

End-to-End Benchmarking of Cloud and Self-Hosted AI in a Ukrainian-Language Voice Nutrition-Tracking Pipeline*

Larysa Katerynych^{1,*†}, Maksym Kubichka^{1†}, Kostiantyn Zhreb^{1,†}

¹ Taras Shevchenko National University of Kyiv, 60 Volodymyrska St., Kyiv, 01033, Ukraine

Abstract

Organizations constrained by privacy, security, or regulation often need to replace cloud AI services with self-hosted models, yet the end-to-end consequences of such migration for real voice applications remain insufficiently studied. This paper presents an end-to-end benchmark of the voice-based nutrition-tracking pipeline of Calories365, in which Ukrainian-language voice messages are processed by automatic speech recognition (ASR) and one or two large language model (LLM) stages for structured extraction of food products and quantities and, when needed, generation of missing nutritional values. A cloud baseline (GPT-4o + Cloud Whisper) is compared with self-hosted configurations based on Qwen3 (4B, 8B, 14B, and 30B) and Faster-Whisper, integrated through an OpenAI-compatible proxy that leaves the original application logic unchanged. Using 28 fixed audio recordings and six concurrency levels (1–28), 1,008 requests were benchmarked for end-to-end latency, stage-level latency, throughput, 95th-percentile latency, transcription accuracy, product extraction quality, and quantity assignment quality. The results show that self-hosting is feasible and, in several configurations, faster while maintaining application-level quality. Faster-Whisper reduced word and character error rates from 8.5% and 3.6% to 7.1% and 3.1%, respectively. At concurrency 1, Qwen3-4B on RTX 4090 and Qwen3-30B on A100 achieved 2.1–2.2 s average latency versus 3.9 s for the cloud baseline; at concurrency 28, Qwen3-4B reached about 7 requests/s versus about 6 requests/s in the cloud. All Qwen3 configurations achieved 100% product extraction recall, and Qwen3-14B/Qwen3-30B remained within 2% of GPT-4o in quantity assignment. Overall, OpenAI-compatible self-hosting is a practical option for Ukrainian-language voice applications, although the LLM stage becomes the main bottleneck and tail latency grows more sharply under load than in the cloud baseline.

Keywords

Calories365, self-hosted AI, end-to-end benchmarking, voice pipeline, automatic speech recognition, large language models, Ukrainian language, calorie tracking

1. Introduction

Voice interfaces are increasingly built as a chain of several AI models of different types—speech recognition, text generation, and classification—where the output of one stage becomes the input to the next. This paper investigates the performance of one such chain using the Calories365 application as a case study: a voice message first passes through an ASR model for transcription and then through one or two LLM stages for extracting products and quantities in a structured format and, when necessary, for generating missing data. Calories365 itself was previously described as a web-based calorie-tracking system with multilingual voice input and OpenAI API integration [1]. The subject of this study is the behavior of the end-to-end pipeline of a real application when its AI components are replaced.

In practice, many organizations cannot transmit data to third-party cloud services because of regulatory requirements, internal security policies, or industry-specific constraints. For them, one of the practically significant options is to replace cloud-based AI components with self-hosted counterparts. If the application code has already been built around cloud services, such a replacement

2nd Workshop "Intelligent information technologies" UkrProg-IIT'2026 co-located with 16th International Scientific and Practical Programming Conference UkrPROG'2026, May 5-6, 2026, Kyiv, Ukraine

* Corresponding author.

† These authors contributed equally.

✉ katerynych@gmail.com (L. Katerynych); maxim.kubichka@gmail.com (M. Kubichka); zhreb@gmail.com (K. Zhreb)

ORCID 0000-0001-7837-764X (L. Katerynych); 0009-0003-6372-5869 (M. Kubichka); 0000-0003-0881-2284 (K. Zhreb)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

should ideally be performed without changing the application logic. In this case, it is important to evaluate the behavior of the entire system end to end: how latency changes, how the pipeline scales under parallel load, and which stage becomes the bottleneck.

Existing studies generally focus either on model quality measured by standard benchmarks or on the throughput of individual inference servers. The full ASR $\rightarrow$ LLM $\rightarrow$ structured data extraction chain under production-like conditions remains underexplored. It is also worth noting that, for Ukrainian-language voice input, the number of reproducible experiments of this kind remains limited, although linguistic features and the format of application data directly affect the behavior of each stage of the pipeline.

The purpose of this paper is to measure the end-to-end performance of migrating AI components from a cloud configuration to a self-hosted configuration within a real application. The migration is carried out through an OpenAI-compatible interface, which makes it possible to substitute the provider without changing the application logic. For the experiment, a benchmarking tool was implemented in Laravel, where the AI provider configuration is changed without modifying the application code, Ukrainian-language audio recordings are used as input data, and the results of each request are recorded in CSV files. The testing covers different levels of parallel load; for each level, total latency, the latency of individual stages (ASR, LLM analysis, LLM generation), the 95th percentile latency, and throughput are measured. The results are presented as a direct comparison of the cloud and self-hosted configurations on the same pipeline with a fixed set of Ukrainian-language input data.

The remainder of the paper reviews the existing literature and open questions, describes the benchmark architecture and experimental setup, presents the results with discussion, and concludes with the limitations and findings.

2. Review of Recent Studies and Publications and Unresolved Aspects of the Problem

Existing Ukrainian-language studies are primarily focused on evaluating individual components and tasks and, so far, do not cover end-to-end measurement of application systems. Within the UNLP workshop series, several Ukrainian-centered benchmarks have been developed. ZNO-Vision (UNLP 2025) offers more than 4,300 expert-prepared questions across 12 disciplines and highlights the lack of benchmarks for low-resource languages [2]. Eval-UA-tion (UNLP 2024) provides a set of Ukrainian-language tasks for evaluating language models at the text level [3]. UAlign (UNLP 2025) introduces a benchmark for evaluating the alignment of LLM behavior with user expectations in Ukrainian and identifies a gap relative to English [4]. Makhortykh et al. empirically show that the quality of LLM chatbot responses to Ukrainian-language queries is substantially lower than to English-language queries [5]. All of these studies remain at the level of text-based evaluation of language models and do not include measurement of an application pipeline.

For Ukrainian-language ASR, the study by Kobylukh et al. compares the accuracy of speech-to-text APIs for Ukrainian [6], while study [7] compares ASR systems for Ukrainian and English using transcription error and processing time metrics. These works confirm the existence of component-level tests; however, they do not cover the subsequent LLM stages and do not establish a protocol for measuring the entire chain.

In recent years, the first attempts to evaluate multi-stage AI pipelines as integrated systems have begun to appear. Malleni et al. implement an ASR (Whisper) $\rightarrow$ LLM (summarization) chain in Kubernetes and measure performance in batch and online modes [8]. Mairittha et al. analyze modular voice pipelines and show that temporal misalignment between stages and limited real-time error correction capabilities are structural consequences of modular architecture [9].

A separate line of research concerns the reliability of structured output produced by large language models. PICARD demonstrates incremental parsing as a way to obtain valid structures [10], while Lu et al. show that even modern LLMs often generate invalid JSON for complex schemas [11]. These studies examine generation reliability from the perspective of format correctness; however,

they do not cover scenarios in which additional LLM calls are triggered by application logic. The impact of such additional calls on latency, tail latency, and throughput of a multi-stage pipeline under load remains unmeasured.

Thus, the existing literature covers individual components of speech processing systems and language models, yet it does not establish a methodology for the reproducible measurement of complete application pipelines.

Based on the conducted review, the following unresolved aspects of the problem have been identified. First, there is a lack of reproducible Ukrainian-language experiments in which voice input passes through the full application chain ASR → LLM for structured data extraction → an additional LLM call for generating missing product attributes, with performance measured under load [2, 6, 8]. Second, Ukrainian-language materials are predominantly component-level in nature and do not provide a comparison of cloud and self-hosted configurations of an application pipeline with a latency breakdown by stage [3, 7]. Third, studies on multi-stage pipelines and structured output do not define a Ukrainian-language application protocol for structured data extraction in a real application and do not show how average latency, 95th percentile latency, and throughput change when cloud components are replaced with self-hosted configurations [8–11].

3. Purpose and Problem Statement

The purpose of this paper is to measure the end-to-end performance of the voice pipeline of the Calories365 application when cloud-based AI components are replaced with self-hosted configurations and to compare the behavior of the cloud and self-hosted configurations on the same application chain using a fixed set of Ukrainian-language input data.

To achieve this purpose, the following objectives were formulated:

1. To implement a benchmarking environment within the existing Laravel application in which the AI provider configuration is changed without modifying the application code through an OpenAI-compatible interface.
2. To prepare a fixed Ukrainian-language dataset of 28 audio recordings in .ogg format containing meal descriptions, used identically for all configurations.
3. To conduct a series of experiments for the cloud baseline configuration (GPT-4o + Cloud Whisper) and self-hosted configurations (Qwen3 models of different sizes + Faster-Whisper) on different hardware platforms (RTX 4090, RTX 6000 Ada, A100) under different levels of parallel load ($c=1$, $c=2$, $c=4$, $c=8$, $c=16$, $c=28$).
4. To measure, for each run, total latency, the latency of individual stages (ASR, LLM analysis, LLM generation), the 95th percentile latency, and throughput.
5. Based on the obtained data, to evaluate the quality of pipeline operation for each configuration, identify the stage that becomes the bottleneck as the load increases, and determine how tail latency changes when transitioning from the cloud configuration to a self-hosted configuration.

4. E2E Benchmark and Experimental Setup

4.1. System Under Test

Calories365 is a calorie diary application. Within the scope of this study, its key application scenario is examined: nutrition tracking through voice input in a messenger environment. In this scenario, the user sends a voice message describing consumed food, after which speech transcription, product recognition, and the return of information about caloric and macronutrient values are performed.

From a technical perspective, the message passes through the following processing chain: the incoming messenger webhook is received by the server-side part of the application, where the request is placed into a Redis queue and processed by a background worker. The audio is then converted from .ogg to MP3 using FFmpeg, after which it is sent to the ASR service for transcription.

The resulting text is passed to the LLM for extracting products and quantities in a structured format during the analyze stage. When necessary, an additional LLM call is performed during the generate stage, in which caloric and macronutrient values are generated for products missing from the database. After all stages are completed, the user receives the final text result. The subject of measurement is the complete application processing chain—from the receipt of a voice message to the generation of the final response.

4.2. Benchmarking Environment

To run the pipeline without real interaction with the messenger and without network latency during audio download, a benchmarking wrapper was implemented. In benchmark mode, the Laravel container replaces two components through the dependency substitution mechanism: the audio download service reads files from the local disk instead of the external messenger platform, and the message sending service stores responses in memory instead of performing real HTTP calls. Audio conversion with FFmpeg, as well as ASR and LLM calls, remains fully real; only the messenger interaction layer is substituted.

Each processed request produces a row in a CSV file containing total latency, the latency of individual stages (ASR, LLM analyze, LLM generate), the number of input and output tokens for each LLM call, the reference transcription, the actual ASR transcription, the recognized products, and the final text responses.

4.3. Testing Configurations

The experiment compares one cloud configuration and five self-hosted configurations. The cloud baseline configuration consists of GPT-4o and Cloud Whisper, accessed through the OpenAI API. The self-hosted configurations are built on Qwen3 family models (4B, 8B, 14B, 30B), deployed through vLLM (v0.15.1) on rented GPU servers of the vast.ai platform, in combination with Faster-Whisper based on the deepdml/faster-whisper-large-v3-turbo-ct2 model.

The following hardware was used for the self-hosted configurations: RTX 4090 for the Qwen3-4B and Qwen3-8B models, RTX 6000 Ada and A100 80GB for Qwen3-14B, and A100 80GB for Qwen3-30B. In the cloud configuration, the Laravel application communicates directly with the OpenAI API. In the self-hosted configurations, requests pass through an OpenAI-compatible FastAPI proxy that adapts the request format to locally deployed models; details of its operation are provided in the technology selection section.

The ASR part of the self-hosted configuration is deployed separately from the LLM part. Faster-Whisper is run on a dedicated server with two GPUs ($2 \times$ RTX 3090), independently of the server on which the LLM models are executed. Five worker processes are launched on each GPU, each representing a separate copy of the model, for a total of ten parallel ASR processes. Requests are distributed among them by the proxy layer using a round-robin scheme. This separation eliminates competition between ASR and LLM for GPU resources within a single configuration. The LLM and ASR configurations used in the experiment are presented in Table 1.

Table 1

LLM and ASR configurations used in the experiment

Component	Model	Hardware
LLM	GPT-4o	undisclosed
ASR	Cloud Whisper	undisclosed
LLM	Qwen3-4B	RTX 4090
LLM	Qwen3-8B	RTX 4090
LLM	Qwen3-14B	RTX 6000 Ada
LLM	Qwen3-14B	A100 80GB
LLM	Qwen3-30B	A100 80GB

4.4. Dataset

The dataset consists of 28 Ukrainian-language audio recordings in .ogg format, generated using ElevenLabs in a single voice style. The set covers different levels of complexity: simple messages with 1–2 products (“200 grams of cottage cheese and a banana”), medium-complexity messages with 3–4 products and quantities (“250 g of borscht with sour cream, 60 g of bread, and half a cutlet”), more complex formulations with several items (“pasta carbonara, three slices of pizza, and tiramisu”), as well as messages with quantitative nuances (“6 nuggets”, “14 varenyky”). For each file, a reference transcription was recorded, which makes it possible to evaluate ASR accuracy and the correctness of product extraction.

4.5. Load Parameters

For each configuration, runs were performed at six levels of parallelism: 1, 2, 4, 8, 16, and 28 simultaneous requests. At each level of parallelism, the corresponding number of tasks was submitted simultaneously to the Redis queue, after which they were processed in parallel by background workers. Before each run, 3 warm-up requests were executed to establish and stabilize connections with the ASR and LLM servers. The timeout for a single request was 180 seconds. In total, 1,008 measurements were performed (6 configurations × 6 levels of parallelism × 28 files).

4.6. Metrics

For each request, four time-based metrics are recorded: total latency, that is, the time from task submission to the queue until the result is received; ASR latency; LLM latency at the analyze stage; and LLM latency at the generate stage. For the ASR, analyze, and generate stages, these values correspond to the wall-clock time of the respective calls, whereas total latency characterizes the full time required to pass through the application chain.

To formalize stage-level timing, each processed request was represented as a sequence of pipeline stages. The end-to-end latency of request i was decomposed according to Eq. (1).

$$T_i^{\text{e2e}} = T_i^{\text{asr}} + T_i^{\text{analyze}} + I_i^{\text{gen}} T_i^{\text{generate}} + T_i^{\text{overhead}} \quad (1)$$

where the first term denotes ASR latency, the second term denotes the latency of the LLM analyze stage, the third term represents the optional LLM generate stage, and the last term captures queueing, dispatching, proxy processing, and other application-level overhead. The generation indicator is equal to 1 when the generate stage is executed and 0 when this stage is skipped.

For aggregated comparison across model configurations and concurrency levels, the average end-to-end latency was calculated according to Eq. (2).

$$\bar{T}_{m,c}^{\text{e2e}} = \frac{1}{N} \sum_{i=1}^N T_{i,m,c}^{\text{e2e}} \quad (2)$$

where m denotes the tested model configuration, c denotes the concurrency level, and $N = 28$ is the number of audio files in the dataset.

In addition, the number of input and output tokens is recorded for both LLM stages, which makes it possible to interpret differences in latency across configurations. Based on the raw data, derived metrics are calculated: throughput, that is, the number of processed requests per second at a given level of parallelism, and the 95th percentile latency (p95), which characterizes the tail of the latency distribution.

Quality is evaluated by comparing the reference and actual transcriptions to determine ASR accuracy, as well as by analyzing the correctness of product recognition and quantity assignment.

4.7. Experimental Limitations

For each configuration $\times$ parallelism level point, 28 requests were used, corresponding to the 28 files in the dataset. With this sample size, the p95 value effectively corresponds to the 27th value in the ordered sequence of 28 observations. This is sufficient for comparative conclusions within the studied load conditions; however, it is insufficient for rigorous statistical generalization.

The self-hosted configurations were deployed on rented servers of the vast.ai platform, so all measured latencies include the network component to the corresponding instance. Similarly, in the cloud baseline configuration, the latencies include the network component to the OpenAI API. Therefore, the measured values reflect the full application latency under remote access to the services rather than only the computational latency on the GPU.

5. Technology and Infrastructure Selection

5.1. Computational Platform

For the deployment of the self-hosted part of the experiment, vast.ai, a GPU server rental platform, was used. Within the scope of this study, it is treated as a remote computing infrastructure that is similar in its mode of use to an organization’s server node: ASR and LLM services are deployed on dedicated GPUs, while the application server accesses them over the network, as in a typical microservice architecture.

In the studied setup, the difference is that rented instances are used instead of owned physical hardware. The network component of access to these instances is included in the measurement results and taken into account in the experimental limitations.

5.2. Qwen3 Model Family

The Qwen3 family was selected for the self-hosted LLM part of the experiment for several reasons. First, Qwen3 supports 119 languages and dialects and was trained on approximately 36 trillion tokens, which provides broad language coverage, including low-resource languages such as Ukrainian [12]. Second, in the work of Grozea and Verbitsky (WMT 2025), Qwen3 is directly evaluated on the EN $\rightarrow$ UK pair, and the paper also presents an analysis of typical errors in Ukrainian-language processing, which indicates the relevance of this model family for Ukrainian-language application tasks [13].

Third, the Qwen3 family covers a wide range of open-weight models released under the Apache 2.0 license: from compact dense models to larger configurations, including MoE architectures. MoE (Mixture of Experts) is an architecture in which only a subset of the model’s parameters is activated for processing each token, while the remaining parameters are inactive during that step. This made it possible to use several variants of different scales within a single experiment—Qwen3-4B, Qwen3-8B, Qwen3-14B, and Qwen3-30B—and to trace how model choice affects the behavior of the full application pipeline under the same processing scheme. This is of particular importance for Qwen3-30B, since this model belongs to the MoE class and uses approximately 3B active parameters per token.

To formalize this architectural difference, the active-parameter ratio was defined according to Eq. (3).

$$\rho_m = \frac{p_m^{\text{active}}}{p_m^{\text{total}}} \quad (3)$$

where total parameters denote the full size of model m , while active parameters denote the subset of parameters used during one-token processing. For dense models, the number of active parameters equals the total number of parameters. For the MoE-based Qwen3-30B configuration, approximately 3B parameters are activated per token, so the active-parameter ratio is approximately $3\text{B} / 30\text{B} = 0.1$.

Based on the active-parameter ratio, the relative LLM computational load was expressed through the number of active parameters and the number of processed tokens, as shown in Eq. (4).

$$C_{i,m}^{\text{llm}} \propto P_m^{\text{active}} \left((n_{i,a}^{\text{in}} + n_{i,a}^{\text{out}}) + I_i^{\text{gen}} (n_{i,g}^{\text{in}} + n_{i,g}^{\text{out}}) \right) \quad (4)$$

where the first token-count term corresponds to the analyze stage, the second token-count term corresponds to the optional generate stage, and the generation indicator is equal to 1 when the generate stage is executed and 0 when this stage is skipped. This formulation links the expected LLM inference load to both model architecture and request-level token volume.

Fourth, even the compact models of the family demonstrate competitive quality: according to the technical report, Qwen3-4B achieves a performance level comparable to Qwen2.5-72B-Instruct [12], which makes this family attractive for scenarios with limited GPU resources.

5.3. Faster-Whisper as an ASR Server

Faster-Whisper was selected for transcription as an implementation of Whisper based on CTranslate2, which provides substantially higher processing speed with lower GPU memory consumption compared with the original OpenAI implementation [17]. The model used in this study is deepdml/faster-whisper-large-v3-turbo-ct2. Like vLLM, Faster-Whisper provides an OpenAI-compatible transcription interface, which allows the application code to access the self-hosted ASR through the same interface that is used for Cloud Whisper.

5.4. OpenAI-Compatible Proxy

Despite the availability of OpenAI-compatible interfaces in vLLM and Faster-Whisper, formal compatibility of requests and responses does not ensure full behavioral compatibility with cloud services. Differences between cloud and self-hosted models remain and are not eliminated by simply substituting the service address. In particular, Qwen3 returns `<think>...</think>` blocks that are not expected by the application code; a prompt designed for GPT-4o requires restructuring for effective operation with self-hosted models; and the parameters of the structured output schema also differ across models.

For this reason, a FastAPI proxy was deployed between the Laravel application and the self-hosted servers, performing prompt adaptation, removal of reasoning control tokens, and centralized management of the structured output schema while keeping the Laravel application code unchanged. This approach corresponds to the substitutable provider pattern described in the documentation of vLLM [14] and Hugging Face TGI [18]: the application code interacts with the same OpenAI-compatible interface regardless of whether a cloud service or a self-hosted model is used.

6. Results and Discussion

6.1. Speech Recognition Quality (ASR)

Since all self-hosted configurations use the same ASR server—Faster-Whisper large-v3-turbo on dedicated GPUs—the quality of transcription is compared between two ASR configurations: Cloud Whisper (OpenAI) and self-hosted Faster-Whisper.

The obtained results show that self-hosted Faster-Whisper achieves better transcription performance than Cloud Whisper. The global Word Error Rate (WER) is 8.5% for Cloud Whisper and 7.1% for Faster-Whisper, while the Character Error Rate (CER) is 3.6% and 3.1%, respectively (Fig. 1).

Within the studied scenario, both ASR configurations provide sufficient accuracy for subsequent processing, since typical transcription errors are mostly limited to writing numerals as words (“three” instead of “3”) or to minor inflectional differences that do not change the meaning of the message and do not hinder subsequent product extraction at the LLM stage. Therefore, the transition from the cloud ASR configuration to the self-hosted ASR configuration is not accompanied by a loss

of transcription quality; within the conducted experiment, a moderate improvement was observed. The corresponding visualization is shown in Fig. 1.

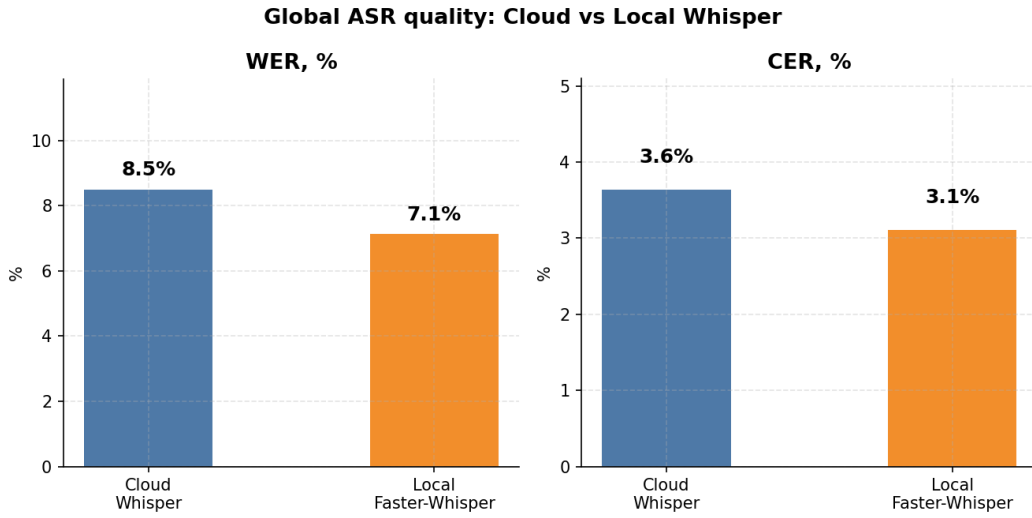


Figure 1: Comparison of ASR Models.

6.2. Performance: Latency and Throughput

The overall performance picture is best represented by the heatmap of average total latency calculated according to Eq. (2) and shown in Fig. 2. Under minimal load ($c=1$), the fastest configurations are Qwen3-30B on A100 (2.1 s) and Qwen3-4B on RTX 4090 (2.2 s); both configurations clearly outperform the cloud GPT-4o configuration (3.9 s). For Qwen3-30B, this result is consistent with the active-parameter ratio defined in Eq. (3): despite having 30B total parameters, only about 3B parameters are activated for each token, which reduces the relative LLM computational load described in Eq. (4). Qwen3-8B on RTX 4090 shows 3.0 s, which also outperforms the cloud configuration, although it trails Qwen3-4B because of the larger computational volume. The slowest configuration is Qwen3-14B on RTX 6000 Ada, for which the average latency at $c=1$ is 5.1 s.

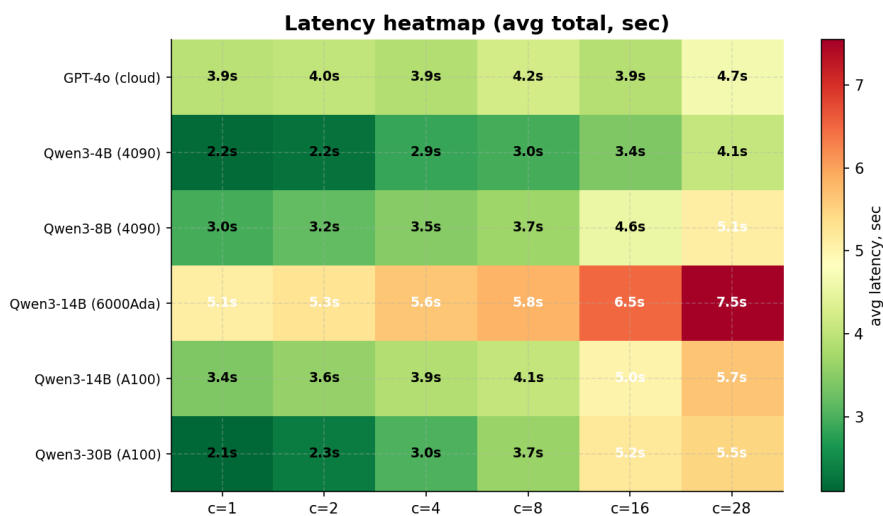


Figure 2: Latency heat map.

As parallelism increases, latency rises in all self-hosted configurations because requests compete for GPU resources. The cloud GPT-4o configuration remains the most stable: its average latency

changes only slightly in the range from $c=1$ to $c=16$ (3.9–4.2 s) and reaches 4.7 s only at $c=28$. This behavior is consistent with the nature of a cloud service. The self-hosted configurations scale differently: for Qwen3-4B, latency increases moderately, from 2.2 to 4.1 s at $c=28$, whereas for Qwen3-14B on RTX 6000 Ada the increase is much more pronounced, from 5.1 to 7.5 s. For the same Qwen3-14B on A100, latency changes from 3.4 to 5.7 s, which shows the substantial impact of GPU class on the behavior of larger models under load.

The latency breakdown by stage at $c=1$ (Fig. 3) follows the decomposition defined in Eq. (1) and clarifies the sources of these differences. For GPT-4o, the largest share comes from the Whisper stage (about 1.3 s), since the cloud ASR configuration is invoked as a separate service. In the self-hosted configurations, ASR latency remains consistently lower, at about 0.5 s, due to the dedicated ASR server. The dominant stage for the self-hosted configurations is the LLM analyze stage, that is, the transformation of user text into a structured list of products. This stage determines the main difference between configurations: for Qwen3-4B, its latency is about 1.2 s, while for Qwen3-14B on RTX 6000 Ada it exceeds 3 s. The generate stage is activated only for some requests and usually adds about 0.3–0.7 s. The share of overhead, which includes the queue, dispatching, and proxy layer, remains relatively small, at about 0.2–0.4 s. The main contribution to the difference between configurations therefore comes from the ASR and LLM stages.

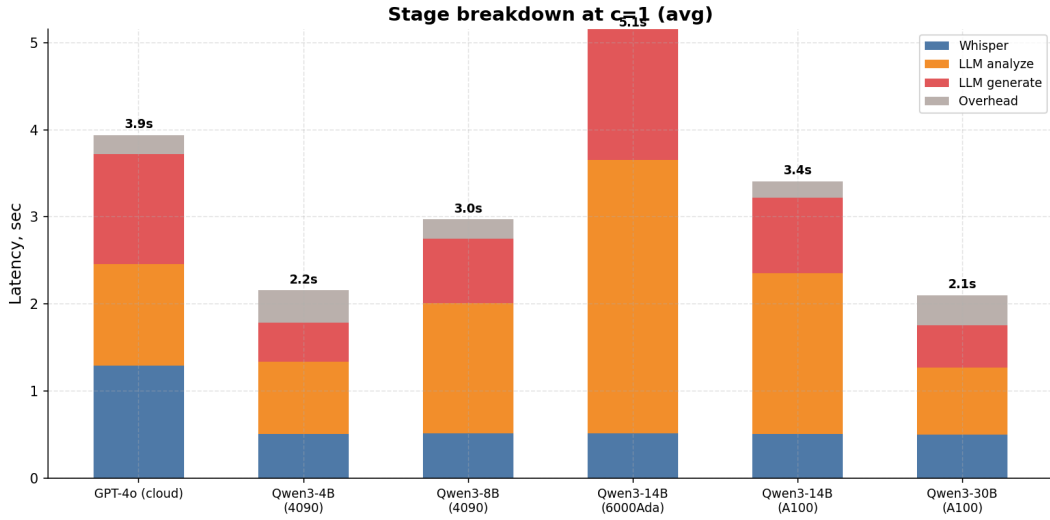


Figure 3: Stage breakdown.

In terms of throughput (Fig. 4), the best result is achieved by Qwen3-4B on RTX 4090: at $c=28$, this configuration reaches about 7 requests/s, outperforming the cloud GPT-4o configuration, which reaches about 6 requests/s. Qwen3-8B and Qwen3-30B reach approximately 5–5.5 requests/s, whereas Qwen3-14B on RTX 6000 Ada remains the slowest configuration, with about 3.7 requests/s under maximum load. All throughput curves continue to rise up to $c=28$, which indicates that the saturation point was not reached within the studied load range.

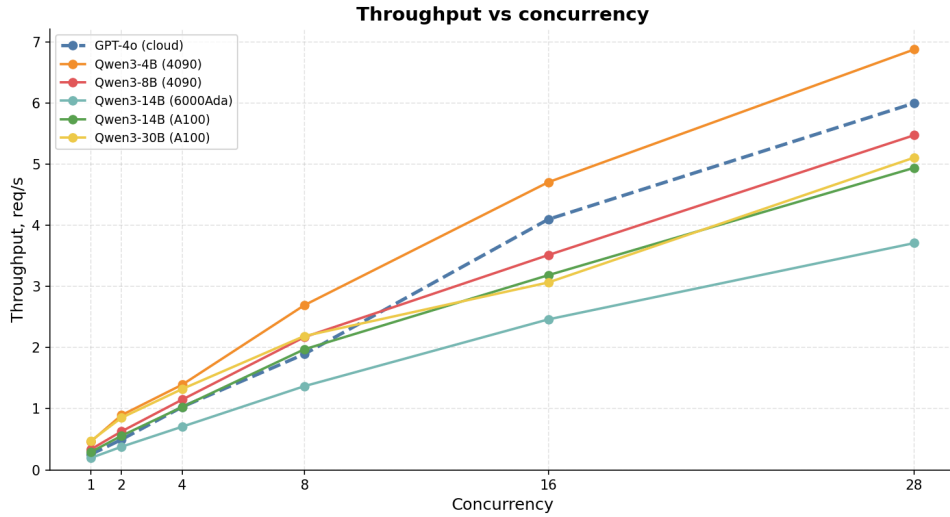


Figure 4: Throughput vs concurrency.

In addition to average latency, the 95th percentile of end-to-end latency was analyzed, as it characterizes the tail of the distribution (Fig. 5). The cloud configuration GPT-4o + Cloud Whisper demonstrates the most stable p95 behavior as parallelism increases: even under high load, tail latency grows moderately. For the self-hosted configurations, p95 increases more markedly, with Qwen3-14B on RTX 6000 Ada showing the least favorable behavior. At the same time, Qwen3-4B on RTX 4090 and Qwen3-30B on A100 are characterized by a more moderate increase in tail latency, which is consistent with the results for average latency and throughput. Therefore, the transition to self-hosted configurations can provide lower average latency, but it requires separate consideration of tail behavior under load.

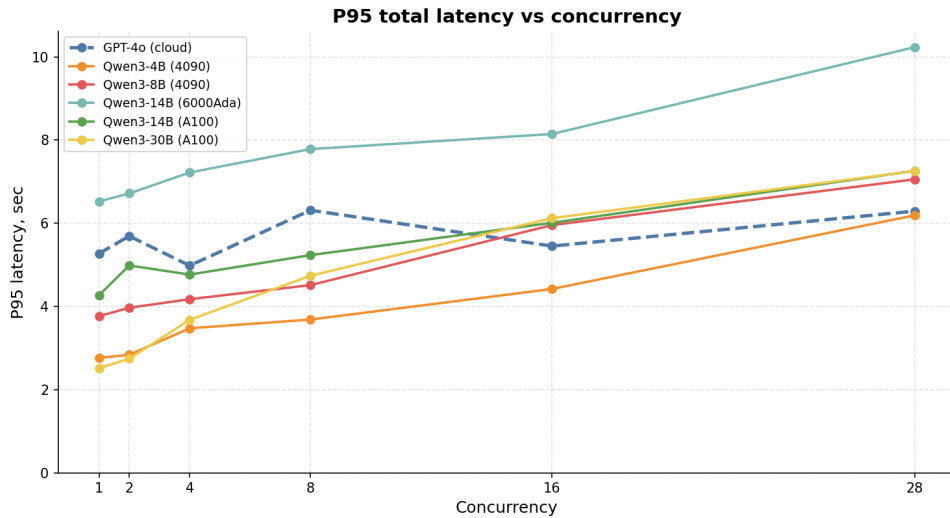


Figure 5: The 95th percentile vs concurrency.

6.3. Product Recognition Quality.

To evaluate the quality of the LLM analyze stage, the lists of products extracted by each configuration from the transcribed text were compared with the reference list for each of the 28 audio files. In total, 86 products were counted in the dataset.

All five self-hosted configurations based on Qwen3 recognized 86 out of 86 products, thereby achieving 100% product recognition recall (Fig. 6). For GPT-4o, 84 correctly extracted products out

of 86 were recorded, which corresponds to 97.93%. Two items were missed: “sauce” in the file describing Caesar salad and “Americano coffee” in the file describing syrnyky.

Both omissions concern products mentioned as additions to the main dish rather than as central elements of the message. This suggests that, in some cases, the cloud configuration interprets such mentions as secondary context rather than as separate items to be recorded.

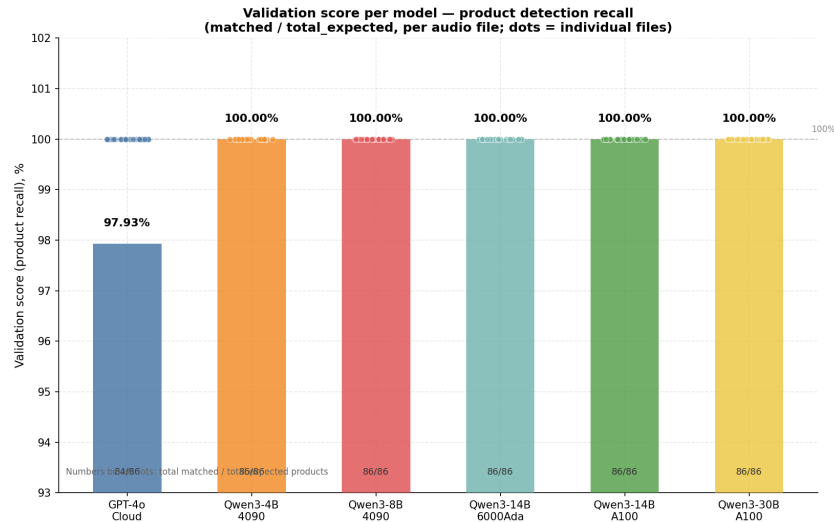


Figure 6: Product Recognition Quality.

The obtained result shows that even the smallest self-hosted model, Qwen3-4B, which has 4 billion parameters, provides 100% product recognition recall within the studied scenario. The deviation of the cloud GPT-4o configuration amounts to two products out of 86; therefore, from a practical perspective, all tested configurations provide sufficient product extraction quality for the Calories365 application scenario.

6.4. Quality of Quantity Assignment.

In addition to product recognition, the LLM analyze stage must determine the quantity of each product. If the user states an exact amount (“200 grams of cottage cheese”), the response must preserve that value. If the amount is not stated explicitly (“a slice of bread,” “half a cutlet,” “6 nuggets”), the model must assign a realistic portion based on the context.

To evaluate the quality of quantity assignment, each “product in the message — product in the LLM response” pair was scored on a scale from 0 to 3: 3 — correct and realistic, 2 — minor deviation, 1 — noticeable error, 0 — incorrect or missing product or quantity. For explicit values (grams, milliliters, counts), closeness to the stated number was assessed; for implicit portions (a spoonful, a slice, a half, a can), the realism of a typical portion was assessed with regard to the dish context. The evaluation was performed manually using fixed criteria identical for all configurations, which ensured consistency of comparison.

The results (Fig. 7) show that GPT-4o has the highest average score, 2.965 out of 3.0. The Qwen3-14B and Qwen3-30B models achieve 2.919–2.920, which is only about 1.5% lower than the cloud configuration. Qwen3-8B shows 2.884. The lowest value is obtained by Qwen3-4B, with 2.625; for this model, errors in assigning implicit portions are recorded more often, including atypical values such as 250 g of sour cream for varenyky instead of a realistic range of 20–50 g.

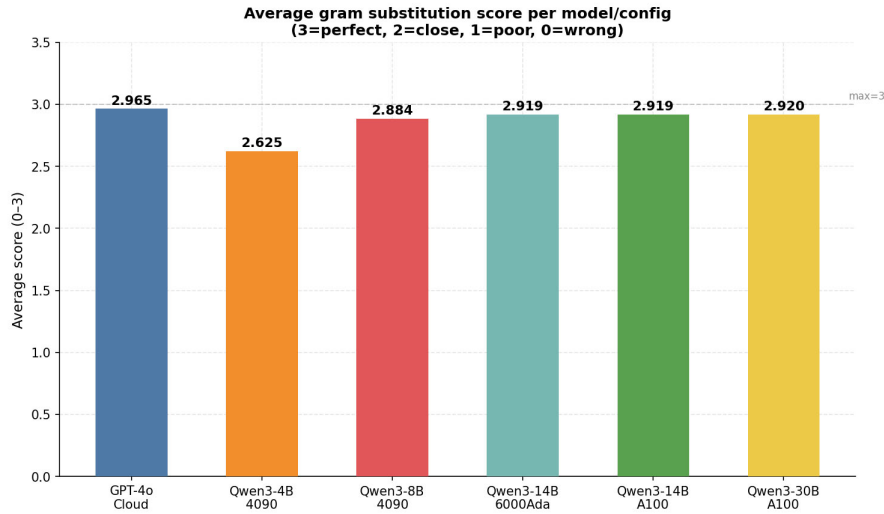


Figure 7: Average Quantity Assignment Score by Model.

Overall, all configurations demonstrate an average score in the range from 2.625 to 2.965 out of 3.0, which indicates the practical suitability of each of them for the studied scenario. The difference between GPT-4o and Qwen3-14B / Qwen3-30B is less than 2%, which means that the larger self-hosted configurations approach the cloud configuration in the quality of quantity assignment. At the same time, Qwen3-4B provides better pipeline performance, although it shows lower accuracy in estimating implicit portions. These results were achieved without additional model training, solely through adaptation of the prompt and the structured output schema at the proxy layer. Potential directions for further improvement include domain adaptation, in particular LoRA fine-tuning on “food description → structured products/grams” pairs, as well as integration of a typical portion database to provide factual reference values when estimating implicit quantities.

7. Conclusions and Future Work

This study performed an end-to-end measurement of the performance and quality of the voice pipeline of the Calories365 application when cloud-based AI components (GPT-4o + Cloud Whisper) were replaced with self-hosted configurations (Qwen3 + Faster-Whisper) through an OpenAI-compatible interface. The experiment covered 6 configurations, 6 levels of parallelism, and 28 Ukrainian-language audio files, which in total yielded 1,008 measurements with a latency breakdown by stage.

The obtained results showed that, within the studied application scenario, replacing cloud components with self-hosted configurations is technically feasible without loss of application quality. For some configurations, this replacement is also accompanied by better performance. In particular, Qwen3-4B on RTX 4090 and Qwen3-30B on A100 at $c=1$ demonstrate a latency of 2.1–2.2 s compared with 3.9 s for the cloud GPT-4o configuration. In terms of throughput, Qwen3-4B at $c=28$ reaches about 7 requests/s, while the cloud GPT-4o configuration reaches about 6 requests/s. In terms of product extraction quality, all self-hosted configurations achieved 100% product recognition recall compared with 97.93% for GPT-4o, and the average quantity assignment score for Qwen3-14B and Qwen3-30B differs from the cloud configuration by less than 2%. The transition to the self-hosted ASR configuration was also not accompanied by a decrease in quality: Faster-Whisper showed a WER of 7.1% compared with 8.5% for Cloud Whisper.

The study also showed that the dominant source of latency in self-hosted configurations is the LLM stage rather than the ASR stage. The choice of GPU class has a substantial effect on pipeline behavior under load: the same Qwen3-14B on RTX 6000 Ada is the slowest configuration, while on A100 it demonstrates markedly lower latency. For Qwen3-30B, the high speed is consistent with the use of an MoE architecture, in which only a subset of parameters is activated for each token.

From a practical perspective, Qwen3-4B on RTX 4090 appears to be the most promising self-hosted configuration among those studied. In the current experiment, it provides the lowest latency and the highest throughput, while its gap in quantity assignment quality relative to the larger models remains moderate. At the same time, these results were achieved without additional fine-tuning, without LoRA adaptation, without RAG, and without other domain improvement mechanisms. In view of this, Qwen3-4B has the greatest practical potential for further deployment: owing to its low computational cost and high speed, this model leaves the largest margin for engineering refinement aimed at improving quantity assignment quality and product recognition. Larger models, in particular Qwen3-14B and Qwen3-30B, already demonstrate higher accuracy in estimating implicit portions in the baseline configuration and are closer to the cloud configuration in the quality of structured extraction; however, Qwen3-4B appears to be the most attractive option precisely as a foundation for further optimization when considering the trade-off among speed, infrastructure requirements, and improvement potential.

In the broader application context, the results of this study indicate that, for Ukrainian-language voice scenarios, the transition from cloud to self-hosted models can be regarded as a practical option for building AI services, provided that the engineering integration is implemented correctly, the models are selected appropriately, and suitable GPU hardware is used. Using a real application as a case study, it is shown that an OpenAI-compatible interface combined with an intermediate proxy layer makes it possible to perform such a replacement without changing the application logic and while preserving the operability of the full pipeline.

The prospects for further development are associated with several directions. First, further training of self-hosted models on domain data is advisable, in particular through LoRA adaptation on “food description → structured products/grams” pairs, which has the greatest potential for improving the quality of implicit portion assignment. Second, integrating a typical portion database or a mechanism for retrieving relevant examples can reduce the number of unrealistic weight estimates for expressions such as “a slice,” “a spoonful,” or “a half.” Third, expanding the dataset will make it possible to confirm the robustness of the obtained results more reliably. Fourth, further investigation of pipeline behavior at higher levels of parallelism ($c=64$, $c=128$) and on multiple GPUs using tensor parallelism will make it possible to localize the saturation range of the studied configurations more precisely.

Declaration on Generative AI

During the preparation of this work, the authors used generative AI tools (models: GPT 5.4 thinking, opus 4.6 and sonnet 4.6) in order to: verification, generating simple code snippets, and clarifying function details. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication’s content.

References

- [1] L. Katerynych, M. Kubichka, K. Zhereb, Calories365: An Innovative Approach to Calorie Tracking with Voice Input, in: A. Doroshenko, N. Kussul (Eds.), Proceedings of the 1st Workshop Software Engineering and Semantic Technologies (SEST 2025), co-located with the 15th International Scientific and Practical Programming Conference (UkrPROG 2025), CEUR Workshop Proceedings, volume 4053, CEUR-WS.org, Kyiv, Ukraine, 2025, pp. 227–236. URL: <https://ceur-ws.org/Vol-4053/paper14.pdf>.
- [2] Y. Paniv, A. Kiulian, D. Chaplynskyi, M. Khandoga, A. Polishko, T. Bas, G. Gabrielli, Benchmarking Multimodal Models for Ukrainian Language Understanding Across Academic and Cultural Domains, in: Proceedings of the Fourth Ukrainian Natural Language Processing Workshop (UNLP 2025), Association for Computational Linguistics, Vienna, Austria, 2025, pp. 14–26. URL: <https://aclanthology.org/2025.unlp-1.2/>.

- [3] M. Romanyshyn, N. Romanyshyn, A. Slabchenko, Eval-UA-tion: A Ukrainian-Language Benchmark for Evaluating Language Models, in: Proceedings of the Third Ukrainian Natural Language Processing Workshop (UNLP 2024), Association for Computational Linguistics, Torino, Italy, 2024, pp. 115–125. URL: <https://aclanthology.org/2024.unlp-1.13>.
- [4] A. Kravchenko, Y. Paniv, N. Drushchak, UAlign: LLM Alignment Benchmark for the Ukrainian Language, in: Proceedings of the Fourth Ukrainian Natural Language Processing Workshop (UNLP 2025), Association for Computational Linguistics, Vienna, Austria, 2025, pp. 36–44. URL: <https://aclanthology.org/2025.unlp-1.4/>.
- [5] M. Makhortykh, A. Baghumyan, V. Vziatysheva, M. Sydorova, E. Kuznetsova, LLMs as information warriors? Auditing how LLM-powered chatbots tackle disinformation about Russia's war in Ukraine, arXiv preprint arXiv:2409.10697, 2024. URL: <https://arxiv.org/abs/2409.10697>.
- [6] L. Kobylyukh, Z. Rybchak, O. Basystiuk, Analyzing the Accuracy of Speech-to-Text APIs in Transcribing the Ukrainian Language, in: Proceedings of the 7th International Conference on Computational Linguistics and Intelligent Systems (COLINS-2023), volume 3396 of CEUR Workshop Proceedings, CEUR-WS.org, Kharkiv, Ukraine, 2023, pp. 217–227. URL: <https://ceur-ws.org/Vol-3396/paper18.pdf>.
- [7] I. S. Bodnarchuk, V. I. Shuliak, O. V. Matsura, O. M. Shymchuk, S. S. Bodnarchuk, Comparative Study of Speech-to-Text Recognition Systems for Ukrainian and English Languages, in: Informatics and Intelligent Systems, Springer Nature Switzerland, Cham, 2024, pp. 195–211. doi:10.1007/978-3-031-60815-5_15
- [8] S. Malleni, K. Shashi, S. S. S. R. Madiraju, P. S. S. R. Madiraju, Benchmarking ASR-LLM Pipelines in Kubernetes: Performance and Scalability Analysis, arXiv preprint arXiv:2602.04900, 2026. URL: <https://arxiv.org/abs/2602.04900>.
- [9] N. Mairittha, T. Mairittha, S. Inoue, Performance Evaluation of Modular Voice Assistant Architectures: A Multi-Stage Pipeline Study, arXiv preprint arXiv:2512.11724, 2025. URL: <https://arxiv.org/abs/2512.11724>.
- [10] T. Scholak, N. Schucher, D. Bahdanau, PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models, in: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP 2021), Association for Computational Linguistics, 2021, pp. 9895–9901. URL: <https://aclanthology.org/2021.emnlp-main.779/>.
- [11] B. Lu, J. He, M. Zhang, C. Zhang, Understanding the Performance of Large Language Models on Structured Data Extraction, arXiv preprint arXiv:2502.11201, 2025. URL: <https://arxiv.org/abs/2502.11201>.
- [12] Qwen Team, Qwen3 Technical Report, arXiv preprint arXiv:2505.09388, 2025. URL: <https://arxiv.org/abs/2505.09388>.
- [13] C. Grozea, A. Verbitsky, Evaluation of QWEN-3 for English to Ukrainian Translation, in: Proceedings of the Tenth Conference on Machine Translation (WMT 2025), Association for Computational Linguistics, 2025, pp. 412–420. URL: <https://aclanthology.org/2025.wmt-1.34/>.
- [14] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, I. Stoica, Efficient Memory Management for Large Language Model Serving with PagedAttention, in: Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23), ACM, New York, NY, 2023, pp. 611–626. doi:10.1145/3600006.3613165.
- [15] G. Yu, J. S. Jeong, G. W. Kim, S. Kim, M. Rhu, Orca: A Distributed Serving System for Transformer-Based Generative Models, in: Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22), USENIX Association, Carlsbad, CA, 2022, pp. 521–538. URL: <https://www.usenix.org/conference/osdi22/presentation/yu>.
- [16] A. Agrawal, A. Khandelwal, S. Panwar, N. Bhardwaj, Sarathi-Serve: Efficient Serving of Large Language Models via Chunked-Prefill, in: Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI '24), USENIX Association, Santa Clara, CA, 2024, pp. 433–450. URL: <https://www.usenix.org/conference/osdi24/presentation/agrawal>.

- [17] Y. Cao, L. Chen, S. Cao, Performance Evaluation of Whisper-Series Speech Recognition Workloads, in: Proceedings of the 2025 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '25), ACM, New York, NY, 2025. doi:10.1145/3769102.3774244.
- [18] Hugging Face, Text Generation Inference — Messages API, 2024. URL: https://huggingface.co/docs/text-generation-inference/messages_api.

A. Online Resources

The project's GitHub repository is available at: <https://github.com/Calories365>