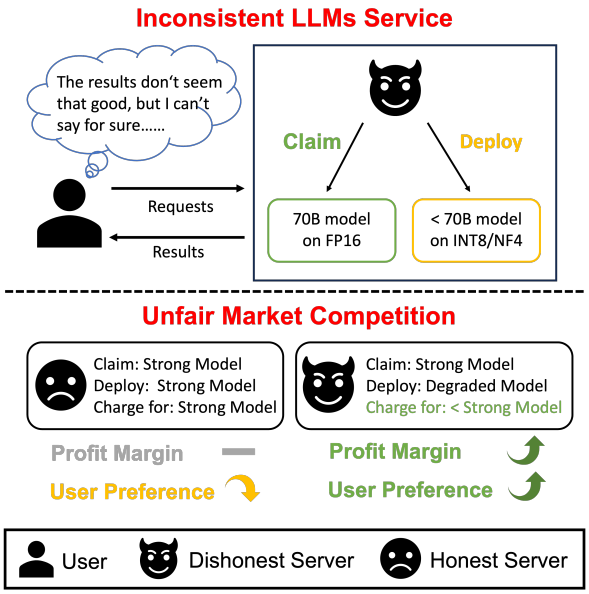

Figure 1: The lack of model consistency monitoring not only undermines the quality of LLM services for users but may also give rise to unfair market competition.

However, due to the lack of transparency of such third-party inference services, an inevitable trust issue arises between users and service providers. Service providers may run a lower-precision quantized version (Dettmers et al., 2022; Yao et al., 2022;

*Corresponding author.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 11610–11622

Xiao et al., 2023; Dettmers et al., 2024) of the model specified by the user, or a smaller-parameter version within the same model series, thereby gaining higher margin profits. This practice can persist despite users' potential awareness of quality degradation, as they lack verifiable evidence to substantiate their concerns, as shown in the upper part of Figure 1. Furthermore, dishonest service providers can exploit users' inability to monitor model consistency to gain a pricing advantage over honest providers, thereby triggering unfair competition, as shown in the lower part of Figure 1. Therefore, from a user-centric standpoint, we emphasize that monitoring the model consistency should also be an important research direction in trustworthy LLMs services, alongside optimizing the performance of the service systems.

While substantial research has been conducted in trusted AI inference services, existing methods inevitably incur significant additional costs, including time and economic expenses. ZKML (Chen et al., 2024) has been proposed to generate zero-knowledge proofs for computations during model inference, enabling users to verify the provenance of received results from designated models. However, the prohibitively high computational overhead required for proof generation confines this approach's applicability exclusively to minimal-scale models. OPML (Conway et al., 2024) seeks to leverage blockchain technology, where on-chain verifiers are required to re-execute the service provider's computations, with result disputes arbitrated through smart contracts to ensure output authenticity. However, the decentralized protocol's inherent characteristics dictate that result credibility grows with the number of verifiers, while the user's economic burden escalates commensurately. Furthermore, the execution of smart contracts introduces non-negligible latency as well. As a result, the practical feasibility of these approaches is limited to small-scale models, making them impractical for LLMs with billions of parameters.

In summary, our goal is to *enable trustworthy monitoring of model consistency for LLM inference services without imposing significant time or economic burden on users*. Fortunately, the characteristics of LLMs generation paradigm, combined with memory-efficient model deployment techniques, present an opportunity to address this challenge. Based on this, we propose a new paradigm for model consistency monitoring at the user end, in which all additional computations are performed on the user's consumer-grade device, which provides not only acceptable extra time and economic costs but also fully trustworthy results.

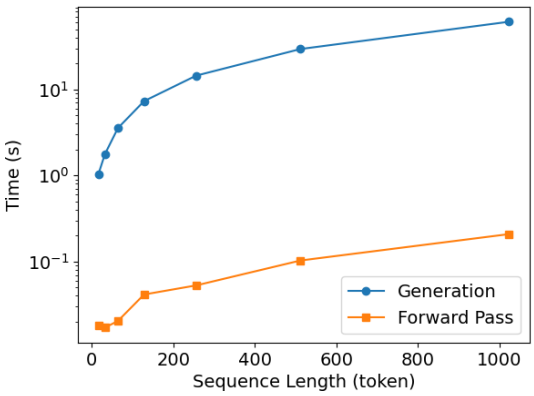

Figure 2: Time taken for sequence generation and a single forward pass of Llama-3.1-8B on an A40.

First, the autoregressive decoding paradigm for LLMs generation provides an opportunity on the computation side. An LLM computes the probability distribution for the next token in a sequence through a single forward pass, followed by employing different sampling methods to select the subsequent token. Consequently, the time cost for sequence generation roughly exhibits linear proportionality to the resultant token length. However, when presented with a generated sequence, the LLM can compute all next-token probability distributions within the sequence through merely one forward propagation, which actually provides sufficient information to monitor the model consistency. Leveraging the massively parallel architecture of modern GPUs, the time cost of the latter procedure is much lower than that of the former as shown in Figure 2.

Second, model offloading (Aminabadi et al., 2022; Sheng et al., 2023) provides another opportunity on the memory side. It's a well-established technique for mitigating device memory constraints in LLMs deployment, operating through strategic parameter storage in CPU and disk resources when model parameters exceed GPU memory capacity. While fundamentally constrained by I/O bottlenecks that induce significant latency penalties, Sheng et al. (2023) demonstrate remarkable throughput scalability when processing batched inputs through efficient computation scheduling to overlap the parameter loading procedure. It implies

that GPU memory constraints are noncritical for latency-insensitive applications, which crucially include the model consistency monitoring we mainly discuss in this paper.

Based on the proposed user-side monitoring paradigm, the first focus of this work is to design a metric that can distinguish between sequences generated by the user-specified model and those from other models. The primary challenge stems from cross-platform floating-point computation inconsistency. Specifically, differences in the hardware and software environments between users and service providers can lead to variations in the scheduling order of floating-point operations, ultimately causing discrepancies in final results. It is crucial for us to *differentiate between result variations caused purely by floating-point errors and those arising from model degradation*.

Considering the sequential nature of the LLMs generation procedure, we intuitively hypothesize that any potential model degrading behaviors by service providers could lead to cumulative errors across the generated sequence. Consequently, we design a novel metric based on the model's output probabilities to the generated sequence, termed token confidence. The metric can perform aggregation on subsequences of arbitrary length, and therefore, it can distinguish errors resulting from model degrading from those arising purely from cross-platform floating-point computation inconsistency. Extensive experiments were conducted across three LLMs series and two different hardware architectures, and the results substantiate the broad effectiveness of the proposed metric.

Next, we implemented a prototype for computing the proposed metrics on the user side, based on the state-of-the-art offloading system FlexGen (Sheng et al., 2023), and performance tests on a consumer-grade RTX 3080 TI device confirmed the feasibility of user-side model consistency monitoring for LLMs services.

The contributions of this work can be summarized as follows:

- We propose a novel paradigm for verifying LLMs services that leverages users' own consumer-grade devices, which ensures fully trustworthy results while not introducing substantial extra costs compared to previous paradigms.
- We design a metric, termed token confidence, to distinguish low-quality outputs generated by potential model degrading behaviors. Extensive experiments demonstrate the broad effectiveness of this metric.
- We implement a prototype for computing token confidence on consumer-grade devices. Performance tests conducted on an RTX 3080Ti confirm the feasibility of user-side model consistency monitoring.

2 Related Works

Trusted AI service. Previous research in trusted AI services mainly focuses on employing cryptography or blockchain technologies to ensure service providers execute inference tasks using user-specified models. ZKML (Chen et al., 2024) requires providers to generate zero-knowledge proofs alongside model outputs, enabling users to verify computational integrity through proof validation. OPML (Conway et al., 2024) leverages blockchain technology with third-party verification nodes, where verifiers re-execute model computations after the service provider submitting results. Discrepancies trigger arbitration via smart contracts, establishing trustworthiness under the assumption that at least one honest node exists among the submitter and verifiers. Opp/ai (So et al., 2024) hybridizes the above two approaches to trade-off between efficiency and privacy. Collectively, these frameworks introduce substantial time and economic overheads, rendering them impractical for LLMs services.

A common challenge in trusted AI services lies in cross-platform floating-point computation errors, which implies that the result discrepancy between service providers and verifiers may stem not only from model inconsistencies but also from hardware/software heterogeneity. Previous work (Zheng et al., 2021; Conway et al., 2024; Srivastava et al., 2024) has attempted to eliminate cross-platform floating-point computation errors through methods such as replacing floating-point operations with integer-based operations or specifically developing consistent floating-point computation libraries. However, these methods not only probably degrade model performance but also impose additional deployment costs on service providers.

Logits-based metrics. In transformer-based LLMs, a single forward pass generates a vector called logits for each token in the input sequence, where the vector length corresponds to the number of tokens in the vocabulary. During autoregressive

generation, applying a softmax operation to the logits produces the probability distribution for the next token. Beyond the widely-used application of sequence generation, the logits of tokens in a complete sequence intuitively reflect the model's confidence in that sequence. Consequently, a large body of research has focused on designing logits-based metrics for tasks such as self-evaluation of language models (Lin et al., 2024; Ren et al., 2023) and detecting machine-generated text (Gehrmann et al., 2019; Solaiman et al., 2019; Ippolito et al., 2019; Mitchell et al., 2023).

It is important to note that these tasks differ fundamentally from the problem mainly discussed in this work: distinguishing sequences generated by degraded models from those generated by the specified model. This fundamental difference renders the metrics designed in prior work unsuitable for our problem. Detailed discussions and corresponding experimental results are available in Appendix A.

Model offloading. Model offloading is a technique that leverages CPU and disk resources to enable model inference when GPU memory is insufficient to load the entire model. In *GPU-centric offloading* (Aminabadi et al., 2022; Sheng et al., 2023), CPU and disk are used solely for storing model parameters, which are dynamically loaded into the GPU as needed for computation, with all computations performed on the GPU. Meanwhile, *hybrid offloading* (Gerganov, 2025; Song et al., 2024) assigns computational tasks to the CPU as well, reducing the overhead of transferring data to the GPU.

However, in the hybrid offloading mode, if the model size exceeds the combined memory capacity of the GPU and CPU, the operating system's virtual memory mechanism results in frequent swapping between CPU memory and disk. This often leads to crashes. Therefore, this work adopts FlexGen (Sheng et al., 2023), the state-of-the-art GPU-centric offloading method, to implement our system prototype. Although frequent parameter transfers to the GPU result in higher latency for a single sequence, FlexGen achieves considerable throughput when handling multiple sequences through efficient computation scheduling.

3 Methods

In this section, we first present the high-level architecture and workflow of user-side model consistency monitoring (Section 3.1). Subsequently, we introduce *token confidence* as the key metric in the monitoring process (Section 3.2). Finally, we demonstrate how this metric is used to monitor the model consistency of LLMs services (Section 3.3).

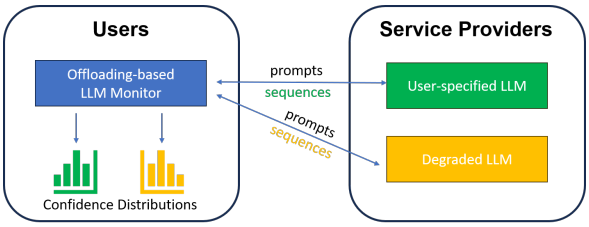

Figure 3: The high-level architecture of user-side model consistency monitoring.

3.1 System Architecture

The high-level architecture of user-side model consistency monitoring is demonstrated in Figure 3. The left hand side is an offloading-based LLM monitor running on users' consumer-grade devices, while the right hand side shows potential LLMs implementations by service providers for sequence generation. Honest providers strictly employ the user-specified LLM, whereas strategic providers might substitute with degraded models.

First, users send their prompts to service providers, who subsequently return generated sequences. Upon accumulating sufficient sequences, the offloading-based LLM monitor starts batch processing. Specifically, the user-specified LLM is loaded to perform a single forward propagation on generated sequences. Then, token confidences are computed for each token in a sequence, quantifying its likelihood of being generated by the user-specified model. Finally, aggregating confidence-based metrics across all sequences will construct a distribution, which indicates the model consistency of the service provider.

3.2 Token Confidence

In the remaining part of this section, we introduce the proposed method based on the greedy sampling scenario. Specifically, the user requests the service provider to select the token with the highest probability as the next token in the sequence at each iteration. After that, in Section 5, we demonstrate that our approach can be easily extended to the stochastic sampling scenario while maintaining equivalent effectiveness.

We first denote a sequence generated by the service provider as $\boldsymbol{x}$, where a token in the sequence is

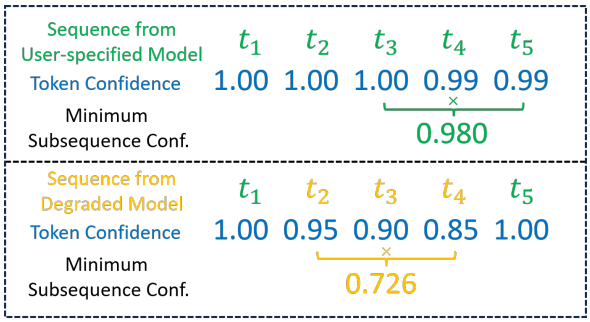

Figure 4: In consecutive subsequences, the aggregate token confidence differentiates the sequence generated by the degraded model from merely floating-point computation errors ($W = 3$).

represented as $x_i \in \boldsymbol{x}$. Let the logits produced by the user-specified model θ for the sequence prefix $\boldsymbol{p}_i = x_1 x_2 \cdots x_{i-1}$ be denoted as $\text{logits}_\theta(v \mid \boldsymbol{p}_i)$, where $v \in \mathcal{V}$ and $\mathcal{V}$ is the vocabulary. Since the model's expected behavior is to select the token with the highest probability (i.e., the token with the largest logits), we define the *logits difference* of x_i as follows to measure the deviation of the model's selection of x_i from its expected behavior:

$$\text{diff}_i = \text{logits}_\theta(x_i \mid \boldsymbol{p}_i) - \max_{v \in \mathcal{V}} \text{logits}_\theta(v \mid \boldsymbol{p}_i). \quad (1)$$

If the sequence was indeed generated by the user-specified model, the logits difference will solely result from floating-point computation errors caused by hardware and software differences between the user and the service provider. We reasonably assume that such errors are numerically much smaller than those caused by model degrading[1].

Next, we apply an exponential operation to logits difference to constrain its range between 0 and 1, thereby introducing the *token confidence*:

$$\text{confidence}_i = \exp \frac{\text{diff}_i}{\max(ki, 1)}. \quad (2)$$

Positional correction. Since cross-platform floating-point computation errors accumulate as the sequence length increases, we introduce the term $\max(ki, 1)$ in Equation 2 to mitigate this effect, where $k > 0$ is a hyperparameter.

3.3 Model Consistency Monitoring

Token confidence can evaluate how the service provider's model deviates from expected behavior at the token level. Nevertheless, a key observation is that a powerful small model can generate short sequences that are nearly indistinguishable from those generated by a large model (Miao et al., 2024). Therefore, we consider computing the product of token confidence values over consecutive subsequences to obtain *subsequence confidence*:

$$\text{confidence}_{i,i+W} = \prod_{i \le j < i+W} \text{confidence}_i, \quad (3)$$

where $W > 0$ is a hyperparameter representing the subsequence window size. It is reasonable to assume that a sufficiently large W can reveal the degraded capability of a small model.

Then, we use the minimum value to aggregate subsequence confidence into *sequence confidence*, which indicates the model's consistency at the sequence level:

$$\text{confidence} = \min_{1 \le i \le L-W+1} \text{confidence}_{i,i+W}, \quad (4)$$

where L denotes the sequence length in tokens. An intuitive example of how the aforementioned metrics work is illustrated in Figure 4.

Finally, we can monitor the model consistency of a service provider by Algorithm 1. For each sequence generated, we first call the user-specified LLM on it to get the logits of each token. Then we compute the sequence confidence by Equation 4. The resulting sequence confidence distributions among all generated sequences can be easily visualized and compared across different service providers, as will be demonstrated in Section 4.

Algorithm 1 Model Consistency Monitoring

Input: User-specified LLM θ, sequence set $\mathcal{S}$ from a service provider
Output: Sequence confidence distribution $\mathcal{C}$

$\mathcal{C} \leftarrow \emptyset$
for $\boldsymbol{x} \in \mathcal{S}$ **do**
  $\text{logits}_\theta(\boldsymbol{x}) \leftarrow$ call LLM θ with input $\boldsymbol{x}$
  $c \leftarrow$ compute confidence for $\boldsymbol{x}$ by Eq. 4
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$
end for
return $\mathcal{C}$

4 Evaluation

4.1 Algorithm Effectiveness

We evaluate the effectiveness of Algorithm 1 from three perspectives: cross-platform robustness, size

[1] If the errors caused by model degrading are of the same magnitude as cross-platform floating-point computation errors, then the degraded model is already sufficiently strong.

Group	Monitoring Config			Generation Config		
	Model	Precision	GPU	Model	Precision	GPU
Cross-platform Robustness	OPT-66B	FP16	A100	OPT-66B	FP16	A100
				OPT-66B	FP16	H100
	Llama-3.1-70B	FP16	A100	Llama-3.1-70B	FP16	A100
				Llama-3.1-70B	FP16	H100
	Qwen-2.5-72B	FP16	A100	Qwen-2.5-72B	FP16	A100
				Qwen-2.5-72B	FP16	H100
Size Degrading	OPT-66B	FP16	A100	OPT-30B	FP16	A100
				OPT-13B	FP16	A100
	Llama-3.1-70B	FP16	A100	Llama-3.1-8B	FP16	A100
	Qwen-2.5-72B	FP16	A100	Qwen-2.5-32B	FP16	A100
				Qwen-2.5-14B	FP16	A100
Precision Degrading	OPT-66B	FP16	A100	OPT-66B	INT8	A100
				OPT-66B	NF4	A100
	Llama-3.1-70B	FP16	A100	Llama-3.1-70B	INT8	A100
				Llama-3.1-70B	NF4	A100
	Qwen-2.5-72B	FP16	A100	Qwen-2.5-72B	INT8	A100
				Qwen-2.5-72B	NF4	A100

Table 1: Settings for evaluating algorithm effectiveness.

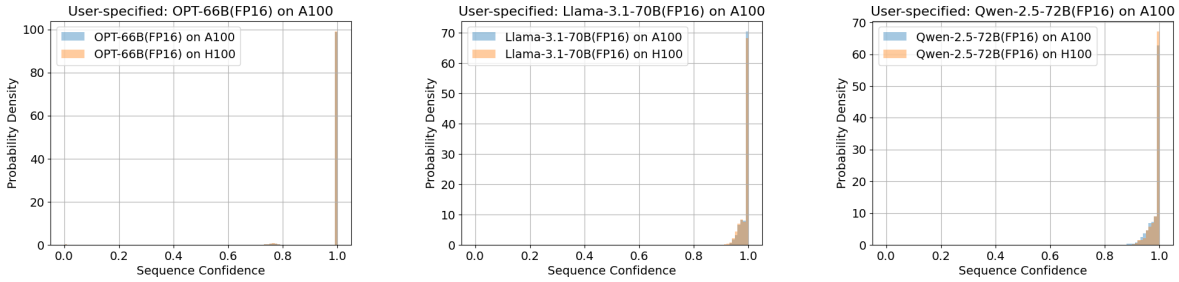

Figure 5: Sequence confidence distributions of the cross-platform robustness group.

sensitivity, and precision sensitivity. Our experiments are conducted on three widely used open-source LLMs series, OPT (Zhang et al., 2022), Llama 3.1 (Dubey et al., 2024), and Qwen 2.5 (Yang et al., 2024), as well as on GPUs from two hardware architectures: A100 80GB (Ampere architecture) and H100 80GB (Hopper architecture).

Settings. As shown in Table 1, each row represents a single user-server pair of model consistency monitoring. First, we apply the Generation Config to generate 1,000 sequences[2] using 1,000 prompts extracted from the Alpaca dataset (Taori et al., 2023) as input. Then, we apply the Monitoring Config to execute Algorithm 1 on generated sequences and obtain the Sequence Confidence Distribution. The hyperparameters in Algorithm 1 are set as follows[3]: $W = 10, k = 0.05$. The INT8 and NF4 quantization of model parameters follow the methods described in (Dettmers et al., 2022) and (Dettmers et al., 2024), respectively. The number of GPUs used for running the LLMs is minimized to the smallest number required.

Cross-platform robustness. In this group of experiments, we aim to evaluate the impact of floating-point computation errors caused by differences in computing platforms on sequence confidence. The results, as shown in Figure 5, indicate that for the three LLMs, an honest service provider, even when using GPUs with different architectures from the user's, produces logits differences within a relatively small range. Consequently, the resul-

[2]Maximum tokens in a sequence is set to 1024.

[3]The majority of the computational overhead arises from invoking the user-specified LLM to generate logits. So the two hyperparameters in the algorithm (consecutive subsequence window size W and coefficient of the positional correction term k) can both be efficiently adjusted in practice. Therefore, we defer the detailed study of these hyperparameters to Appendix B

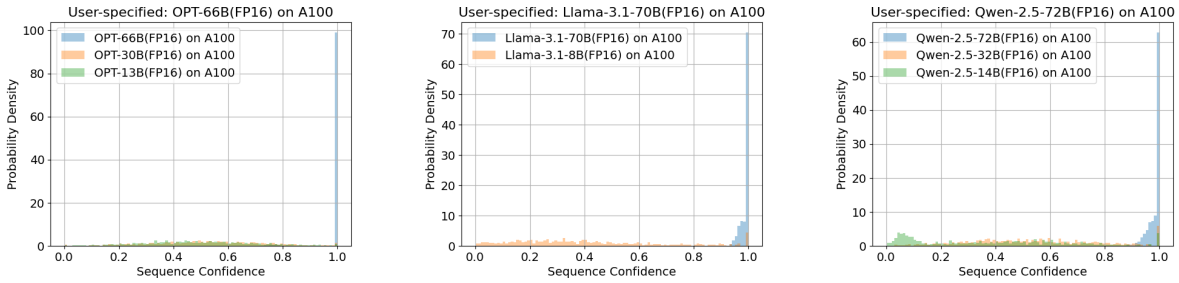

Figure 6: Sequence confidence distributions of the size degrading group.

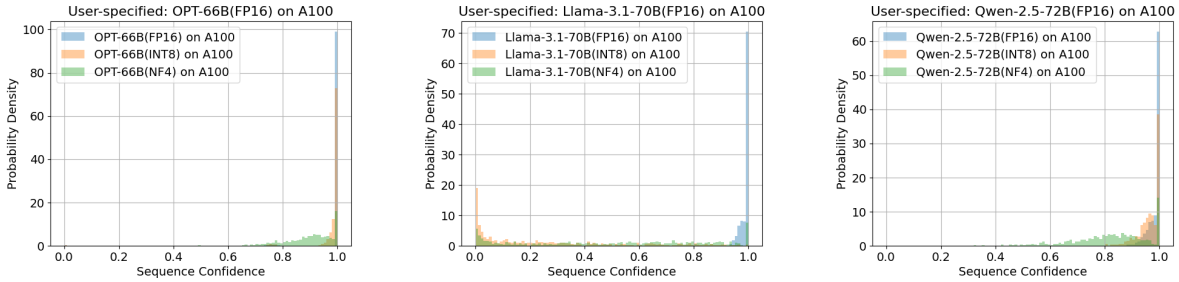

Figure 7: Sequence confidence distributions of the precision degrading group.

tant sequence confidence remains very close to 1. Specifically, for the OPT-66B, sequence confidence is the least sensitive to cross-platform differences, with almost all sequences achieving a confidence score higher than 0.99. For Llama-3.1-70B[4] and Qwen-2.5-72B, approximately 70% of sequences have confidence scores higher than 0.99, and nearly all sequences have that higher than 0.9.

It is worth noting that even when the service provider and the user use the same hardware architecture, floating-point computation errors can still occur due to differences in the instruction scheduling between the autoregressive generation and the parallel forward propagation on GPUs. As observed, this factor has a more significant impact on sequence confidence compared to differences in hardware architectures. Therefore, in the next two sets of experiments involving degraded models, we use the same GPUs for sequence generation and monitoring to reduce the redundant workload.

Size degrading. In this group of experiments, we aim to evaluate whether the sequence confidence distribution can effectively distinguish sequences generated by smaller-parameter models within the same series from those generated by the user-specified model. The results, shown in Figure 6, present the distributions for each LLM series in separate subplots. As observed, sequences generated by smaller-parameter models within the same series tend to have confidence values either uniformly distributed between 0 and 1 or partly concentrated in the range close to 0. In contrast, sequences from the user-specified model exhibit a distinct distribution, with the majority of confidence values concentrated between 0.9 and 1, which creates a clear differentiation.

Precision degrading. In this group of experiments, we aim to evaluate whether the sequence confidence distribution can effectively distinguish sequences generated by lower-precision versions of the same model from those generated by the user-specified model. The results, shown in Figure 7, present the distributions for each LLM series in separate subplots. As observed, a significant portion of the sequence confidence produced by lower-precision models is still concentrated near 1. This indicates that parameter quantization techniques have a relatively limited impact on LLMs generation.

Nevertheless, there are still noticeable differences between the full-precision and lower-precision models in terms of the proportion of sequences with extremely high confidence (greater than 0.99) and the magnitude of the lower quantiles. These differences further demonstrate the effectiveness of the proposed algorithm.

In addition, the results of this experiment empiri-

[4]Due to hardware limitations, we selected the 70B version instead of the 405B version as the largest-parameter model in the Llama 3.1 series.

Device	Model	Memory
GPU	NVIDIA RTX 3080 Ti	12GB
CPU	Intel Xeon @ 2.10GHz	376GB
Disk	Samsung SSD (NVMe)	960GB

Table 2: Hardware specs in system efficiency evaluation.

cally reveal several insights. First, the impact of the same quantization method varies across different models. For instance, the Llama-3.1-70B model is significantly more affected by reduced precision compared to the other two models. Interestingly, and somewhat counterintuitively, NF4 quantization on Llama-3.1-70B achieves results closer to the original precision than INT8 quantization. Another practical insight relates to model deployment under limited resource budgets: with similar memory requirements, lower-precision versions of larger models can achieve results closer to the original model compared to full-precision smaller models within the same series.

4.2 System Efficiency

We build a prototype model consistency monitoring system based on FlexGen (Sheng et al., 2023), enabling it to run on consumer-grade devices. We test its amortized sequence processing speed to demonstrate the feasibility of monitoring model consistency on the user side.

Settings. The LLM running on the system is OPT-66B (FP16) (Zhang et al., 2022), and the hardware specifications of the device used for running the prototype are shown in Table 2. The SSD has a read bandwidth of about 3.0 GB/s and a write bandwidth of about 1.2 GB/s.

It is important to note that, even though our CPU memory is sufficient to store the LLM used, we actually place the whole models on disk to better simulate real-world user devices. Model parameters are only loaded into the GPU when required for computation. Furthermore, to account for the varying memory capacities of consumer-grade GPUs, we limit the GPU memory available to the system by PyTorch command. For each set of experiments, we randomly generated 100 sequences, each with a length of 1024 tokens. Multiple sequences were processed simultaneously according to the maximum batch size we can apply corresponding to the specific memory limit, and we recorded the total time required to process all sequences. The ratio of this time to the total number of sequences was used as an efficiency metric for the system (measured in seconds per sequence).

GPU Memory Limit	Maximum Batch Size	Speed (seconds/seq)
4GB	2	9.64
6GB	8	4.08
8GB	12	3.63
10GB	16	3.69
12GB	20	3.65

Table 3: Speed performance under different GPU memory limits and the corresponding maximum batch size.

Results analysis. The results are shown in Table 3. As observed, even with a memory usage limit of just 4 GB, it is possible to perform consistency monitoring for a model with up to 66B parameters at a relatively slow speed. As the memory limit increases, the maximum batch size also increases, resulting in improved efficiency. The efficiency saturates when the batch size reaches 12, with a memory usage of only 8 GB.

These results demonstrate that, with offloading techniques, consumer-grade GPUs are fully capable of monitoring consistency of models that far exceed their GPU memory capacity.

5 Discussion and Future Work

In this section, we discuss how to extend the proposed method in Section 3.2 to the stochastic sampling scenario.

The first approach requires the user to provide a *deterministic sampling rule*. For instance, at each token position i, the rule specifies selecting the token with the k_i-th highest probability. This sampling rule can be randomly generated at first and supplemented with a minimum probability threshold to prevent the selection of tokens with extremely low probabilities. When computing the logits difference, the maximum logits term in Equation 1 is replaced with the k_i-th highest logits. This approach is formally equivalent to the method used for greedy sampling, and thus it should exhibit the same effectiveness.

A more practical approach allows the user to define the sampling parameters (e.g., top-k, top-p, temperature, etc.) in the usual manner. In this case, the service provider performs stochastic sampling based on these parameters while concurrently recording the token with the highest sampling probability at each iteration. The sequence formed by

these tokens is termed the *greedy sequence*, and the service provider ultimately returns both the sampled sequence and the corresponding greedy sequence to the user. The user then computes the logits from the sampled sequence and uses the greedy sequence to calculate the logits difference in Equation 1. Given that the time cost of the sampling phase is negligible relative to the overall generation process, this approach imposes minimal additional burden on the service provider. Moreover, because our method monitors that the greedy sequence originates from the user-specified model, there is no incentive to additionally use a degraded model for generating the sampled sequence.

Additionally, most service providers support returning the log probability for each generated token, which is derived from the logits. Based on this, it would be interesting to explore further feasible methods and perform empirical validations of the above approaches.

6 Conclusion

To address the trust issue between users and open-source LLM inference service providers, we propose a paradigm for monitoring model consistency on the user side. To this end, we designed a logits-based metric to differentiate sequences generated by degraded models. In addition, we implement an offloading-based system prototype to demonstrate the feasibility of the proposed paradigm.

7 Limitations

More general methods for stochastic sampling. Even though Section 5 discusses several ways for extending the proposed method to the stochastic sampling scenario, each of these methods requires additional information provided by either the user or the service provider. Considering the need for easy deployment, more general approaches should rely solely on the user's stochastic sampling parameters and the sequence returned by the service provider. This represents a challenging direction for future research because stochastic sampling inherently involves a trade-off between text quality and diversity, and the generated sequence will be increasingly difficult to distinguish from a sequence produced by a strong degraded model.

Model Degrading Approaches. In this work, we only discuss the two most straightforward model degrading approaches: size degrading and precision degrading. Other degrading strategies, such as knowledge distillation (Xu et al., 2024) for relatively small models using user-specified models, and structured pruning (Wang et al., 2021; Hubara et al., 2021; Frantar and Alistarh, 2023) of user-specified models, could also effectively increase the service provider's margin profits.

Implementation. The prototype system implemented in this work currently supports only the OPT model series. Although this is sufficient for testing system efficiency, further implementation for more LLMs series is necessary before the system can be deployed in real-world applications.

Acknowledgement

This work is supported by Tsinghua University Dushi Program and Shanghai Qi Zhi Institute Innovation Program SQZ202312.

References

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 2022. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE.

Anthropic. 2025. Introducing Claude 4. https://www.anthropic.com/news/claude-4. Accessed on: May 30, 2025.

Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. 2024. Zkml: An optimizing system for ml inference in zero-knowledge proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 560–574.

KD Conway, Cathie So, Xiaohang Yu, and Kartin Wong. 2024. opml: Optimistic machine learning on blockchain. *arXiv preprint arXiv:2401.17555*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Sebastian Gehrmann, Hendrik Strobelt, and Alexander M Rush. 2019. Gltr: Statistical detection and visualization of generated text. *arXiv preprint arXiv:1906.04043*.

Georgi Gerganov. 2025. ggerganov/llama.cpp: LLM inference in C/C++. https://github.com/ggerganov/llama.cpp.

Groq. 2025. Overview - GroqDocs. https://console.groq.com/docs/overview. Accessed on: May 30, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Itay Hubara, Brian Chmiel, Moshe Island, Ron Banner, Joseph Naor, and Daniel Soudry. 2021. Accelerated sparse neural training: A provable and efficient method to find n: m transposable masks. *Advances in neural information processing systems*, 34:21099–21111.

Daphne Ippolito, Daniel Duckworth, Chris Callison-Burch, and Douglas Eck. 2019. Automatic detection of generated text is easiest when humans are fooled. *arXiv preprint arXiv:1911.00650*.

Zhen Lin, Shubhendu Trivedi, and Jimeng Sun. 2024. Contextualized sequence likelihood: Enhanced confidence scores for natural language generation. *arXiv preprint arXiv:2406.01806*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. 2024. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 932–949.

Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D Manning, and Chelsea Finn. 2023. Detectgpt: Zero-shot machine-generated text detection using probability curvature. In *International Conference on Machine Learning*, pages 24950–24962. PMLR.

OpenAI. 2025. ChatGPT: Get answers. Find inspiration. Be more productive. https://openai.com/chatgpt/overview/. Accessed on: May 30, 2025.

OpenRouter. 2025. OpenRouter Quickstart Guide. https://openrouter.ai/docs/quickstart. Accessed on: May 30, 2025.

Jie Ren, Yao Zhao, Tu Vu, Peter J Liu, and Balaji Lakshminarayanan. 2023. Self-evaluation improves selective generation in large language models. In *Proceedings on*, pages 49–64. PMLR.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR.

Cathie So, KD Conway, Xiaohang Yu, Suning Yao, and Kartin Wong. 2024. opp/ai: Optimistic privacy-preserving ai on blockchain. *arXiv preprint arXiv:2402.15006*.

Irene Solaiman, Miles Brundage, Jack Clark, Amanda Askell, Ariel Herbert-Voss, Jeff Wu, Alec Radford, Gretchen Krueger, Jong Wook Kim, Sarah Kreps, et al. 2019. Release strategies and the social impacts of language models. *arXiv preprint arXiv:1908.09203*.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024. Powerinfer: Fast large language model serving with a consumer-grade gpu. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 590–606.

Megha Srivastava, Simran Arora, and Dan Boneh. 2024. Optimistic verifiable training by controlling hardware nondeterminism. *arXiv preprint arXiv:2403.09603*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

together.ai. 2025. Introduction to Together AI and all its services. https://docs.together.ai/docs/introduction. Accessed on: May 30, 2025.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Hanrui Wang, Zhekai Zhang, and Song Han. 2021. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 97–110. IEEE.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR.

Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. 2024. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Zihan Zheng, Peichen Xie, Xian Zhang, Shuo Chen, Yang Chen, Xiaobing Guo, Guangzhong Sun, Guangyu Sun, and Lidong Zhou. 2021. Agatha: Smart contract for dnn computation. *arXiv preprint arXiv:2105.04919*.

A Discussions and results on Previous Logits-based Metrics

In this section, we review previously proposed metrics that rely on the logits outputs of sequences from language models, and we discuss why these metrics are not suitable for the problem in this paper, accompanied by experimental results.

First, we discuss metrics related to the self-evaluation of language models (Lin et al., 2024; Ren et al., 2023). In this task, the goal is to assess the "confidence" of a language model in its own generated sequences, thereby allowing users to roughly gauge how "proficient" the model is at the current generation task, rather than being misled by outputs that lack credibility. Consequently, metrics for self-evaluation are highly dependent on the prompt provided by the user, making them naturally unsuitable for monitoring model consistency. We believe that further experimental validation on this point is unnecessary.

However, another popular topic, detection of machine-generated text, appears at first glance to be quite similar to the problem in this paper, and there have been many logits-based methods proposed in that field (Gehrmann et al., 2019; Solaiman et al., 2019; Ippolito et al., 2019; Mitchell et al., 2023). Thus, it is necessary to experiment with these metrics in the context of model consistency monitoring.

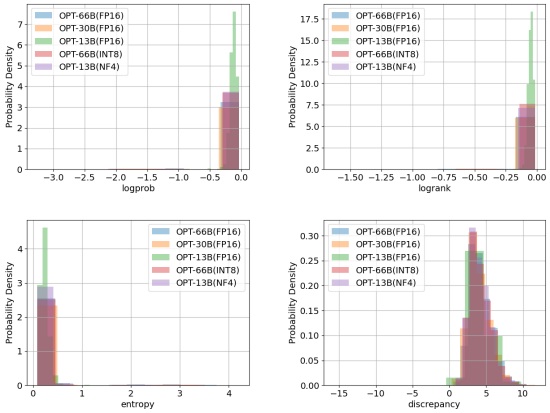

Figure 8: Four metrics' distributions of OPT-66B and its degraded models.

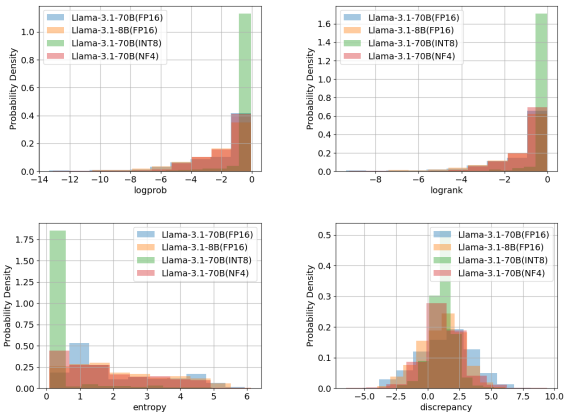

Figure 9: Four metrics' distributions of Llama-3.1-70B and its degraded models.

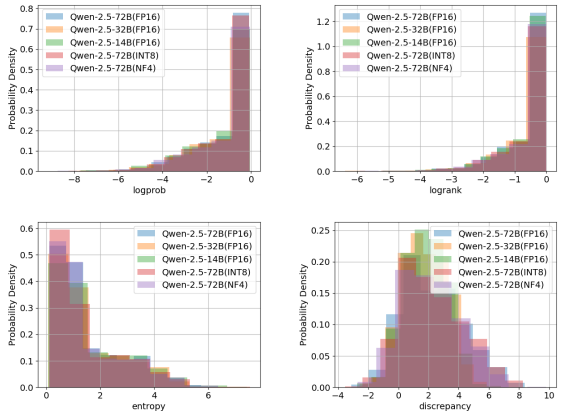

Figure 10: Four metrics' distributions of Qwen-2.5-72B and its degraded models.

We selected four representative metrics: log probability (logprob), log token ranks (logrank), predictive entropy (entropy), and discrepancy between samples (discrepancy). Using the user-specified model as the reference model, we con-

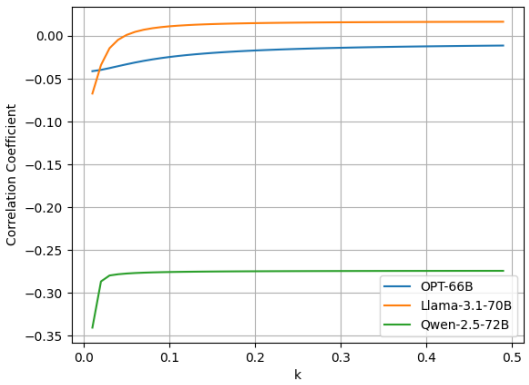

Figure 11: The correlation coefficients between sequence length and sequence confidence under three LLMs as k varies.

ducted experiments under model degradation settings as described in Section 4.1 on A100 80GB GPUs. The experimental results for three LLMs series are shown in Figure 8, Figure 9, and Figure 10. In each figure, a subplot presents the distributions of a particular metric computed on sequences generated by the reference model on its own and those generated by its degraded versions; each subplot corresponds to one metric.

It is evident that these metrics do not clearly distinguish between the outputs of the reference model and the degraded models. We offer the following explanation: the key to detecting machine-generated text lies in identifying the differences between human-authored and model-generated texts, rather than the differences between outputs from different models. In fact, in the detection task, it is actually preferable for the metric distributions across different models to be consistent, as this indicates that the proposed method only requires a sufficiently strong reference model to detect sequences generated by various models. This also leads to the observation that the focal points of the detection problem and the issue discussed in this work are fundamentally different.

B Hyperparameter Study

In this section, we discuss the impact of the two hyperparameters introduced in Algorithm 1 on the resulted sequence confidence distributions and their appropriate values. All experiments in this section were conducted on A100 80GB GPUs.

First, we investigate the appropriate value for the coefficient of the positional correction term (k) in Equation 2. The purpose of introducing this term is to mitigate the impact of floating-point computation errors that accumulate with increasing sequence length. To this end, we only use the user-specified models to generate sequences and analyze the confidence distribution of itself. We then calculate the Pearson correlation coefficient between the confidence of all sequences and their token lengths and observe how this correlation changes with different values of k.

The results for the three LLMs (OPT-66B, LLaMA-3.1-70B, and Qwen2.5-72B) are shown in Figure 11. As observed, the actual value of correlation coefficients varies across models, likely due to intrinsic structural differences between them. However, a general trend can be obtained: once the value of k reaches a certain threshold, the correlation coefficient stabilizes near a relatively small value, and further increases in k have minimal impact on the correlation. Based on the observation, we select $k = 0.05$ as the value used in Section 4.1.

Then, we investigated the effect of varying the length of the consecutive subsequence window (W) on the sequence confidence distribution. Results for three LLMs series are shown in Figure 12, Figure 13, and Figure 14. For each user-specified model (the FP16 versions of OPT-66B, LLaMA-3.1-70B, and Qwen2.5-72B), we compared the sequence confidence distributions of itself and all its degraded versions discussed in Section 4.1. These distributions are visualized in individual subplots, each applying a different value of W: focusing on individual tokens ($W = 1$), shorter subsequences ($W = 10$), and longer subsequences ($W = 100$).

The results reveal that, across all three LLMs series, the difference in sequence confidence distributions between the user-specified model and the degraded models becomes more obvious as W increases. However, the differences are less pronounced for shorter subsequence sizes. Considering the generalizability to sequence lengths, we selected a shorter subsequence size ($W = 10$) for the experiments in Section 4.1.

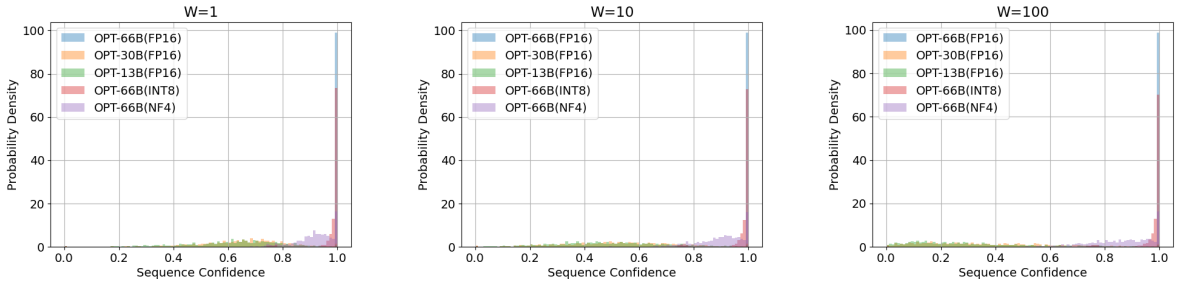

Figure 12: Sequence confidence distributions of OPT-66B and its degraded models under three sizes of consecutive subsequence windows.

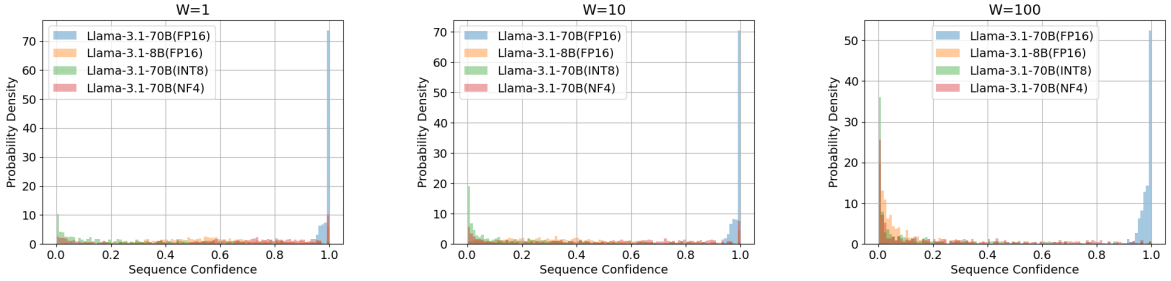

Figure 13: Sequence confidence distributions of Llama-3.1-70B and its degraded models under three sizes of consecutive subsequence windows.

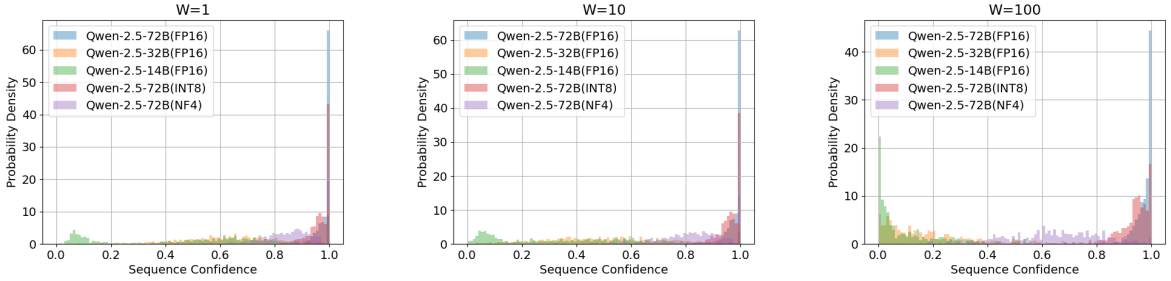

Figure 14: Sequence confidence distributions of Qwen-2.5-72B and its degraded models under three length of consecutive subsequence windows.