Applied Computer Systems
ISSN 2255-8691 (online)
ISSN 2255-8683 (print)
2025, vol. 30, no. 1, pp. 147–156
https://doi.org/10.2478/acss-2025-0016
https://content.sciendo.com

1862
RIGA TECHNICAL UNIVERSITY

Whisperer: A Real-Time Prompting System with Multilayered Semantic Matching and Adaptive Speech Synthesis

Gülbahar Elmas[1], Şeymanur Karaçalı[2], Ebrar Yiğit[3], Fatma Patlar Akbulut[4*]
[1–4]*Istanbul Kültür University, Istanbul, Turkey*

***Abstract* – Whisperer is introduced as an intelligent, real-time prompting system that aims to improve the flow and naturalness of speaking in public and on camera. It is different from regular teleprompters because it does not just follow a script. Instead, it uses Google Cloud's low-latency speech-to-text (STT) and text-to-speech (TTS) services to sync spoken content with a prepared script in real time. The system can handle synonyms, homophones, numeric variations, and spontaneous improvisations because it uses linguistic models such as CMUDict for phoneme-level alignment, FastText for semantic similarity, and BERT for contextual understanding. Whisperer also has adaptive TTS feedback that matches the speaker's speed. This includes changes made in real time based on how long the speaker pauses and how fast they speak. Testing shows that the speakers' fluency and consistency of delivery have both improved.**

***Keywords* – Phonetic confusion handling, real-time speech synchronization, semantic similarity evaluation, transformer-based NLP, word-level and sentence-level matching.**

I. Introduction

Speaking in public is a mentally taxing job that requires more than just knowing the material. It also requires fluency, adaptability, and engaging the audience. Speakers often rely on teleprompter systems to maintain coherence and timing, especially in high-stakes scenarios such as news broadcasting, corporate presentations, or public addresses. Traditional teleprompters work on idea that the speaker will read script exactly as it is written. This strict way of interacting makes it hard for speakers to be natural that makes it hard for them to improvise, change their speech flow, or rephrase things based on the situation, audience feedback, or their own speaking style.

Standard systems do not allow for much deviation from the script. Even small changes, like using different words, changing the order of words, or skipping parts, can make the display and the spoken content out of sync. Because of this, the speaker has to either follow the script exactly or risk losing alignment completely. Recent improvements in automatic speech recognition (ASR), semantic similarity, and text-to-speech (TTS) synthesis have made some things better, but most implementations are still limited by fragile matching algorithms, latency problems, or a lack of understanding at the discourse level. Also, not many systems can change to match the user's speaking speed or let them improvise meaning without losing sync.

To handle these problems, Whisperer is proposed as a smart, real-time speech synchronization and prompting system that will help with natural and adaptive presentation delivery. It changes teleprompting from a one-way interaction to a two-way one. Instead of making the speaker read the text, it dynamically aligns the script with what the speaker is saying. This allows for semantic changes, spontaneous restructuring, and personalised pacing. This is possible by combining advanced natural language processing (NLP) models for homophone recognition, synonym substitution, and context-aware semantic matching with real-time ASR and TTS technologies. Also, the system uses a chunk-based text management strategy that changes based on the rhythm and pause times of each speaker. The contributions of this work are as follows:

- A semantic alignment engine has been developed, employing BERT-based similarity models, FastText embeddings, and phonetic dictionaries to maintain synchronization even when speakers improvise or paraphrase.
- Cloud-based speech-to-text (STT) and TTS services have been utilised to establish a low-latency, real-time feedback loop that provides adaptive audio prompts responding to the speaker's pace.
- A user-centered interface has been designed to display alignment states, support live transcription monitoring, and allow for customisable pacing control.
- Quantitative evaluation has demonstrated that Whisperer effectively maintains synchronization (over 80 % match rate during non-verbatim speech), reduces latency (below 300 ms), and improves user satisfaction (over 80 % approval in user studies).
- A roadmap for future work has been proposed, including multilingual support, offline functionality, and mobile deployment to enhance accessibility and scalability.
- Whisperer moves the field of AI-driven speech assistance forward by changing the way people think about

* Corresponding author's e-mail: f.patlar@iku.edu.tr
Article received 2025-06-21; accepted 2025-11-03

teleprompting as an intelligent, adaptable process. It also opens up new ways for people and computers to work together in real time on spoken communication tasks.

II. Related Work

To get strong semantic alignment in real-time speech support systems, you need to be able to both accurately recognise and flexibly interpret what people say. The ability to match speech to script at a conceptual level, rather than relying on exact lexical correspondence, is a key factor. Sentence-BERT (SBERT) is an adaptation of BERT that Reimers and Gurevych [1] made to create sentence-level embeddings that can be easily compared using cosine similarity. Westera et al. [2] point out that this model makes interactive NLP systems more responsive, but it has problems in discourse-heavy areas like presentations, where coherence between sentences is very important. Vector-based models have been used in other ways to measure semantic similarity. Using Word2Vec representations, Qurashi et al. [3] looked at Jaccard and cosine similarity and found that cosine measures were better for finding paraphrases. Rana et al. [4] went a step further and added FastText embeddings. They found that the semantic fidelity was better, but the computational load was higher, which was something to think about for real-time systems.

Recent progress in automatic speech recognition (ASR) has greatly lowered the number of words that are wrong. Yerramreddy et al. [5] tested the best ASR models – SpeechBrain, Whisper, and Wav2Vec2 – on both lexical and semantic accuracy. Even though they all did well at recognising speech, none of them were able to keep their alignment stable when people spoke naturally. Loda [6] tested Whisper in offline settings that were focused on privacy and found that hardware requirements were a major barrier to low-latency inference. Wang et al. [7] improved Whisper's adaptability by adding speech-based in-context learning (SICL), which allowed making small changes during inference. It looked good, but it could only handle single sentences in Mandarin Chinese, so it di not have coherence at the discourse level.

Domain-specific ASR is still challenging. Jain et al. [8] found that Whisper did not do very well at recognising children's speech without any training, but it got better with fine- tuning. Wav2Vec2 showed better generalization after adaptation, demonstrating that self-supervised learning could work well with limited data. FastSpeech [9] came up with a non-autoregressive architecture for TTS that could quickly and naturally synthesize speech by modelling duration. Later, Ren et al. [10] added semi-supervised methods to make things work better in places with few resources. However, aligning speech over time and prosody for each speaker were still big problems.

IntelliPrompter [11] looked into the need for adaptive prompting by using keyword spotting to control the display of notes in real time. It worked well for navigation, but it did not have semantic generalization, which made it harder to use in more free-form situations. Pandey and Arif [12] showed that the speed at which someone speaks has a big effect on ASR accuracy. This means that prompter systems should let users set their own pace instead of forcing them to follow a set pace. The quality and availability of data are still very important issues. Rossenbach et al. [13] made ASR better by training it on TTS-synthesized audio. This works well for training models offline, but not so well for live use. Alabi et al. [14] present data curation can make a big difference in multilingual settings, where small but clean corpora can do better than big, noisy datasets. This is an important insight for making Whisperer's multilingual features better. Previous research [15] has given us some basic information about semantic alignment, ASR robustness, and adaptive speech systems. However, there are still gaps in real-time, discourse aware, and speaker-adaptive prompting.

III. Methodology

Whisperer is an intelligent teleprompter system that allows users to follow the text naturally and improvise when necessary. The aim is to analyse speech in real time and provide feedback based on its semantic and rhythmic fit with the text. Figure 1 shows the layered architecture of the system. Each module plays a specific role in the process of perceiving, analysing, and responding to user speech.

The system starts by breaking down a text uploaded by the user into meaningful pieces. Rhythm calibration performed with Deepgram analyses the user's speaking speed and pauses, allowing Google TTS to provide personalised voice-over. The user's voice is transcribed in real time with the Google STT API. The resulting text is compared with the target text using BERT and FastText models; CMUdict and word2number are used for homophone and numerical matches. Match types (exact, homophone, and semantic) are shown on the screen by being colored. In case of 70 % BERT similarity, 80 % classical similarity or detection of certain substitute expressions, the system moves on to the next piece. The new piece is presented to the user by being voiced in accordance with the TTS rhythm. This structure provides a harmonious and interactive synchronization experience that is tolerant of the speaker's improvisations.

A. Real-Time Speech Recognition Module

The real-time speech recognition module in the SyncWhisper system instantly transcribes the user's speech while they are speaking, providing text tracking, visual feedback, similarity analysis, and scenario transitions without interruption. Google Cloud STT is used in the module to meet this requirement. It supports streaming recognition and gives partial hypotheses during speech instead of waiting for the end of the utterance. One of the main goals of the system's design is to keep the end-to-end latency $\tau_{total} \leq 300$ ms. Latency is the time it takes for the system to respond to a user's spoken or written word. Total latency decomposed into four part τ_{total}:

$$\tau_{total} = \tau_{capture} + \tau_{transmit} + \tau_{STT} + \tau_{processing}.$$

The delay caused by audio sampling and buffer windowing is $\tau_{capture}$. Using a LINEAR16 encoding format with a 16 kHz sampling rate and a 100 ms buffer stride makes this value as small as possible. $\tau_{transmit}$ is the network transmission latency. Under stable conditions, it is usually between 30 and 50 ms

when streaming over HTTPS. τ_{STT} denotes the internal STT engine delay in generating partial transcriptions. Google's engine supports incremental decoding returning partial tokens every 200–250 ms in the streaming setup. $\tau_{processing}$ involves text normalization, punctuation restoration, and passing the output to the semantic matching pipeline. This is optimised using lightweight, in-memory transformations and thread-level concurrency.

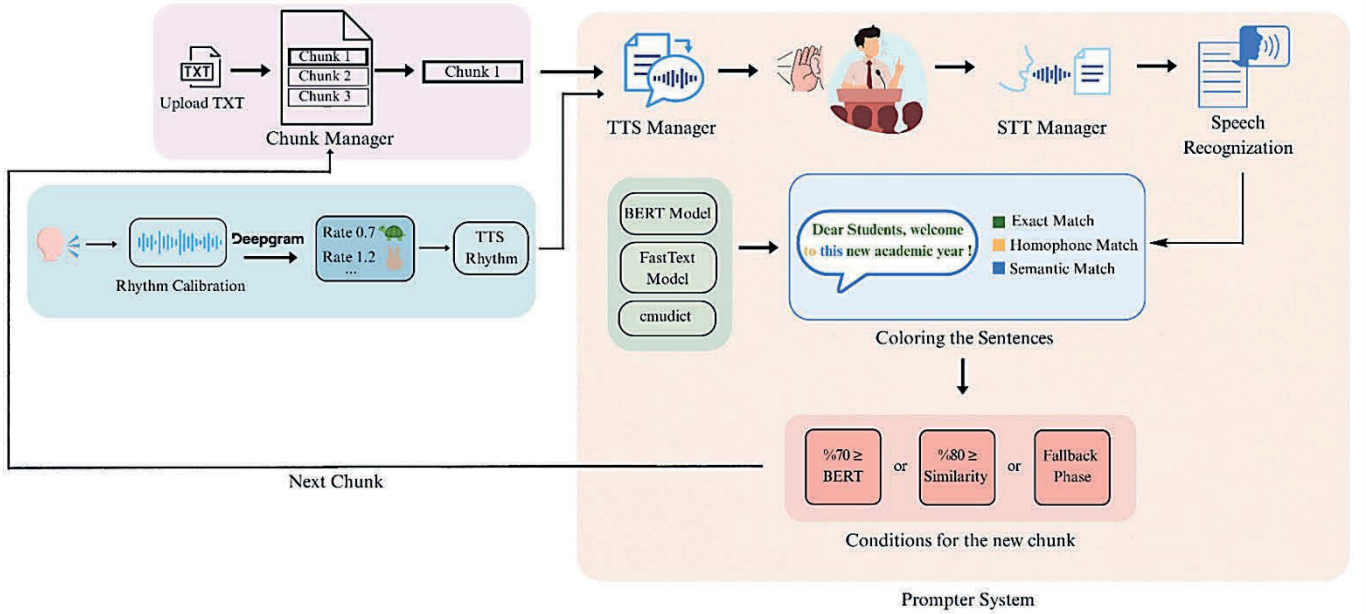


Fig. 1. Layered system architecture of Whisperer System.

Empirical observations show that $\tau_{capture} \approx 100$ ms, $\tau_{transmit} \approx 40$ ms, $\tau_{STT} \approx 100$–120 ms, and $\tau_{processing} \approx 30$ ms, yielding a total system latency in the range of 270–290 ms, satisfying the upper bound requirement for real-time responsiveness.

Streaming API sends audio data to the cloud engine all the time. The STT system uses two-way communication to send intermediate results back to the front end every few hundred milliseconds. This makes it possible for the user interface to show what the speaker said almost right away. This low-latency feedback loop is necessary for making decisions about synchronization, like moving chunks forward and keeping track of what you see. Unlike batch STT systems, which wait for complete sentence boundaries, Whisperer's design allows for chunk-level recognition granularity by leveraging partial result callbacks. These callbacks are caught at the client level and sent to the similarity engine without needing final recognition stability. This reduces unnecessary wait times for computations. Furthermore, the real-time STT module is resilient to variation in speaking tempo and accent that includes a secondary thread that continuously monitors for predefined utterances such as "next" or "continue". These utterances act as control triggers, allowing user agency in chunk progression even before similarity thresholds are reached. The operations are done over encrypted TLS/SSL channels, and audio payloads are only stored in memory for a short time.

B. Real-Time Speech Synthesis Module

The real-time TTS conversion module strengthens the interaction with the user by enabling scenario tracking not only visually but also auditorily. The system uses the Google Cloud TTS service that supports a wide range of languages, lets you control pitch and speed, and has a low latency of 200–300 ms. This delay makes sure that synthesized speech does not fall behind the speaker's rhythm and lets the system fall back quickly if the user stops talking or goes silent. The total latency of the speech synthesis pipeline, which is called $\tau_{TTS\text{-}total}$, can also be broken down into the following parts:

$$\tau_{TTS-total} = \tau_{chunk-transfer} + \tau_{generation} + \tau_{playback-init} + \tau_{audio-buffering}.$$

τ_{chunk} transfer is the time it takes to send the current text chunk to the cloud TTS API. This time is usually between 30 and 50 ms over HTTPS. The actual synthesis time at the TTS engine is given by $\tau_{generation}$. Google TTS sends audio in about 150 to 200 ms for short to medium phrases when using optimised voice models like WaveNet or Tacotron. The delay for audio playback to begin is denoted by $\tau_{playback\text{-}init}$. The streaming lag caused by buffering mechanisms used to avoid underruns or glitches is shown by τ_{audio} buffering. This remains below 30 ms in most tested configurations. These delays result in a total average latency of approximately: $\tau_{TTS\text{-}total} \approx 250 \pm 30$ ms. This ensures the feedback voice remains sufficiently reactive to the speaker's pace without delay. The synthesis module also changes to fit the way each user speaks. When the device is first calibrated, it guesses the user's speaking rate in words per second (WPS). The speakingRate parameter is then used to set the TTS playback speed which is scaled to fit within the engine's allowed range. This makes sure that the synthesized fallback speech matches the user's natural rhythm very closely.

The proposed system is different from regular teleprompters because it uses chunk-level control with real-time TTS that only turns on when semantic or pacing thresholds are crossed, like when the user is silent or goes off track. Thus, TTS module serves as both a correction and an addition keeping user interested and reinforcing the script if necessary. To avoid cognitive overload or overlaps, the system only plays one active audio stream at a time. Any prior audio instance is terminated gracefully before initiating a new one, using synchronized thread handling and audio queue management via the pydub and playsound backends.

C. Natural Language Processing Matching

In real-time speech recognition and synchronization systems, preserving semantic integrity even if the user does not follow the scenario exactly is a significant technical challenge. Whisperer has integrated multi-layered Natural Language Processing (NLP) techniques by going beyond classical text-based matching methods.

Phonetic Similarity and Homophone Handling

Real-time speech recognition systems often struggle to distinguish homophones, which are words that sound the same but have different meanings and spellings, like "to", "too", and "two". These words that sound similar are a big problem for the Whisperer system's alignment mechanism because they are often misrecognised and mismatched during the semantic comparison stages. To get a better idea of how big this problem is, waveform analyses were conducted on a few homophones. Figures 2–4 show that the words "to", "two", and "too" have almost the same sound signatures in terms of amplitude, duration and frequency.

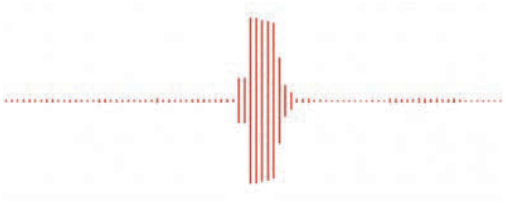


Fig. 2. Waveform of the word "to".

Fig. 3. Waveform of the word "two".

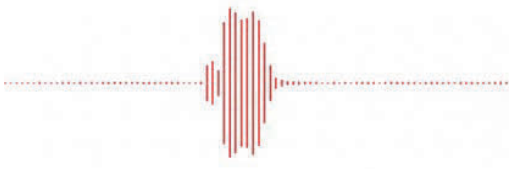


Fig. 4. Waveform of the word "too".

This visual similarity confirms standard STT systems alone are insufficient to distinguish such terms. To mitigate this issue, phoneme-based matching was integrated with CMU Pronouncing Dictionary (CMUDict). This lexicon-based phonetic matching enables the system to better account for homophone-induced confusion by recognising equivalence at the phonemic level rather than relying purely on lexical or semantic forms. This approach improves the systems robustness, fluent speech where such ambiguities are common.

Word-Level Semantic Matching

Word-level semantic matching is used to make sure that small changes in vocabulary, like synonyms, inflections, or derivational forms, do not corrupt the alignment between what the user says and the pre-written script. This is particularly important in scenarios involving spontaneous reformulations, where exact word matching would lead to unnecessary synchronization failures. FastText was used to figure out how similar the meaning of the spoken word w_{STT} is to that of the expected scripted word w_{ref}. Using its subword-enhanced embedding function f: $W \rightarrow R_d$, each word is put into a continuous vector space. Then, the cosine similarity $\sigma(w_{ref}, w_{STT})$ is calculated as follows:

$$\sigma(w_{\text{ref}}, w_{\text{STT}}) = \frac{\left(f(w_{\text{ref}}) \cdot f(w_{\text{STT}})\right)}{\|f(w_{\text{ref}})\| \cdot \|f(w_{\text{STT}})\|}.$$

If σ is greater than a set semantic threshold θ_{word} (which was chosen to be 0.70), the words are semantically matched. This mechanism allows for strong real-time transitions, even when the STT module makes mistakes or changes things on its own.

Semantic Alignment at the Sentence Level

Our system uses sentence-level semantic similarity to check if the user's spoken sentence matches the current script segment. This helps make sure that the synchronization is strong at the phrase or chunk level. This alignment takes into account paraphrasing, phrases that have been reordered or lines that have been partially remembered but still keep their meaning. A pre-trained Sentence-BERT, the paraphrase-mpnet-base-v2 model, is used to encode both the spoken sentence S_{STT} and the reference script segment S_{ref} into contextual embeddings $g(S) \in R_d$. Semantic similarity is computed with cosine similarity.

$$\Sigma(S_{\text{ref}}, S_{\text{STT}}) = \frac{g(S_{\text{ref}}) \cdot g(S_{\text{STT}})}{\|g(S_{\text{ref}})\| \cdot \|g(S_{\text{STT}})\|}.$$

A match is considered when $\Sigma(S_{ref}, S_{STT}) \geq \theta_{sent}$, where the sentence level threshold θ_{sent} is empirically determined as 0.70. This feature lets the system move on to the next text chunk even if the user does not follow the exact script wording perfectly, which makes the delivery sound more natural and smoother. This method finds broader semantic equivalence and is less sensitive to small differences in word choice than word-level matching. The main use of sentence-level analysis is to find out if a chunk can transition and how to handle it if segments are skipped or poorly paraphrased.

D. Text Chunking and Scenario Segmentation

The text chunking and scenario segmentation module lets the system change the flow of the visual script in real time to match the speaker's rhythm. It breaks the input script into parts that make sense and are easy for the brain to handle, and it controls how transitions happen during live speech delivery. This module is the structural foundation for real-time synchronization. It supports the NLP and TTS components by determining the unit of alignment.

Chunking Strategy and Cognitive Load

Long, unsegmented texts can overload the speaker's working memory when they are prompted to speak in real time, which makes them less fluent and less able to remember what they said. Whisperer helps with this by using the "7±2" cognitive load rule [16] to break scripts into text chunks that are about 40 characters or 7±2 words long. Experiments have shown that this length is the best for understanding and remembering things. Beyond this limit, the probability of correctly recalling words from memory decreases as shown in Fig. 5.

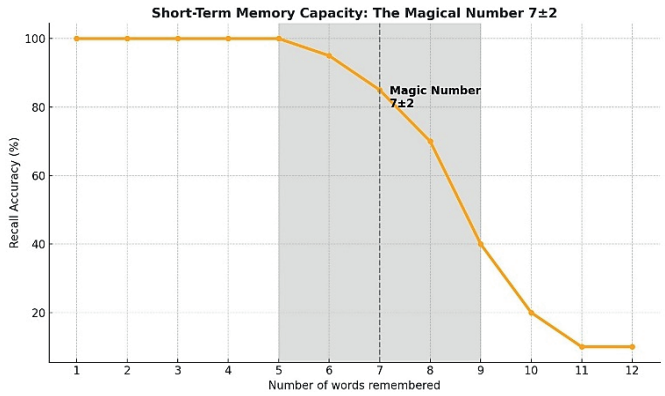


Fig. 5. Word recall accuracy and the 7±2 memory limit.

This cognitive aware chunking makes sure that users are not mentally overloaded, which makes live presentations or recordings go more smoothly.

Tokenizer Based on Rules

A rule-based tokenizer breaks up the script into chunks based on syntax and character length. First, it gives punctuation marks like periods, commas, and question marks, priority to keep semantic boundaries. If there is no punctuation within a target chunk length, it will split at the nearest space near the 40-character limit. It also lets users manually resegment through the interface to meet their needs.

Chunk Transition Logic

The system transition mechanism starts next chunk based on two conditions of similarity. If the classical similarity score is higher than 0.80 or the sentence BERT similarity score is higher than 0.70, a transition happens.

$$\text{Trigger if } \sigma_{\text{classic}} \geq 0.80 \text{ or } \Sigma_{\text{BERT}} \geq 0.70.$$

The system also keeps an eye on speech rhythm and set control phrases like "next" and "continue" to make sure everything flows smoothly. These let users turn off automatic triggers when they want to. This dual logic framework makes sure that chunk transitions stay relevant to the context, keep the rhythm, and let the user control them.

E. Adaptive Transition Triggers

The system has adaptive mechanisms for conversational signals and behavioural patterns. This makes it possible for natural delivery and things to go more smoothly during breaks, improvisations, or planned user-driven transitions. Silence based detection is the first adaptive mechanism. The system sees a pause in the speaker's speech, if that lasts longer than a certain amount of time, called δ_{silence}, as a possible transition cue. In practice, δ_{silence} is set to 1.5 s:

$$\text{Trigger}_{\text{silence}} = \begin{cases} \text{Warn, No Transition} & \text{if silence} \geq \delta_{\text{silence}} \\ \text{No Action} & \text{otherwise} \end{cases}.$$

This mechanism sends a passive "long pause" warning to the user without starting an automatic transition, which keeps the speaker in control. It also knows command level triggers like "next", "continue", and "go on". These predefined phrases make up a command set $C = \{\text{next, continue, } \ldots\}$. If a match is found in the transcribed stream S_t, the system overrides semantic thresholds:

$$\text{if } S_t \cap C \neq \emptyset \Rightarrow \text{Immediate Chunk Transition.}$$

The system also supports skipping and overlapping logic to deal with spontaneous delivery. If a spoken segment makes more sense with a future chunk C_{i+k} than with the current chunk C_i, that is, if

$$\Sigma(S_t, C_{i+k}) > \Sigma(S_t, C_i)$$

then C_i is skipped and C_{i+k} is shown. This makes sure that things stay the same even if the user skips ahead or changes the order of words while speaking. In cases where phrases partially match both C_i and C_{i+1}, the system goes to C_{i+1} and hides repeated content by tracking tokens at the level of the phrase. Divergent or unrelated statements that do not meet any match conditions are visually marked but do not cause a transition. This adaptive interaction framework combines semantic alignment with behavioural context.

$$\text{if } \Sigma(S_t, C_i) < \theta_{\text{sent}} \wedge S_t \cap C = \emptyset \Rightarrow \text{No Transition.}$$

F. Similarity Evaluation

A multi-layered similarity engine was built that checks how similar the user's speech is to the reference script in terms of meaning and structure. This evaluation is important for making decisions in real time, like moving on to the next text chunk, finding improvisation, and skipping content that has not been matched yet. There are a number of strategies to evaluate similarity at the word level. First, exact lexical match between the recognised word (w_{STT}) and the reference word (w_{ref}). If this

does not work, the system uses the word2number module to check for numeric equivalence.

For instance, “two” and “2” are seen as the same. If these do not match, CMUDict is used to see if the words sound the same. For example, it would recognise “to”, “too”, and “two” as the same word. If none of these conditions are met, FastText embeddings are used to try to find a semantic match. The cosine similarity between the vector representations of the words is computed. If it is greater than a certain level, $\theta_{word} = 0.70$, the words are semantically aligned. Final similarity ratio (Sim_{word}) is the total word-level match score divided by the number of reference words.

At the sentence level, a more complete comparison is conducted. The Sentence-BERT model paraphrase-mpnet-base-v2 turns both the transcribed sentence (SSTT) and the reference chunk (S_{ref}) into contextual embeddings. The cosine similarity is calculated with these vectors. If it exceeds the threshold $\theta_{sent} = 0.70$ determined through testing, it is concluded that the sentences have the same meaning. This makes sure that the alignment is strong even if there are paraphrased phrases, phrases that are out of order, or only partial memories.

The system uses a dual-condition logic to move chunks forward. If the similarity at the word level is more than 80 % or the similarity at the sentence level is more than 70 %, a chunk transition starts. If the user talks beyond the current chunk, the unaligned part is checked against the next chunk. The system skips the current chunk and moves on if the rest of the text also has a sentence-level similarity above the threshold. The pseudocode shows the steps to make this decision.

Algorithm 1 Multi-Layered Similarity Evaluation for Chunk Progression

Require: Transcribed sentence S_{STT}, Current chunk C_i, Next chunk C_{i+1}, Word list W_{ref}
Ensure: Decide whether to trigger chunk transition 1: Initialize match_score ← 0, total_words ← $|W_{ref}|$ 2: **for all** $w_i \in W_{ref}$ **do**
3: $w_{STT} \leftarrow$ FindBestMatch(w_i, S_{STT})
4: **if** $w_{STT} == w_i$ **then**
5: match_score ← match_score +1.0
6: **else if** PhoneticMatch(w_i, w_{STT}) **then**
7: match_score ← match_score +0.9
8: **else if** NumericEquivalent(w_i, w_{STT}) **then**
9: match_score ← match_score +1.0
10: **else if** FastTextSim(w_i, w_{STT}) $\geq \theta_{word}$ **then**
11: match_score ← match_score +0.9
12: **end if**
13: **end for**
14: Sim_{word} ← match_score / total_words
15: $\Sigma \leftarrow$ BERTSim(C_i, S_{STT})
16: **if** $Sim_{word} \geq 0.80$ or $\Sigma \geq 0.70$ **then**
17: **return** Transition to C_{i+1}
18: **else if** $\Sigma_{remain}(S_{STT}, C_{i+1}) \geq 0.70$ **then**
19: **return** Skip C_i, Jump to C_{i+1}
20: **else**
21: **return** Stay at C_i
22: **end if**

This combined architecture makes sure that speech is very tolerant of changes, allows for natural improvisation, and keeps alignment fidelity through both local and global semantic comparisons.

G. User Calibration and Adaptation

The system also has a calibration module to adjust the TTS playback and make sure the synthesized speech sounds natural and is in sync with the speaker’s pace. The Deepgram STT engine processes this speech to tell the exact start and end times of each word down to the millisecond. The system uses this to adjust the speech rate and the average length of pauses between words. To calculate the speech rate, the total number of words spoken N_{words} and the total speaking time T_{total} must be known.

$$\text{WPS} = \frac{N_{\text{words}}}{T_{\text{total}}}.$$

After taking WPS value is transformed into TTS speakingRate parameter using a linear transformation (speakingRate = α × WPS + β). For this transformation, α and β are constants that have been found through experience. For example, it was found that α = 0.5 and β = 0.7 worked well to keep perceptual alignment (see Fig. 6). This mapping makes sure that the TTS engine gives faster feedback if the user speaks quickly and slower feedback if the user speaks slowly.

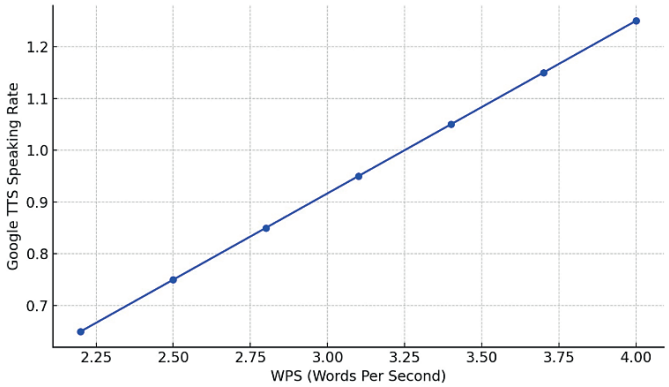


Fig. 6. Linear relationship between WPS and TTS speaking rate.

Additionally, the system changes the rate of speech and finds average pause length between words, which is called τ_{pause}. This calibration improves user experience by making the synthetic voice sound more like a real person and aware of its surroundings.

IV. Experimental Results

A. Performance Analysis at the System Level

Speech-to-Text Performance Evaluation

Four well-known speech recognition engines were compared for the system based on a number of factors, such as latency, accuracy, streaming capability, accent support, and ease of integration. As seen in Table I, Google Cloud Speech-to-Text was determined as the most suitable solution for the system with its high accuracy, low latency, and easy integration features. The latency values presented in Table I represent the

time spent by the system during the speech-to-text conversion stage, as defined in (1). This delay also contributes to the overall end-to-end latency of the system, which includes additional processing, analysis, and speech synthesis stages described later in the paper.

TABLE I
COMPARISON OF STATE-OF-THE-ART STT MODELS

Model	Latency (ms)	Accuracy (%)	Streaming	Accent Support	Integration
Google STT	250	90	Yes	High	Very Easy
Vosk	800	88	Yes	Medium	Medium
Whisper (OpenAI)	1200	80	No	Medium	Medium
Wav2Vec2	1500	75	No	Low	Difficult

Despite being a paid service, its approximately 250 ms latency, 90 % accuracy rate, and comprehensive language/accent support meet the real-time and reliability expectations of the system. Thanks to its streaming support, intermediate results can be obtained during the conversation, enabling instant synchronization. Although Whisper initially stood out due to its open-source structure and ability to work offline, it was found insufficient because of its high latency and inconsistent results. Vosk was advantageous in terms of being lightweight and offline-capable; however, it compromised the naturalness of the system by demonstrating instability in handling homophones and phonetically similar words. Wav2Vec2, on the other hand, was not preferred due to its difficulty of integration, as well as its relatively low speed and accuracy.

Evaluating the Performance of Text-to-Speech

In order to determine the most suitable TTS solution for the system, leading speech synthesis models were compared based on various technical criteria. The evaluation process was based on the basic requirements of the system, such as low latency, natural sound, language diversity, ease of integration and speech rate/intonation control.

TABLE II
COMPARISON OF TTS MODELS

Model	Latency (s)	Naturalness (/10)	Lang. Support	Pitch/Speed Control	Integration
Google TTS	0.25	8.5	100+	Yes	Easy
YourTTS	1.5–2.0	9.0	10+	No	Difficult
FastSpeech2	0.5	7.8	5+	No	Difficult
VITS	2–3	9.2	5+	Yes	Difficult

According to the analysis of Table II, Google Cloud Text-to-Speech was used in the system due to its low latency, multilingual support, and fine-grained control over voice parameters, which are essential for adaptive speech synchronization.

The naturalness scores presented in Table II were obtained from subjective evaluations conducted with ten participants. Each participant rated the perceived naturalness of the synthesized voices on a scale of 1 to 10, and the average of these ratings was used to represent the overall naturalness score for each TTS model.

Despite being a paid service, features such as real-time voice production with latency below 250 ms, support for over 100 languages and dialects, easy integration via REST API, and direct control over voice speed and tone showed high compliance with the system requirements. Although YourTTS and VITS, among the alternative TTS models, were successful in natural voice production, they were insufficient in terms of synchronous performance due to high extraction time and GPU dependency. FastSpeech2, while advantageous in terms of speed, was limited in voice quality and language support. Glow-TTS was not preferred due to installation difficulties and latency issues. While the chunk transition time was 3.5 s with Vosk + VITS in the beginning, this time was reduced to 1.5 s with word-based similarity analysis. However, the targeted real-time could not be achieved. With the use of Google STT and TTS, the total latency decreased to ≤300 ms and, therefore, it was preferred in the system.

End-to-End Latency and Transition Time

The system's real-time performance was tested by measuring the total delay from voice input to synchronized TTS output across all components, such as STT, semantic analysis, and TTS rendering. The system's responsiveness is less than a second, which is important for real-time prompting situations.

TABLE III
END-TO-END LATENCY BREAKDOWN

Component	Latency (ms)	Contribution (%)
Speech-to-Text (Google)	250	33.3 %
NLP Semantic Engine (FastText + BERT)	200	26.7 %
Text-to-Speech (Google)	250	33.3 %
Control + Delay Handling	50	6.7 %
Total End-to-End Latency	**750**	**100 %**

The total end-to-end latency (τ_{e2e}) can be defined as the sum of all module-level delays, including speech recognition, semantic processing, speech synthesis, and control handling:

$$\tau_{E2E} = \tau_{\text{STT}} + \tau_{\text{NLP}} + \tau_{\text{TTS}} + \tau_{\text{control}}.$$

This corresponds to the total delay of approximately 750 ms shown in Table III. According to Table III, the system achieved a total end-to-end latency of 750 ms. Speech-to-Text and Text-to-Speech each contributed about one-third of the delay, while the NLP module accounted for 26.7 %. The response time remained consistent across tests, indicating reliable real-time performance.

B. Scenario-Based Evaluation and Results

In order to evaluate the performance of the Whisperer system in a multi-faceted way according to different user profiles, text structures, and speaking styles, various scenarios were defined, and separate tests were applied for each. These scenarios include parameters such as speech rate, text length, user accent,

and improvised speech. Each scenario is structured to evaluate the real-time synchronization, chunk transition accuracy, and overall performance of the system. For enhanced clarity, all metrics used in the evaluation are explicitly defined: Latency represents the total end-to-end delay (τ_{E2E}), measured in milliseconds. Word Error Rate (WER) is calculated by comparing the STT transcription to the original reference text. Transition Accuracy is the percentage of successful chunk transitions adhering to the dual criteria: BERT similarity ≥0.70 or classical match ≥80 %. Naturalness scores for TTS models were based on subjective 1–10 ratings obtained from ten participants in user studies, and Confusion Rate (Table VIII) is the percentage of homophone misrecognitions by the STT engine. Match Decision is a binary outcome based on whether the word-level FastText similarity exceeds $\theta_{word} = 0.70$ or the BERT sentence similarity exceeds $\theta_{sent} = 0.70$.

The evaluation was rigorously conducted with ten participants (six female, four male) aged between 21 and 27, consisting of undergraduate and graduate students. They were selected from five different nationalities (Turkish, Egyptian, Iranian, Saudi Arabian, and Jordanian), a distribution which enabled the measurement of diverse speech characteristics, including variations in speaking rate, accent, and pronunciation. Each participant completed reading sessions lasting approximately 10–16 min.

The system was tested with different user profiles and text structures; TTS speed calibration was applied for users with varying speaking rates (WPS). Transition performance was evaluated using text lengths: short (≤200 characters), medium (∼500 characters), and long (1000+ characters) texts. The effect of pronunciation diversity was examined with non-native English speakers, and transition stability was analysed through BERT similarity based on return expressions in improvised speech. The core material for the improvisation scenario was a Reference Sentence ("Taking regular breaks improves concentration and productivity") and three deliberate paraphrased versions that participants were asked to speak to test semantic equivalence. Finally, it should be noted that while the system relies on commercial cloud services and subject data is not publicly available due to privacy concerns, the specific test sentence materials have been fully disclosed to support research continuity.

Speech Rate-Based Evaluation

In Table IV, TTS speed calibration was tested for users with different WPS (words per second) rates. The goal was to maintain a natural flow in speech synthesis and prevent synchronization loss during chunk transitions. The table compares the system's matching performance, transition delay, and WER values across users.

TABLE IV

PERFORMANCE METRICS ACROSS USERS WITH DIFFERENT SPEAKING RATES

User	WPS	Speaking Rate	Avg. Transition Time (s)	Avg. BERT (%)	WER (%)
U1 (Slow)	2.00	0.65	10.84	90.32 %	4.28 %
U2 (Medium)	3.23	1.00	7.83	94.00 %	3.68 %
U3 (Fast)	4.12	1.25	5.57	84.66 %	6.44 %

Text Length-Based Evaluation

In this Table V, the system's performance was evaluated based on text length. The goal was to analyse how chunk structure affects transition speed and matching accuracy. Although the increasing number of chunks in longer texts makes synchronization more challenging, the results demonstrate the overall stability of the system.

TABLE V

EFFECT OF TEXT LENGTH ON SYNCHRONIZATION AND RECOGNITION METRICS

Text Length	Character Count	Avg. Transition Time (s)	Avg. BERT (%)	Classic Match (%)	WER (%)
Short	≤200	6.85	91.89 %	97.50 %	2.50 %
Medium	∼500	6.66	94.20 %	97.60 %	5.68 %
Long	1000+	8.89	90.58 %	89.43 %	5.42 %

Accent and Pronunciation Diversity Evaluation

In Table VI, tests were conducted with non-native English speakers to evaluate the impact of accent variation on STT accuracy and chunk transitions. The system was analysed in terms of transition delay, WER, and BERT similarity scores.

TABLE VI

EFFECT OF ACCENT AND PRONUNCIATION DIVERSITY

User	Avg. Transition Time (s)	Avg. BERT (%)	Classic Match (%)	WER (%)
Native	7.53	98.46 %	97.98 %	4.70 %
Non-native	8.69	96.31 %	96.66 %	3.22 %

Improvisation and Fallback Handling

In Table VII, the system's response to users' improvised speech was evaluated. The aim was to assess whether such expressions triggered chunk transitions and to measure the effect of BERT similarity. The table presents transition decisions and detection rates.

TABLE VII

EFFECT OF IMPROVISATION AND FALLBACK HANDLING

Chunk	Keyword Match (%)	Classic Match (%)	BERT (%)	Skip
Chunk 1	50.00 %	66.25 %	81.00 %	Yes
Chunk 2	37.50 %	57.50 %	72.00 %	Yes
Chunk 3	38.46 %	66.15 %	67.75 %	No

For this evaluation, a set of improvised sentences was prepared to test the system's response to semantically similar but lexically different expressions. The reference sentence and its three variations used during the experiment are as follows:

Test Sentence: Taking regular breaks improves concentration and productivity.
Version 1: Short breaks boost your focus and productivity.
Version 2: Resting at intervals can sharpen your attention and work results.
Version 3: Taking breaks is common in project work.

C. Evaluation of Semantic and Phonetic Processing

The system uses a layered NLP engine that can handle phonetic ambiguities and semantic alignment at both the word and sentence levels. This section presents the experimental validation of these components.

Phoneme-Level Confusion in STT Systems

For strong synchronization, it is important to find semantically similar terms even when their surface forms are different. The FastText and BERT embedding models were compared on a number of word pairs that stood for synonyms, morphological derivation, and unrelated ideas (Table VIII).

TABLE VIII
COMMON PHONEME-LEVEL CONFUSIONS IN STT OUTPUT

Target Word	STT Misrecognition	Confusion Rate (%)
to	two / too	88.5
their	there / they're	81.7
right	write	73.2
know	no	69.6
four	for	75.4

Word-Level Semantic Matching

It is important to be able to find semantically equivalent terms even when their surface forms are different for strong synchronization. The FastText and BERT embedding models were compared for synonyms, morphological derivation, and unrelated ideas.

FastText consistently performed better than BERT at measuring semantic similarity between pairs of words, especially in real-time settings where matching isolated words is crucial (Table IX). BERT's ability to understand context made it less useful for comparing individual tokens, even though it was very good at making inferences at the sentence level.

Sentence-Level Semantic Alignment Using Transformer Models

For evaluating improvised speech and checking how similar two pieces of speech are in meaning, sentence-level matching is very important. The Semantic Textual Similarity (STS) dataset was used to test a number of Transformer-based models. The results are exhibited in Table X.

TABLE IX
COMPARISON OF WORD-LEVEL SEMANTIC SIMILARITY SCORES: FASTTEXT VS. BERT

Word Pair (Reference ↔ STT)	Relation Type	FastText Score	BERT Score	Match Decision
education ↔ learning	Synonym	0.74	0.51	FastText matched
buy ↔ purchase	Synonym	0.81	0.46	FastText matched
car ↔ automobile	Synonym	0.78	0.60	Both matched
fast ↔ rapid	Synonym	0.76	0.58	Both matched
child ↔ children	Derivative	0.83	0.61	FastText matched
run ↔ running	Derivative	0.88	0.59	Both matched
education ↔ electricity	Unrelated	0.21	0.36	Both unmatched
book ↔ chair	Unrelated	0.17	0.30	Both unmatched

TABLE X
SENTENCE-LEVEL SIMILARITY EVALUATION ON STS BENCHMARK

Model	Pearson Correlation	Spearman Correlation
paraphrase-MiniLM-L6-v2	0.77	0.76
bert-base-nli-mean-tokens	0.76	0.74
distilroberta-base	0.79	0.78
paraphrase-mpnet-base-v2	**0.83**	**0.82**

The model paraphrase-mpnet-base-v2 had the strongest correlation with human-annotated similarity judgments out of all the ones tested. This is why it was chosen for Whisperer's sentence-level alignment module. Its high semantic accuracy was especially helpful for supporting adaptive transitions in speech that was not written down or was paraphrased.

V. CONCLUSIONS

The Whisperer system presents a robust and smart teleprompter framework that combines real-time speech recognition, synchronized text display, and semantic adaptability. Utilising Google Cloud's low-latency STT and TTS services, alongside multi-layered similarity analysis using CMUDict, FastText, and BERT, the system effectively handles homophones, synonyms, and free-form speech. Empirical results show that the system adapts well to variations in speech patterns and maintains semantic alignment without strict adherence to the script. Numerical evaluations support these findings. In speech rate tests, optimal performance was recorded at a moderate pace (WPS = 3.23), achieving a 94 % BERT similarity and 3.68 % WER. Slow speech led to synchronization delays (BERT: 90.32 %), while fast speech increased the error rate to 6.44 %. In text length evaluations, medium-length texts (500 characters) resulted in a 94.2 % BERT score and 97.6 % classical match, whereas longer texts saw a decline to 89.43 %, indicating chunking complexity.

Accent variation tests showed strong results for non-native speakers (96.66 % match, 3.22 % WER), and in improvisation scenarios, BERT similarity remained consistently above 96 %, ensuring successful transitions. Despite its strengths, the system faces limitations tied to its reliance on cloud-based services, which require constant internet access and may raise privacy and cost concerns. Additionally, the computational demand of models like BERT can hinder responsiveness in real-time use.

REFERENCES

[1] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese BERT-networks," *arXiv preprint arXiv:1908.10084*, Aug. 2019. https://doi.org/10.48550/arXiv.1908.10084

[2] M. Westera, J. Amidei, and L. Mayol, "Similarity or deeper understanding? Analyzing the TED-Q dataset of evoked questions," in *Proceedings of the 28th International Conference on Computational Linguistics*, Barcelona, Spain, Dec. 2020, pp. 5004–5012. https://doi.org/10.18653/v1/2020.coling-main.439

[3] A. W. Qurashi, V. Holmes, and A. P. Johnson, "Document processing: Methods for semantic text similarity analysis," in *2020 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, Novi Sad, Serbia, Aug. 2020, pp. 1–6. https://doi.org/10.1109/INISTA49547.2020.9194665

[4] A. Rana, A. Pant, N. Rawat, P. Rawat, S. Vats, and V. Sharma, "Semantic similarity analysis using FastText," in *2024 IEEE 3rd World Conference on Applied Intelligence and Computing (AIC)*, Gwalior, India, Jul. 2024, pp. 454–460. https://doi.org/10.1109/AIC61668.2024.10731025

[5] D. R. Yerramreddy, J. Marasani, P. Gowtham, and G. Harshit, "Speech recognition paradigms: A comparative evaluation of SpeechBrain, Whisper and Wav2Vec2 models," in *2024 IEEE 9th International Conference for Convergence in Technology (I2CT)*, Pune, India, Apr. 2024, pp. 1–6. https://doi.org/10.1109/I2CT61223.2024.10544133

[6] E. Loda, "Riconoscimento del parlato mediante openai Whisper," unpublished report, 2024.

[7] S. Wang, C.-H. Yang, J. Wu, and C. Zhang, "Can Whisper perform speech-based in-context learning?" in *ICASSP 2024–IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Seoul, Korea, Apr. 2024, pp. 13421–13425. https://doi.org/10.1109/ICASSP48485.2024.10446502

[8] R. Jain, A. Barcovschi, M. Yiwere, P. Corcoran, and H. Cucu, "Adaptation of Whisper models to child speech recognition," *arXiv preprint arXiv:2307.13008*, Jul. 2023. https://doi.org/10.48550/arXiv.2307.13008

[9] Y. Ren, Y. Ruan, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu, "FastSpeech: Fast, robust and controllable text to speech," in *Advances in Neural Information Processing Systems*, vol. 32, 2019. https://proceedings.neurips.cc/paper_files/paper/2019/file/f63f65b503e22cb970527f23c9ad7db1-Paper.pdf

[10] Y. Ren, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu, "Almost unsupervised text to speech and automatic speech recognition," in *International Conference on Machine Learning*, 2019, pp. 5410–5419. https://proceedings.mlr.press/v97/ren19a/ren19a.pdf

[11] R. Asadi, H. Trinh, H. J. Fell, and T. W. Bickmore, "IntelliPrompter: Speech-based dynamic note display interface for oral presentations," in *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, Nov. 2017, pp. 172–180. https://doi.org/10.1145/3136755.3136818

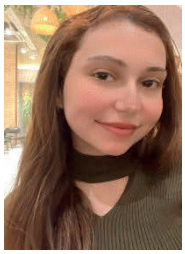
[12] L. Pandey and A. S. M. N. Arif, "Effects of speaking rate on speech and silent speech recognition," in *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, Apr. 2022, pp. 1–8. https://doi.org/10.1145/3491101.3519611

[13] N. Rossenbach, A. Zeyer, R. Schlüter, and H. Ney, "Generating synthetic audio data for attention-based speech recognition systems," in *ICASSP 2020 – IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Barcelona, Spain, May 2020, pp. 7069–7073. https://doi.org/10.1109/ICASSP40776.2020.9053008

[14] J. Alabi, D. I. Adelani, D. Ruiter, and C. C. Emezue, "The effect of curated vs. noisy data on cross-lingual transferability for low- resource languages," in *Findings of the Association for Computational Linguistics: EMNLP 2022*, 2022, pp. 2741–2752.

[15] R. Sadigov, E. Yıldırım, B. Kocaçınar, F. Patlar Akbulut, and C. Catal, "Deep learning-based user experience evaluation in distance learning," *Cluster Computing*, vol. 27, pp. 443–455, Feb. 2024. https://doi.org/10.1007/s10586-022-03918-3

[16] G. A. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information," *Psychological Review*, vol. 63, no. 2, pp. 81–97, 1956. https://doi.org/10.1037/h0043158



Gülbahar Elmas is a Computer Engineer specialising in AI and backend development. She graduated from İstanbul Kültür University with a B.Sc. in Computer Engineering. Gülbahar focuses on developing intelligent systems using modern frameworks, combining artificial intelligence techniques with scalable backend architectures.
E-mail: glelmas13@gmail.com
ORCID iD: https://orcid.org/0009-0003-3899-571X

Ebrar Yiğit is a Data Warehouse and Business Intelligence Associate Consultant with a background in computer engineering. She graduated from Istanbul Kültür University with a B.Sc. in Computer Engineering. Ebrar specialises in data analytics, ETL processes, and business intelligence systems, contributing to data-driven decision-making and performance optimisation in enterprise environments.
E-mail: ebraryigit02@gmail.com
ORCID iD: https://orcid.org/0009-0009-4909-3408

Şeymanur Karaçalı is an Assistant Data and Analytics Governance Specialist with a background in Computer Engineering. She holds a B.Sc. degree from Istanbul Kültür University and has experience in the banking and defence industries. With an analytical mind-set and technical foundation, she has contributed to data governance, mining, and analysis projects, while also gaining experience in project management and business analysis. She continues to advance her expertise in data science.
E-mail: seymakaracali34@hotmail.com
ORCID iD: https://orcid.org/0009-0007-3697-915X

Fatma Patlar Akbulut received the B.S. and M.S. degrees in Computer Engineering from Istanbul Kultur University and the PhD degrees in Biomedical Engineering from Istanbul University in 2017. Her doctoral work focused on developing machine-learning-enabled wearable systems for long-term cardiovascular disease monitoring. She joined the Computer Engineering Department at Istanbul Kultur University (IKU) in 2019. She is currently serving as the Chair of the Software Engineering Department, further contributing her expertise in the field. Prior to IKU, Dr. Patlar Akbulut worked as a Postdoctoral Researcher at the Computer Science Department and Advanced Self-Powered Systems of Integrated Sensors and Technologies (ASSIST) Centre of North Carolina State University (NCSU) between 2017 and 2019. Her research interests include biomedical signal processing, affective computing, data analytics, and wearable systems for healthcare.
E-mail: f.patlar@iku.edu.tr
ORCID iD: https://orcid.org/0000-0002-9689-7486