Wizard of POdz: An Adaptive AI Podcast Clip Generator Modeling and Scaling of Generative AI Systems

Sebastian Käslin [*1] **Kaushik Raghupathruni** [*2]

Abstract

Generative AI systems face fundamental scalability challenges when transitioning from prototypes to production services, particularly when balancing the competing objectives of response latency and output quality. This paper presents *Wizard of POdz*, an adaptive AI podcast clip generation system that dynamically switches between text-to-speech (TTS) backends to maintain quality-of-service (QoS) constraints under varying workloads. Our system combines a large language model (Meta-LLaMA 3.1 8B) for script generation with two TTS models offering different latency-quality trade-offs: Kokoro for high-quality audio (40s processing time) and Piper for faster generation (20s processing time).

1. Introduction

Recent advances in generative artificial intelligence (GenAI) have made accessible to the public a wide range of AI services, such as text generation, speech synthesis, and image generation. Scalability represents one of the main challenges when these systems evolve from being a prototype to a deployed service. Service providers have to consider three objectives when designing the system: maintaining low response latency, providing high product quality, and making a profit. Considering the first two factors, typically, in such systems, they are not aligned; higher quality requires either greater computational resources (higher costs) or longer inference times, and on the other hand, faster models may sacrifice the output quality.

The service provider's ultimate goal is to create a loyal group of customers consuming the product, thus the tradeoff between latency and quality directly affects the success or failure of the service itself. Indeed, a system that is either considered too slow for the value it offers, or that it produces low-quality products, in the long run, will most probably lose users.

In this work, we study the tradeoff between latency and quality in the context of an AI podcast clip generator. The proposed system generates short introductory audio clips, of approximately 200 words in length (equivalent to 70-80 seconds of audio), depicting a fictional conversation between a host and a guest on a user-specified topic. The system combines two classes of generative models: a large language model (LLM) to generate structured podcast scripts, and a neural text-to-speech (TTS) model to synthesize the corresponding audio waveform.

In the paper, two alternative LLM-TTS tandems, which differ in their performance characteristics, are explored. One configuration prioritizes audio quality at the cost of higher inference latency, while the second one favors faster generation with reduced audio generation quality. The system has the ability to adapt to the changing workload conditions by dynamically selecting a specific tandem configuration and therefore switching to a different latency-quality tradeoff policy.

The paper first describes the architecture of the proposed system, focusing on the individual models and their integration into an end-to-end generative pipeline. Then, offline experimental results are discussed to define how key system parameters affect generation latency and output quality. Building on these results, a switching strategy is described to manage a scenario in which requests arrive at different rates over time and an acceptable latency must be maintained. Finally, we discuss the limitations of the proposed approach and outline directions for future work.

2. Combination of LLM and TTS for Podcast Clips Generation

The proposed service generates short introductory audio clips for podcasts given a user-specified topic. The generation process consists of two main stages. First, a structured podcast script is produced in textual form. Second, the generated script is converted into an audio waveform. The first stage implementation involves an LLM model, while the second stage relies on a TTS. This modular design allows the system to be customizable and flexible, see Figure 1.

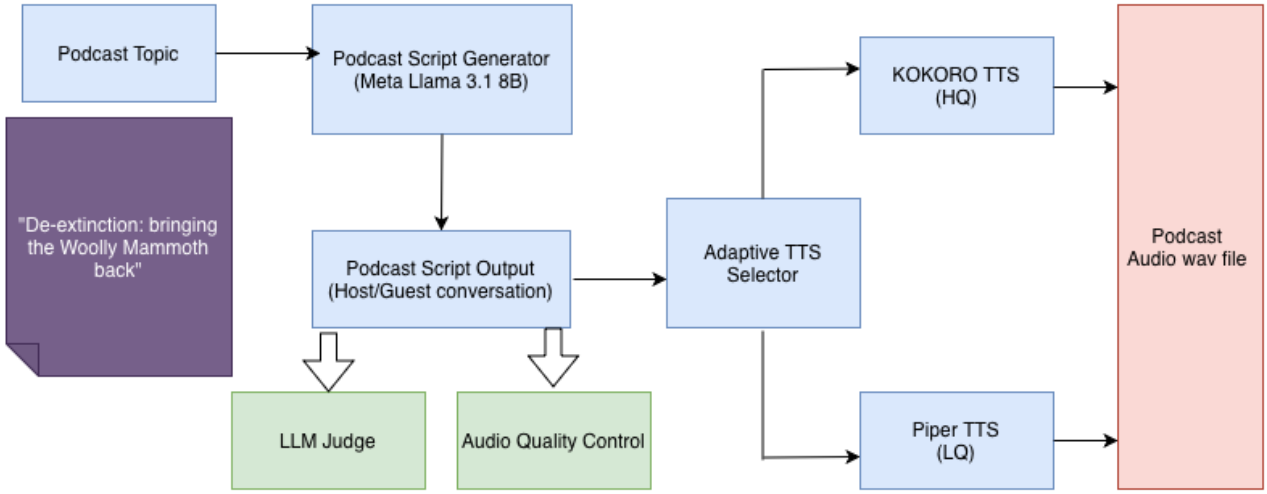


Figure 1. End-to-end podcast generation pipeline with adaptive TTS switching. The system processes user topics through LLM script generation, followed by TTS backend selection between Kokoro (high quality) and Piper (fast).

2.1. LLM

Modern large language models are typically built on the transformer architecture, and they are trained to generate coherent natural language based on a prompt. In our system, we decided to work with Meta-LLaMA 3.1 8B Instruct model[1], which contains approximately 8 billion parameters.

This model was chosen for different reasons, first, it is well-suited for our purpose since it is instruction-tuned, and one requirement of our system is that it needs to be able to produce a structured podcast script, which combines both metadata and dialogue. Second, we have prior experience with the model, it is open source, easily deployable via Hugging Face, and the chat template facilitates prompt engineering and enables the use of few-shot examples to improve the generated output quality.

2.2. TTS

Text-to-speech models convert written text into an audio waveform that simulates human speech. Compared to LLM, TTS are typically smaller; in this work, we consider two open-source TTS models with different performance characteristics: Kokoro TTS[2] and Piper TTS[3].

Kokoro is a neural TTS model with approximately 82 million parameters, and it supports multiple voices[4]. Not all voice embeddings were trained for the same number of hours, resulting in varying audio quality across speakers. To ensure comparability, we select one male and one female voice with similar quality ratings. Kokoro generates audio at a sampling rate of 24 kHz. On Hugging Face TTS arena v1[5] for benchmarking such models, it was placed 6th and 7th, first across open source models.

Piper follows a different design choice; it supports multiple voices[6], but a voice is not an embedding but a full separate model. The voice models are categorized by quality levels, such as high, medium, and low. In our system, we use medium-quality male and female voice models, each with approximately 15-20 million parameters and producing audio at a sampling rate of 22.05 kHz.

2.3. LLM-TTS Pipeline

To integrate the LLM and TTS components into a single pipeline, the system relies on a structured output format produced by the LLM. The generated script structure requirement is that it must contain according labels for the host and the guest, allowing text to be segmented into speaker-specific chunks. The LLM, therefore, is prompted using a carefully engineered instruction which enforces a strict output format composed of a starting metadata section and a following dialogue section. The metadata section contains a definition about the gender of the two speakers, host and guest, allowing later correct voice assignment in the TTS stage, while the dialogue section contains the labelled podcast script. To further encourage adherence to the desired format, the prompt can be preceded by a small number of few-shot examples illustrating valid answers.

Once the dialogue is segmented into chunks, to each one

[1] Hugging Face LLaMA-3.1-8B-Instruct (December 2025)
[2] Kokoro Hugging Face (December 2025)
[3] Piper Repo (December 2025)
[4] Kokoro Voices (December 2025)
[5] Hugging Face TTS Arena v1 (December 2025)
[6] Piper Voices (December 2025)

of them a voice is assigned, and each individual segment is passed sequentially to either the correct TTS voice model, in case Piper is being used, or passed together with the correct voice embedding, in case Kokoro is being used. Audio waveforms for the host and guest are generated alternately and concatenated to form a complete podcast introduction clip.

3. Experiments

To comprehensively evaluate our podcast generation system, we conducted experiments across two distinct phases: offline analysis to characterize component-level behavior, and online queuing analysis to assess system scalability under varying workloads.

3.1. Offline Analysis

The goal of the offline evaluation is to analyse the behaviour of the proposed LLM-TTS pipeline under different configurations and identify an optimal setup for the key parameters, finding a tradeoff between qualities and inference time. In particular we focus on two key controllable hyperparameters of the text generation stage: the maximum token length and the number of few-shot examples included in the prompt.

The prompt passed to the LLM is engineered to enforce the generation of the metadata section, followed by a labelled dialogue. To the prompt, we include an optional number of few-shot examples to encourage format compliance. The prompt template is available in Appendix A.2.

We evaluate a total of 20 configuration pairs obtained by combining five values for the maximum token length (64, 128, 256, 512, 1024) and four values for the number of few-shot examples (0,1,3,5). For each configuration, ten distinct scripts are generated, resulting in a total of 200 generated scripts. The experimental evaluation was conducted in multiple stages. The LLM inference was executed on an NVIDIA A100 GPU using Google Colab[7], and we later ran the TTS stage on an NVIDIA L4 GPU. For this reason, absolute inference time measurements in this experiment section are different from the ones presented later in the online evaluation section. However, this separation does not affect the relative comparison between configurations, which remains consistent across hardware settings.

3.1.1. Few-shot examples generation

Few-shot examples were constructed by prompting a separated larger LLM (GPT-5-mini[8]) with the same structured generation prompt used by our system. Ten podcast scripts covering diverse topics were generated and manually validated to ensure the correct formatting, coherence, and compliance with the prompt-defined constraints, see Appendix A.4. These validated scripts were then used as few-shot examples during offline experimentation.

3.1.2. LLM-Based Script Quality Evaluation

The manual evaluation of the 200 generated scripts is impractical. For this reason, we explored the use of LLM as judges to approximate human assessment of the script quality. As judges, we choose to use the model itself to judge its own generated outputs, and a model from a different family, Mistral-7B-Instruct-v0.3[9]. Each judge assigns scores for relevance, coherence, and compliance based on the original prompt instructions and the output generated. The engineered prompt for the judges is available in the Appendix A.3, and two different rating granularities were tested, 1-5 and 1-10. Indeed, two granularities were tested because at first, with 1-5, it seemed that the judges tended to assign consistently high scores across configurations and doubts about the validity of the judges arose, as shown in Figure 2.

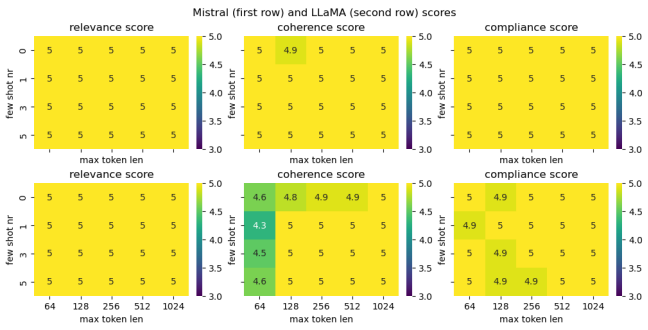


Figure 2. Mean scores on 10 topics for each configuration and for the two LLM judges, with rating granularity 5.

We tried with granularity 10, and we obtained more reasonable results, where the scores vary a bit more, and they aren't set to the max score every time.

To validate whether the judges can still discriminate erroneous outputs, we conduct a controlled validation experiment using the ten few-shot examples. Five of the ten few-shot examples were modified by GPT-5 to introduce structural and semantic errors (e.g., switch script topic, incorrect metadata, or broken dialogue structure). These scripts were then evaluated by the LLM judges, and we decided to test only on granularity 10. As we can see, both models are able to catch errors, and give lower ratings, it seems that Mistral does it in a much more pronounced way, by giving very low ratings if necessary, while LLaMA is less judgmental, respectively, Figure 5 and Figure 4. Both models fail to catch

[7] Google Colab
[8] ChatGPT
[9] Mistral Hugging Face

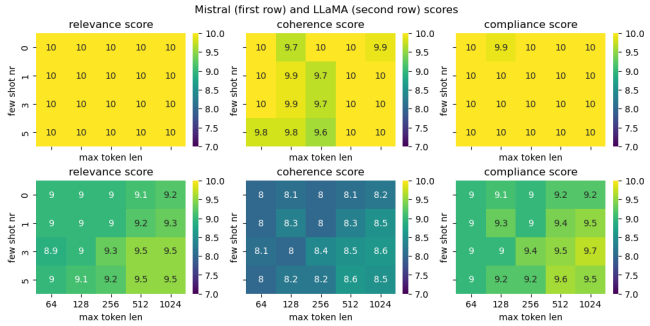


Figure 3. Mean scores on 10 topics for each configuration and for the two LLM judges, with rating granularity 10.

the fourth introduced error, in which the metadata initial line is missing, the topic and the content do not coincide, and there are some structural errors. In addition to that, they seem not to care about the length of the produced output when giving a score, max token length effect on the compliance score should be visible, since the prompt requires a 150/200 words podcast.

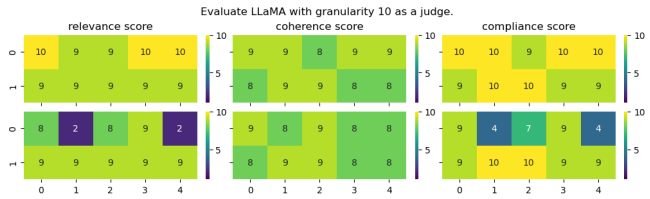


Figure 4. LLaMA scores on the original few-shot examples (first row), and few-shot examples with the first five with deliberately introduced errors (second row).

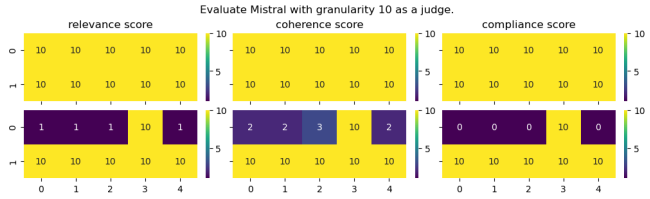


Figure 5. Mistral scores on the original few-shot examples (first row), and few-shot examples with the first five with deliberately introduced errors (second row).

Based on these validation experiments, we conclude that LLM as judges provide a meaningful signal for detecting severely incorrect outputs. And this is positive from the point of view of what our system produces as podcast scripts. However, for the majority of configurations producing structurally correct scripts, the judges tend to assign consistently high scores with limited variance. In particular, Mistral reliably penalizes gross errors, but it does not provide sufficient resolution to discriminate between configurations that produce scripts of comparable quality. On the other hand, the LLaMA as a judge is more ambiguous and difficult to interpret. Due to these limitations, we find LLM-based evaluation with these two models unsuitable for fine-grained configuration selection in this setting.

3.1.3. Configuration Selection Based on Task-Specific Metrics

Given these limitations, we base the final configuration selection on deterministic and task-specific metrics that directly impact system scalability and user experience. To evaluate audio quality, we use Word Error Rate (WER) as an objective metric. For each generated audio clip, the Whisper-small[10] automatic speech recognition (ASR) model transcribes the waveform back into text, which is then compared against the original script. WER captures errors introduced by the TTS model that may reduce the overall audio intelligibility.

Our primary task constraint is that each generated podcast introduction should produce an audio clip corresponding to 150-200 words (60-80 seconds). Figure 6 shows the relationship in our system between the maximum token length and the resulting script length.

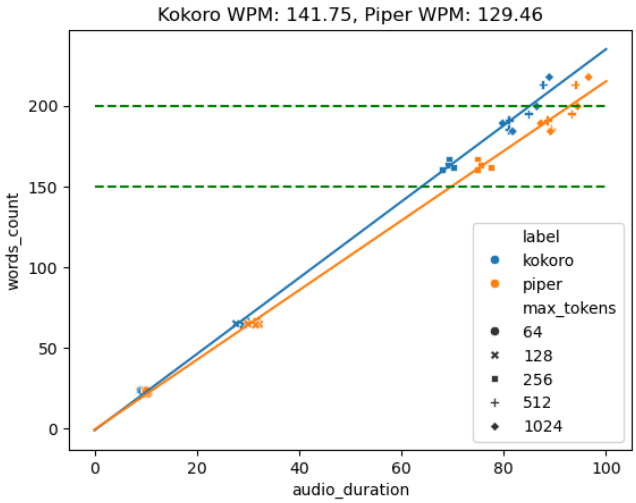


Figure 6. Words count vs audio duration for both Piper and Kokoro for the different tested configurations.

From this analysis, we observe that the maximum token lengths of 256 and 512 reliably produce scripts within the desired range, with the latter that may potentially exceed it. A maximum token length of 1024 does not significantly overshoot the target length, indicating that the model follows the prompt instruction adequately.

Restricting the candidate configurations to a maximum to-

[10] Whisper small Hugging Face

ken length of 256 and 512, we then analyze inference latency and audio quality. Figure 7 shows that increasing the maximum token length from 256 to 512 results in an additional end-to-end latency of approximately 5-8 seconds when using Kokoro and 2-3 seconds when using Piper.

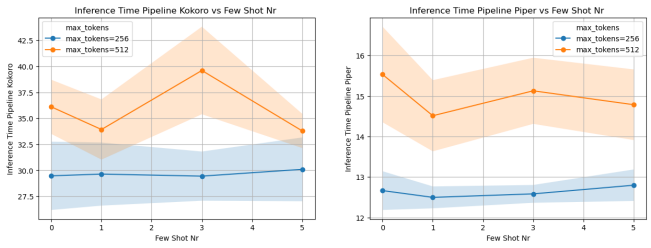


Figure 7. Inference vs number of few shot example for both Piper and Kokoro.

Since the maximum token length of 256 still meets the task requirements and offers lower latency, we select it over the 512 configuration. This choice is further supported by the observed variance in WER for Kokoro, where longer scripts do not consistently yield improved audio quality, Figure 8.

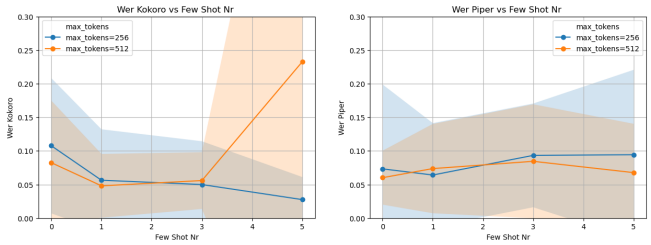


Figure 8. WER vs number of few shot example for both Piper and Kokoro.

Finally, we select three few-shot examples as a compromise between format enforcement and prompt size. Using five examples does not provide measurable improvements and introduces a risk of overfitting, e.g., repeated host or guest names. The final configuration, therefore, balances robustness, latency, and output diversity.

3.1.4. Pipeline Profiling

With the selected configuration of maximum token length of 256, and three few-shot examples included in the prompt, we run the full pipeline for 25 podcasts and measured the individual inference for the two distinct pipelines on NVIDIA RTX4090. Results are summarized in Table 1.

The profiling data revealed several critical insights:

1. **Significant Performance Gap**: Piper demonstrated 3.8× faster average latency compared to Kokoro (11.19s vs 43.72s)
2. **Throughput Advantage**: Piper's higher speed translated to 3.8× greater throughput (5.24 vs 1.38 requests/minute), enabling substantially higher system capacity
3. **Consistent Performance**: Both backends showed low standard deviation (1.45-2.00 seconds), indicating predictable performance suitable for QoS guarantees

Table 1. Pipeline profiling results for different TTS backends (N=25 requests)

Metric	Kokoro	Piper	Unit
Average Latency	43.47	11.45	sec
Median Latency	43.72	11.19	sec
Std Deviation	2.00	1.45	sec
Throughput	1.38	5.24	req/min

To understand these performance differences, we measured individual stage processing times, and found out that the LLM generation stage consistently required 5.4 seconds across the two pipelines, representing 12-47% of total processing time for Piper and Kokoro respectively. While, TTS synthesis emerged as the dominant performance factor, consuming 88-53% of total processing time for Kokoro and Piper respectively, Table 2.

Table 2. Stage-wise processing time breakdown (N=25 requests)

Processing Stage	Kokoro	Piper
LLM Script Generation	5.4s	5.4s
TTS Audio Synthesis	37.9s	6.0s
Total Processing Time	43.3s	11.4s

3.2. Online Analysis

3.2.1. Queuing System Modeling

To evaluate our adaptive podcast generation system under realistic workload conditions, we modeled it as an M/G/1 queue with state-dependent service rates. This model choice allows us to apply established queuing theory results while capturing the unique characteristics of our adaptive system.

The queuing model incorporates two methods:

1. **State-dependent service**: Service rate μ adapts dynamically based on queue length
2. **QoS-aware switching**: TTS Backend is selected by response time constraints

3.2.2. Adaptive Switching Policy

Building on our profiling results, we implemented a queue-aware switching policy. Given a QoS constraint $T_{\text{max}} = 90$ seconds (maximum acceptable response time), the system selects TTS backend as follows:

$$\text{Backend} = \begin{cases} \text{Kokoro} & \text{if } n \times t_{\text{kokoro}} + t_{\text{kokoro}} \leq T_{\text{max}} \\ \text{Piper} & \text{otherwise} \end{cases} \tag{1}$$

where n is the current queue length, $t_{\text{kokoro}} = 40$ seconds, and $t_{\text{piper}} = 20$ seconds. We arbitrarily chose the QoS constraint to be $T_{\text{max}} = 90$ seconds based on the assumption that users could tolerate 90 seconds of wait time for the production of a 70-80 second audio podcast clip. The service times $t_{\text{kokoro}} = 40$s and $t_{\text{piper}} = 20$s were derived from our offline profiling experiments (Table 1), which showed average processing times of 43.47s and 11.45s respectively, rounded to estimate to account for variability.

3.2.3. Experimental Setup

We simulated arrival rates ranging from $\lambda = 0.015$ to 0.045 requests/second (54 to 162 requests/hour) over 30-minute periods. This range was selected to span from light load conditions (where Kokoro can be predominantly used) to heavy load approaching system capacity (where Piper usage becomes dominant). The upper bound of $\lambda = 0.045$ requests/second was chosen to stay safely below Piper's theoretical capacity of $\mu_{\text{piper}} = 0.050$ requests/second while testing near-capacity conditions.

For each λ, we generated Poisson arrival times and processed requests sequentially through our adaptive pipeline. The simulation tracked key metrics including response time, queue length, backend usage, and system stability. Queue thresholds were determined by the switching policy: when queue length n satisfies $n \times 40 + 40 \leq 90$, the system uses Kokoro; otherwise it switches to Piper. This corresponds to a threshold of $n \approx 1.25$, meaning the system switches to Piper when more than one request is waiting.

Hardware Configuration: All experiments were conducted on a UBELIX cluster instance equipped with a 4-core CPU and NVIDIA RTX 4090 GPU with 24GB VRAM. This configuration provided sufficient computational resources for LLM inference and TTS synthesis while maintaining realistic constraints for a single-server deployment.

Experiment Design: We conducted seven distinct experiments corresponding to the arrival rates in Table 3. Each experiment simulated 30 minutes of continuous operation, with metrics collected after system stabilization. This duration ensured sufficient statistical significance while maintaining practical simulation times.

3.2.4. Queuing Performance Results

The results in Table 3 summarize the steady-state behavior of the adaptive queuing system under increasing arrival rates. For each workload level, the reported metrics represent averages over the full simulation horizon and capture the interaction between arrival intensity, backend selection, and effective service capacity. In particular, the table highlights how adaptive backend switching enables the system to maintain stability while dynamically trading off response time and audio quality as load increases. These results provide a quantitative foundation for the stability, QoS compliance, and adaptation trends analyzed in the following subsections.

3.2.5. Stability and QoS Analysis

All configurations remained stable ($\lambda < \mu_{\text{piper}} = 0.050$ requests/second), confirming the theoretical stability condition. The system successfully handled arrival rates up to 75.8% of Piper's maximum capacity. Throughput scaled linearly with arrival rate, reaching 3.26 requests/minute at $\lambda = 0.045$ requests/second, Figure 10(b).

QoS Compliance The adaptive system maintained response times well below the $T_{\text{max}} = 90$ second constraint across all tested arrival rates (Table 3). Maximum observed response time was 55.9 seconds at $\lambda = 0.015$ requests/second, representing a safety margin below the QoS threshold.

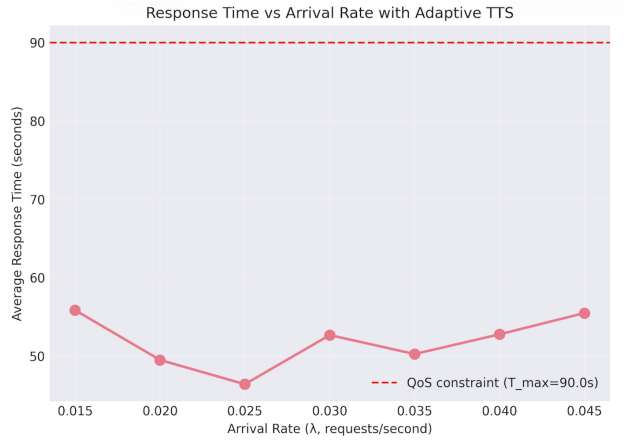
3.2.6. Adaptive Performance Visualization

Adaptive Behavior Analysis The four performance visualizations reveal key insights about system behavior:

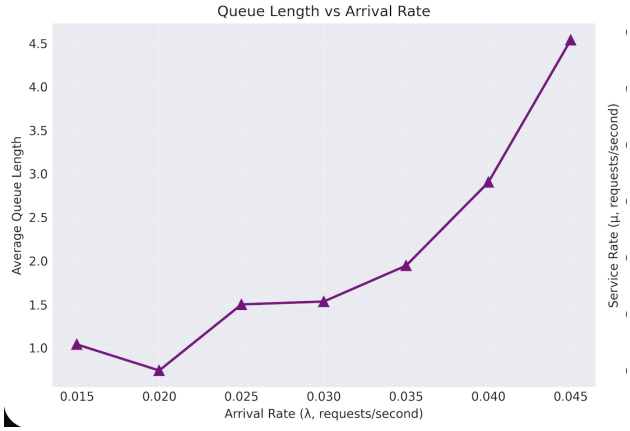
- **Response Time Management** (Fig. 9(a)): Response times remain between 46-56 seconds across all arrival rates, showing effective QoS enforcement with the red dashed line representing the $T_{\text{max}} = 90$ second constraint.
- **Progressive Quality Degradation** (Fig. 10(a)): Kokoro usage (blue bars) decreases from 68% to 24% as arrival rate increases, while Piper usage (pink bars) increases correspondingly, demonstrating load-based adaptation.
- **Queue Regulation** (Fig. 9(b)): Queue lengths grow predictably from 1.0 to 4.5 as arrival rate increases, showing that the adaptive policy prevents unbounded queue growth.
- **Service Rate Adaptation** (Fig. 10(b)): Effective service rate μ (green squares) increases with arrival rate, demonstrating how the system automatically adjusts capacity by switching to faster backends under load.

Table 3. Queuing analysis results with adaptive TTS switching

λ (req/s)	μ (req/s)	Response Time (s)	Avg. Queue Length	Stable	Kokoro (%)	Piper (%)	Throughput (req/min)
0.015	0.031	55.9	1.04	✓	68.0	32.0	1.85
0.020	0.028	49.5	0.74	✓	77.8	22.2	1.68
0.025	0.033	46.4	1.50	✓	61.9	38.1	1.97
0.030	0.033	52.7	1.53	✓	60.0	40.0	2.00
0.035	0.038	50.3	1.95	✓	49.1	50.9	2.27
0.040	0.042	52.8	2.90	✓	40.3	59.7	2.55
0.045	0.054	55.5	4.54	✓	24.2	75.8	3.26

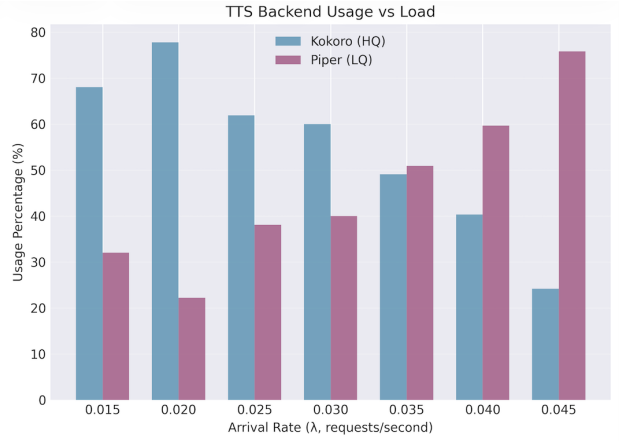


(a) Response time vs arrival rate

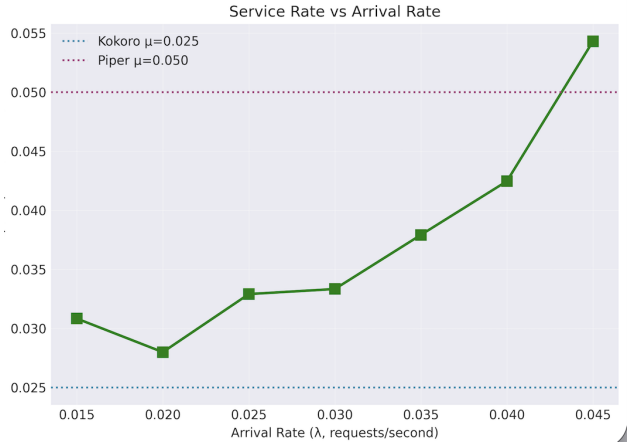


(b) Queue length vs arrival rate

Figure 9. System latency and queuing behavior under increasing load.



(a) TTS backend usage vs load



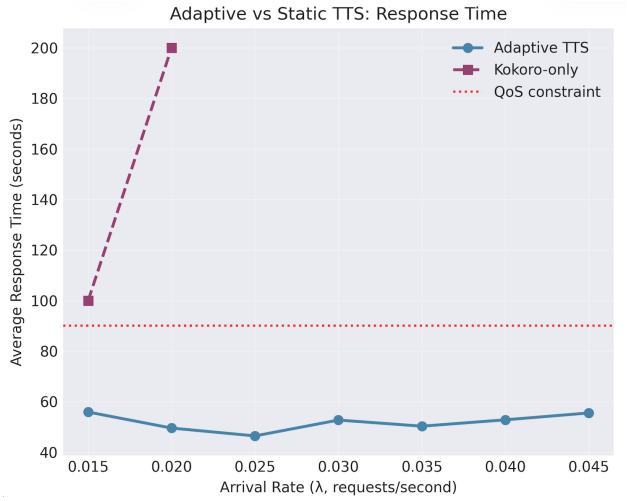
(b) Service rate vs arrival rate

Figure 10. Adaptive switching behavior under increasing load. As arrival rate increases, the system shifts toward faster TTS backends, resulting in higher effective service rates.

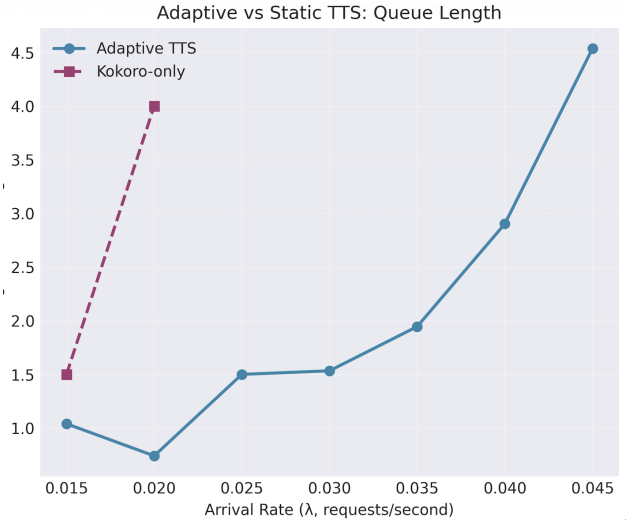
Comparison with Static Strategies Figure 11(a) compares response times between adaptive and static Kokoro-only strategies, while Figure 11(b) shows queue length comparisons.

3.2.7. Key Findings and Practical Implications

- **Response Time Advantage** (Fig. 11(a)): The adaptive strategy (blue circles) maintains response times under 60 seconds, while Kokoro-only (red squares) would exceed 90 seconds at $\lambda > 0.025$ requests/second.



(a) Response time comparison



(b) Queue length comparison

Figure 11. Comparison between adaptive TTS switching and a static Kokoro-only strategy. The adaptive approach maintains bounded response times and queue lengths under high load.

- **Queue Management** (Fig. 11(b)): Adaptive switching (blue circles) keeps queue lengths manageable (0.7-4.5), while Kokoro-only (red squares) would experience unbounded queue growth at higher arrival rates.

Limitations and Practical Considerations

- **Assumption Validity**: The Poisson arrival assumption may not hold for all real-world scenarios (e.g., bursty traffic patterns)
- **Switching Overhead**: Although minimal, frequent switching could impact system predictability
- **Multi-server Extension**: Current implementation assumes single-server configuration; scaling to multiple servers requires additional coordination

4. Conclusion

This paper presented *Wizard of POdz*, an adaptive AI podcast generation system. It navigated the individual characteristic of its foundational components, and explored the fundamental latency-quality trade-off in generative AI services. A detailed analysis on both the LLM and the two TTS models was done in order to select an optimal pipeline configuration by tuning two hyperparameters, the maximum token length and the number of few shot examples. Different kind of metrics were taken into account, first LLM as judges were explored, and then we switched to more objective metrics based on our task-specific goal, such as WER, word count, and inference time. Through profiling, we quantified the performance characteristics of the two TTS backends: Kokoro delivering high-quality audio with 43.47s average latency, and Piper providing faster 11.45s generation with acceptable quality reduction. Based on these results we implemented our queue-aware adaptive switching policy, which dynamically selects TTS backends based on current queue conditions to maintain response times below a 90-second QoS constraint.

In our experimental evaluation we observed the following, first, the adaptive system maintains stability across arrival rates up to 0.045 requests/second. Second, the system achieves graceful quality degradation rather than catastrophic failure, with Kokoro usage decreasing predictably from 68% to 24% as load increases.

Several limitations provide directions for future work. From the point of view of the TTS implementation, we have a fixed number of voices, only one for a male and one for a female, meaning that a conversation between two persons of the same gender becomes a monologue. Due to limited computation resources LLM as judges was done with the model itself and a smaller model, it would be interesting to explore more the capability of this kind of evaluation using larger models. The TTS models are currently CPU bound and GPU implementations would probably lower the latency. Regarding our current implemented switching policy, the main limitations are that we assume Poisson arrivals and we use single-server configuration, with no parallelization. For example, we could use batching for the LLM, and set up multiple TTS models or explore concurrency in this context. Additionally, exploring more sophisticated switching policies, such as those incorporating request priority or user preferences, could further optimize the latency-quality trade-off.

A. Appendix

A.1. Code Repository

The complete code is available at:
https://github.com/rkaushik97/MSGAI-AI-Podcast-Generator/tree/main

A.2. Engineered Generative Prompt

```
{
    "podcast_script_v1": """
Your entire output MUST start with the exact token: ---METADATA---

---METADATA---
HOST_GENDER: [MALE or FEMALE]
GUEST_GENDER: [MALE or FEMALE]
GUEST_NAME: Create a relevant fictional or historical expert name based on the topic
---DIALOGUE---

Your entire output must contain a two-person podcast conversation between HOST and GUEST,
    formatted strictly as:
HOST: ...
GUEST: ...
(continue dialogue)

No other text, instructions, or blank lines are permitted AFTER the '---DIALOGUE---' line.

Content Requirements
Podcast: Adaptive AI Podcast
Topic: {topic}
Length: 150-200 words

Dialogue Structure (follow exactly):
HOST: hook about topic
HOST: podcast + host intro
HOST: guest intro (MUST use the GUEST_NAME you generated)
GUEST: brief greeting
HOST: simple first question
GUEST: 3-5 sentence answer
HOST: follow-up question
GUEST: short answer
HOST: closing line: "Let's get started."

---BEGIN DIALOGUE (must immediately follow the '---DIALOGUE---' line)---
HOST:"""
}
```

A.3. Engineered Judge Prompt

```
LLM_JUDGE_SYSTEM_INSTRUCTION_V2 = (
    "You are an impartial, expert evaluator of podcast scripts. Your task is to score the
        GENERATED SCRIPT "
    "based on the original topic request and the instructions that were given to the model
        . "
    "You MUST output a JSON object containing only the keys 'relevance_score', '
        coherence_score', and 'compliance_score', "
    "all as integers from 1 (Very Poor) to 5 (Excellent). "
    "DO NOT provide any external commentary, reasoning, or text outside of the required
        JSON object.\n\n"
    "Scoring Criteria:\n"
    "- Relevance (1-5): How closely and accurately does the script address the original
        topic request.\n"
```

```
    "- Coherence (1-5): Does the script flow logically? Are the transitions between
        speakers smooth? Is the structure sound?\n"
    "- Compliance (1-5): How well does the script follow the instructions that were
        provided to the model, including structure, metadata usage, and style."
)

LLM_JUDGE_SYSTEM_INSTRUCTION_V3 = (
    "You are an impartial, expert evaluator of podcast scripts. Your task is to score the
        GENERATED SCRIPT "
    "based on the original topic request and the instructions that were given to the model
        . "
    "You MUST output a JSON object containing only the keys 'relevance_score', '
        coherence_score', and 'compliance_score', "
    "all as integers from 1 (Very Poor) to 10 (Excellent). "
    "DO NOT provide any external commentary, reasoning, or text outside of the required
        JSON object.\n\n"
    "Scoring Criteria:\n"
    "- Relevance (1-10): How closely and accurately does the script address the original
        topic request.\n"
    "- Coherence (1-10): Does the script flow logically? Are the transitions between
        speakers smooth? Is the structure sound?\n"
    "- Compliance (1-10): How well does the script follow the instructions that were
        provided to the model, including structure, metadata usage, and style."
)

{
"llm_judge_v2":
    {
    "system_instruction": [LLM_JUDGE_SYSTEM_INSTRUCTION_V2 |
        LLM_JUDGE_SYSTEM_INSTRUCTION_V3],
    "user_query": """
EVALUATION TASK:
Please evaluate the following GENERATED SCRIPT based on the ORIGINAL TOPIC REQUEST and the
     INSTRUCTIONS GIVEN TO THE MODEL.

--- ORIGINAL PROMPT INSTRUCTIONS ---
{original_prompt}

--- ORIGINAL TOPIC REQUEST ---
{original_topic}

--- GENERATED SCRIPT ---
{generated_script}
"""
    }
}
```

A.4. Few-shot Example

```
"
---METADATA---
HOST_GENDER: MALE
GUEST_GENDER: FEMALE
GUEST_NAME: Dr. Lila Montgomery
---DIALOGUE---
HOST: Imagine a world where any idea in your mind can instantly become a vivid, detailed
    artwork. That's the reality AI-generated art is creating today.
HOST: Welcome to the Adaptive AI Podcast, I'm your host, Marcus Lee, exploring how AI is
    reshaping our world.
HOST: Today, we have Dr. Lila Montgomery, a leading researcher in computational creativity
     and digital art innovation. Welcome, Lila!
GUEST: Thank you, Marcus. Excited to be here.
HOST: Lila, how is AI-generated art influencing traditional creativity?
```

```
GUEST: AI-generated art is both a tool and a challenge for traditional creativity. It
    allows artists to experiment with forms and styles instantly, breaking conventional
    boundaries. However, it also raises questions about originality and authorship. Many
    artists find inspiration in collaborating with AI, while others worry about being
    overshadowed by machine-generated content. Overall, it's reshaping how we define and
    engage with creativity.
HOST: Are we seeing a shift in how audiences perceive art because of this technology?
GUEST: Yes, definitely. Audiences are increasingly curious about the process behind AI art
    , and some value the human-AI collaboration more than purely human-made work.
HOST: Fascinating insights. Let's get started.
"
```