J.A.R.V.I.S: A Tri-Tier Architecture for Low-Latency, Context-Aware Desktop AI Assistants

Aangee Jain[1], Harry Joshi[2], Rutvik Ghadigaonkar[3], Priyanka Bhilare[4]
[1234] *Department of Computer Engineering, MCT's Rajiv Gandhi Institute of Technology, Andheri (West), Mumbai – 400053, Maharashtra, India*
[123] *Student, MCT's Rajiv Gandhi Institute of Technology, Andheri (West), Mumbai – 400053, Maharashtra, India*
[4]*Assistant Professor, MCT's Rajiv Gandhi Institute of Technology, Andheri (West), Mumbai – 400053, Maharashtra, India*

***Abstract*—The rapid advancement of artificial intelligence has significantly transformed the interaction between humans and computing systems. However, existing voice assistants such as Siri, Alexa, and Google Assistant are primarily optimized for mobile or smart-home environments and often fail to meet the latency, privacy, and contextual processing requirements of professional desktop workflows. This paper presents J.A.R.V.I.S (Just A Rather Very Intelligent System), a desktop AI assistant designed using a Tri-Tier Gateway Architecture that balances the advantages of local execution and cloud-based intelligence. The proposed architecture integrates three layers: a rule-based command execution layer for zero-latency system operations, a local machine learning layer based on a Bidirectional Long Short-Term Memory (Bi-LSTM) intent classifier for natural language understanding, and a cloud-based generative AI layer using the Groq Llama-3 model for advanced reasoning and contextual conversation. The system also incorporates bilingual interaction capabilities, screen-aware computer vision, persistent conversational memory, and acoustic latency-masking techniques to improve user experience. Experimental observations demonstrate that the proposed hybrid architecture significantly reduces command response latency while maintaining high contextual intelligence and privacy. The results highlight the potential of hybrid AI architectures for next-generation desktop automation systems.**

Index Terms- AI Desktop Assistant, Bi-LSTM, Context-Awareness, Desktop Automation, Human-Computer Interaction, Hybrid AI Architecture, Natural Language Processing, Zero-Wait Control.

I. INTRODUCTION

Human–computer interaction has evolved from traditional keyboard-based interfaces to intelligent systems capable of understanding natural language and voice commands. Modern voice assistants such as Amazon Alexa, Apple Siri, and Google Assistant have become widely adopted in mobile devices and smart home ecosystems. Despite their popularity, these systems exhibit several limitations when applied to professional desktop environments.

First, most commercial assistants rely heavily on cloud-based processing, which introduces significant latency due to network communication. This delay disrupts workflow efficiency when users require instantaneous system commands such as adjusting brightness, opening applications, or executing automation scripts. Second, cloud-dependent architectures raise privacy concerns, as user data must be transmitted to remote servers for processing. Finally, purely local assistants lack the conversational intelligence and reasoning capabilities offered by large language models.

To address these limitations, this research proposes J.A.R.V.I.S, a hybrid desktop AI assistant built on a Tri-Tier Gateway Architecture. The proposed system integrates rule-based execution, local neural network processing, and cloud-based generative AI. By distributing tasks across these three layers, the architecture enables *zero-latency execution* for system commands, intelligent local interpretation of user intent, and advanced reasoning through large language models.

The primary objective of this research is to design and implement *a context-aware desktop assistant* that achieves *low latency, high intelligence,* and *enhanced user experience* through a hybrid AI architecture.

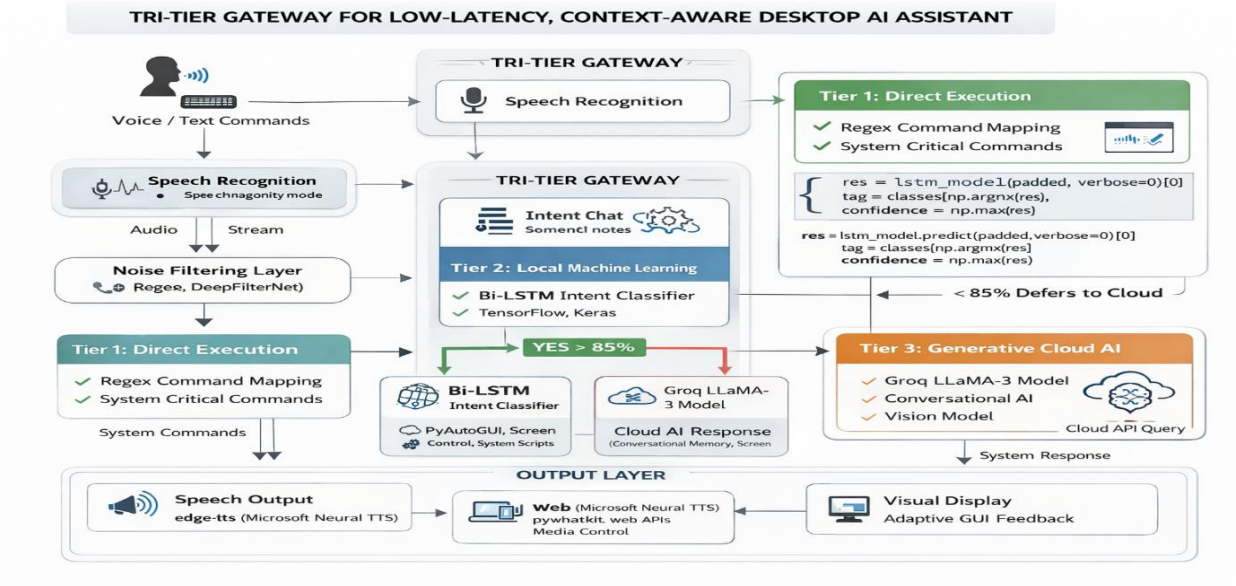

Fig. 2.1. Proposed Tri-Tier Gateway Architecture illustrating the routing of user commands between rule-based execution, Bi-LSTM intent classification, and cloud-based generative AI processing.

II. SYSTEM ARCHITECTURE

The core innovation of J.A.R.V.I.S lies in its *Tri-Tier Gateway Architecture*, which dynamically routes user commands to the most efficient processing layer.

A. Tier 1: Direct Execution (Rule-Based)

The first tier performs *rule-based command execution* using predefined regular expression mappings. This layer handles critical and deterministic commands such as:

- Opening or closing applications
- Adjusting screen brightness
- System shutdown or sleep
- Triggering predefined APIs

Because these commands bypass machine learning and cloud processing, they *achieve near-zero latency execution,* making them suitable for system-level automation.

B. Tier 2: Local Machine Learning Layer (Bi-LSTM)

The second tier introduces *local natural language understanding through a Bidirectional Long Short-Term Memory (Bi-LSTM) neural network*. This layer interprets fuzzy or informal user commands.

Example:

User command:

"My eyes are hurting"

Detected intent:

brightness_down

The neural network processes tokenized input sequences and predicts the corresponding intent class. A *confidence threshold of 85%* determines whether the prediction is sufficiently reliable. If the confidence falls below this threshold, the request is forwarded to the cloud AI layer

C. Tier 3: Generative Cloud AI (Groq Llama-3)

The third tier utilizes *Groq's Llama-3 large language model* to process complex queries requiring reasoning, contextual understanding, or general knowledge. This layer is responsible for:

- Conversational dialogue
- Question answering
- Context-aware assistance
- Code explanation and debugging

The Groq API enables ultra-fast token generation, reducing typical cloud latency while maintaining advanced AI capabilities.

As illustrated in *Fig. 2.1.* ,the system utilizes a gateway to evaluate user input. Tier 1 handles high-

priority regex commands, while Tier 2 employs a Bi-LSTM model for local intent classification. Queries falling below the 85% confidence threshold are automatically routed to the Tier 3 Cloud LLM.

III. METHODOLOGY & IMPLEMENTATION

The J.A.R.V.I.S assistant was implemented using a *Python-based modular architecture* integrating multiple machine learning and automation frameworks.

A. Technology Stack

The proposed J.A.R.V.I.S desktop assistant was implemented using a *Python-based modular architecture* integrating multiple machine learning, automation, and web service frameworks.

The primary programming language used for development is *Python 3.10,* which provides extensive libraries for artificial intelligence, system automation, and web integration. For natural language understanding, the system employs the *TensorFlow and Keras deep learning frameworks, through which a Bidirectional Long Short-Term Memory (Bi-LSTM) neural network* is implemented to perform intent classification.

Voice input is captured using the *SpeechRecognition Python library*, which converts spoken commands into textual input for further processing. The system generates natural voice responses using *edge-tts,* which utilizes Microsoft Neural Text-to-Speech voices to provide realistic speech synthesis.

To enable system automation and control of desktop applications, the assistant integrates *pyautogui*, along with built-in Python modules such as *subprocess and os,* allowing the execution of system-level commands and automation tasks.

The assistant also interacts with external information services through various APIs. These include the OpenWeatherMap API for weather information, *Google News RSS feeds* for retrieving current news updates, and the *Wikipedia API* for knowledge retrieval. Additionally, messaging automation functionality is implemented using *pywhatkit,* enabling the system to send messages through *WhatsApp Web.*

This combination of machine learning frameworks, automation libraries, and web APIs enables the assistant to perform real-time task automation, conversational interaction, and information retrieval efficiently.

B. Bi-LSTM Intent Classification Model

To enable local natural language understanding, the proposed system employs a *Bidirectional Long Short-Term Memory (Bi-LSTM) neural network* for intent classification. The model was implemented using the *TensorFlow and Keras deep learning frameworks.*

The purpose of this model is to interpret user commands expressed in natural language and classify them into predefined intent categories, such as system control commands, information queries, or automation tasks. Bidirectional LSTM networks are particularly suitable for this task because they process textual sequences in both forward and backward directions, allowing the model to capture contextual dependencies within user commands.

The architecture of the neural network is shown below:

Snippet 1: The Bi-LSTM Architecture

```
model = Sequential([
    Embedding(input_dim=len(word_index) + 1, output_dim=32, input_length=MAX_LENGTH),
    Bidirectional(LSTM(64)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(len(classes), activation='softmax')
])
```

The *Embedding layer* converts textual input into dense numerical vector representations that capture semantic relationships between words. These embeddings are then processed by a *Bidirectional LSTM layer* consisting of 64 units, which learns contextual relationships within the command sequence.

To prevent overfitting and improve the generalization capability of the model, a *Dropout layer with a rate of 0.3* is incorporated. The extracted features are then passed through *a fully connected dense layer* with ReLU activation to enhance feature representation. Finally, the output layer uses *a Softmax activation*

function to classify the input command into one of the predefined intent categories.

This model enables the system to accurately interpret flexible natural language commands while maintaining efficient local processing within the *Tier-2 layer of the Tri-Tier architecture.*

C. Confidence-Based Tier Routing

After the intent prediction stage, the system evaluates the *confidence score produced by the Bi-LSTM classifier* before executing any command. This mechanism ensures that only highly reliable predictions are executed locally, while uncertain commands are forwarded to a more powerful reasoning model.

The confidence value is derived from *the Softmax probability distribution* produced by the neural network. The class with the highest probability is selected as the predicted intent, and its corresponding probability value represents the model's confidence score.

The decision routing mechanism is implemented as follows:

```
res = lstm_model.predict(padded, verbose=0)[0]

tag = classes[np.argmax(res)]

confidence = np.max(res)


if confidence > 0.85:

    # Execute Local PC Command (Tier 2)

else:

    # Route to Cloud LLM (Tier 3)
```

A *confidence threshold of 85%* is used to determine whether the predicted intent is sufficiently reliable. When the confidence value exceeds this threshold, the command is executed locally through system automation modules such as application control, media playback, or operating system operations. This enables *low-latency response times* since the command is handled entirely on the local machine.

However, if the confidence value falls below the threshold, the command is forwarded to the *Tier-3 cloud-based generative AI layer,* which utilizes the *Groq Llama-3 large language model* for deeper semantic reasoning and conversational processing. This fallback mechanism ensures that ambiguous or complex queries are handled with higher contextual intelligence.

The confidence-based routing strategy therefore plays a critical role in the proposed *Tri-Tier Gateway Architecture*, *balancing execution speed, prediction accuracy, and computational efficiency* by dynamically selecting the most appropriate processing layer.

D. Acoustic Latency Masking for Improved User Experience

One of the key challenges in hybrid AI systems that rely partially on cloud-based processing is *response latency.* While local command execution can be performed almost instantly, queries that are routed to the cloud-based generative AI layer require additional time for network communication and response generation. This delay may negatively affect the user experience by creating a perceived waiting period after issuing a voice command.

To address this issue, the proposed system incorporates an *Acoustic Latency Masking mechanism* designed to provide immediate auditory feedback to the user. As soon as the system detects and processes the user's voice input, a short *dual-tone acoustic signal* is generated locally. This signal confirms that the assistant has successfully received the command and has begun processing it.

The acoustic feedback is implemented using a lightweight thread-based audio signal that produces a *500 Hz and 700 Hz double-tone chime* immediately after voice recognition is triggered. Because the tone is generated locally, it introduces *negligible computational overhead and near-zero delay.*

This mechanism effectively reduces the *perceived latency* experienced by users during cloud processing operations. Even if the actual response requires several hundred milliseconds for completion, the immediate auditory confirmation creates the impression of a responsive and interactive system.

By incorporating this latency masking technique, the assistant improves the overall *human–computer interaction experience*, ensuring that users remain engaged while the system processes more complex

queries. Such UX-focused optimizations are particularly important in conversational AI systems where responsiveness directly influences usability and user satisfaction.

IV. KEY FEATURES

The proposed J.A.R.V.I.S assistant integrates several advanced modules that enhance system intelligence, usability, and contextual awareness. These features enable the system to perform real-time automation tasks while maintaining efficient human–computer interaction.

A. Dynamic Bilingual Interaction

To improve user accessibility and conversational naturalness, the assistant supports *dynamic bilingual communication* between English and Hinglish. Unlike traditional assistants that rely on direct translation systems, the proposed architecture utilizes *prompt engineering techniq*ues to guide the cloud-based Llama-3 model in generating conversational responses in a natural Hinglish style.

This approach enables the assistant to produce more *contextually appropriate and culturally relevant responses*, thereby improving the user interaction experience, particularly for multilingual users.

B. Screen Awareness Using Computer Vision

The system incorporates *screen-awareness capabilities* that allow the assistant to interpret and analyze the user's active desktop screen. This functionality is implemented using the *Pillow ImageGrab library,* which captures the current screen content and converts it into a Base64-encoded image format.

The encoded image is then processed by *the Llama-3 Vision model,* enabling the assistant to understand visual information displayed on the screen. This feature allows the system to perform tasks such as:

- Analyzing programming errors
- Summarizing articles or documents
- Interpreting visual data on the screen

By combining computer vision with generative AI, the assistant achieves *context-aware interaction with the user's workspace.*

C. Persistent Long-Term Memory

To enable personalized and context-aware interactions, the assistant maintains *persistent conversational memory* using structured context arrays. These arrays store relevant conversation history, user preferences, and previously executed commands.

This long-term memory mechanism allows the system to recall past interactions and provide more *personalized responses during future sessions*. As a result, the assistant gradually adapts to the user's behavior and preferences, improving the efficiency of task execution.

D. Safety Confirmation Mechanism

Certain system-level commands such as *shutdown, restart, or sleep operations* can significantly impact the user's workflow if executed unintentionally. To prevent accidental execution caused by speech recognition errors or model misclassification, the system incorporates a *safety confirmation mechanism.*

When a potentially critical command is detected, the assistant requests a secondary confirmation from the user before executing the operation. This additional verification step enhances system reliability and reduces the risk of unintended system actions.

E. Real-Time Desktop Automation

The assistant provides extensive *desktop automation capabilities* through integration with Python automation libraries. Using modules such as *pyautogui, subprocess, and os*, the system can perform various tasks including:

- Launching applications
- Controlling media playback
- Adjusting system settings
- Sending messages via web-based platforms
- Retrieving information from external APIs

These automation capabilities enable the assistant to function as an intelligent digital companion capable of performing both *informational and operational tasks* within the desktop environment.

V. SYSTEM IMPLEMENTATION & EXPERIMENTAL RESULTS

This section presents the practical implementation and operational evaluation of the proposed *J.A.R.V.I.S desktop AI assistant*. The experiments demonstrate the effectiveness of the Tri-Tier architecture in handling user commands with minimal latency while maintaining high accuracy in intent classification. The

implementation results also illustrate the system's ability to perform desktop automation tasks and analyze screen content using computer vision.

A. Training of the Bi-LSTM Intent Classification Model

The intent classification model was trained using the *TensorFlow and Keras framework***s** on a labeled dataset consisting of various natural language commands representing different system actions. The dataset includes commands related to system control, information retrieval, application management, and conversational interactions. The model was trained for multiple epochs until convergence was achieved. During the training process, the model gradually improved its classification performance by learning contextual relationships between words within command sequences. The training progress, including the epoch count and accuracy metrics, is shown *in Fig. 6.1.*

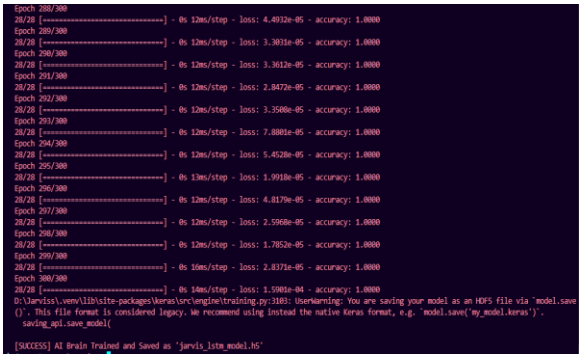

Fig. 6.1. Training process of the local Bi-LSTM intent classifier model.

The results indicate that the Bi-LSTM model successfully learns meaningful patterns in user commands, enabling accurate intent classification during real-time system operation

B. Tri-Tier Routing Mechanism in Operation

Once the model is trained, it is integrated into the runtime system where it processes incoming user commands. The assistant first converts the voice input into textual form and then passes it to *the Bi-LSTM intent classifier*. The model predicts the most likely command category along with its confidence score.

If the predicted confidence value exceeds the predefined threshold *of 85%,* the system executes the command locally through desktop automation modules. Otherwise, the command is forwarded to the *cloud-based generative AI layer* for deeper semantic interpretation.

An example of this routing behavior during runtime is shown in *Fig. 6.2.,* where the assistant correctly identifies a natural language command and maps it to the appropriate system action.

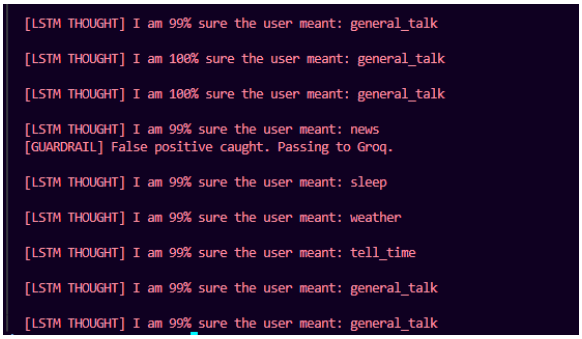

Fig. 6.2. Bi-LSTM intent classification output during runtime showing predicted command categories and confidence levels.

The runtime behavior of the proposed intent classification model is illustrated in Fig. 6.2. After converting the user's speech input into text, the command is passed to the *Bi-LSTM intent classifier,* which analyzes the contextual relationships between words in the command sequence. Based on this analysis, the model predicts the most probable intent category along with a corresponding confidence score. As observed in the figure, the classifier successfully identifies several command categories such as *general_talk, news, sleep, weather, and tell_time*, with confidence values reaching *99–100%,* indicating high prediction accuracy. In cases where the system detects a potential false positive or uncertain prediction, a *guardrail mechanism* is triggered, which forwards the request to the *cloud-based generative AI layer (Groq)* for deeper semantic interpretation. This routing strategy supports the proposed *Tri-Tier architecture,* enabling simple commands to be executed locally while complex or ambiguous queries are handled by the cloud AI module.

C. Graphical Interface and Conversational Interaction

The graphical user interface of the proposed *J.A.R.V.I.S assistant* is illustrated in *Fig. 6.3*. The interface provides a centralized command interaction panel through which users can communicate with the assistant in real time.

Fig. 6.3. J.A.R.V.I.S Desktop Interface and Command Processing Console

The graphical user interface of the proposed *J.A.R.V.I.S desktop assistant* is illustrated in *Fig. 6.3*. The interface provides a centralized command interaction panel through which users can communicate with the assistant in real time. The right-side console displays the ongoing dialogue between the user and the assistant, including the user's query and the system-generated response.

The dashboard also presents system status indicators such as *command processing state, listening mode, and system activity*, allowing users to monitor the operational status of the assistant. The circular visualization labeled *Core Matrix* represents the internal processing activity of the assistant, while the lower panel displays system metrics and operational parameters.

This graphical interface improves the usability of the assistant by offering a *transparent interaction environment,* where users can observe how commands are processed and how responses are generated. The interface therefore serves as a bridge between the *voice-based interaction system and the underlying AI processing modules*, enhancing both user experience and system monitoring during runtime operations.

D. Desktop Automation Through Voice Commands

The assistant also demonstrates strong desktop automation capabilities by executing tasks triggered through voice commands. After recognizing a user request, the system activates predefined automation routines that interact directly with the operating system and web services.

An example of such automation is illustrated in *Fig. 6.4,* where the assistant automatically opens a web browser and prepares an email draft based on a voice command.

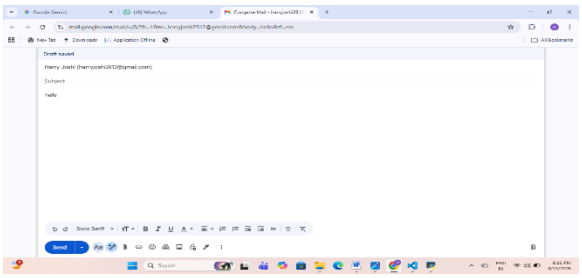

Fig. 6.4. Automated email drafting through voice-command–triggered desktop automation.

The system utilizes automation libraries such as *pyautogui, subprocess, and pywhatkit* to simulate user actions and control applications. This functionality allows the assistant to perform repetitive tasks efficiently, improving productivity and reducing manual effort.

VI. FUTURE WORK

Although the proposed *J.A.R.V.I.S desktop assistant* demonstrates efficient command processing and intelligent task automation using the Tri-Tier architecture, several improvements can be explored in future research. One potential enhancement is the integration of *fully offline speech recognition models,* which would reduce dependency on external APIs and improve system privacy and reliability.

Future work may also focus on incorporating *adaptive contextual learning mechanisms*, enabling the assistant to learn from user interactions and personalize responses over time. Additionally, expanding the system's *multilingual capabilities* beyond English and Hinglish would improve accessibility for a broader user base.

Further enhancements could include *edge AI optimization techniques* to enable efficient execution on low-power devices, as well as the integration of *advanced multimodal interaction capabilities*, such as gesture recognition and enhanced computer vision modules, to provide more context-aware assistance.

VII. CONCLUSION

This paper presented *J.A.R.V.I.S (Just A Rather Very Intelligent System),* a desktop AI assistant designed to improve human–computer interaction through an efficient *Tri-Tier Gateway Architecture*. The proposed system combines rule-based command execution, a local *Bi-LSTM intent classification model, and a*

cloud-based generative AI layer to achieve a balance between low-latency response and advanced contextual understanding.

The architecture enables the assistant to process simple commands locally with minimal delay while routing complex or ambiguous queries to a cloud-based large language model for deeper semantic interpretation. Experimental results demonstrate that the proposed system successfully performs real-time command processing, accurate intent classification, and intelligent desktop automation.

Furthermore, the integration of conversational interaction, screen awareness capabilities, and an intuitive graphical interface enhances both usability and system transparency. The results indicate that hybrid architectures combining *local machine learning and cloud-based AI* can significantly improve the responsiveness and intelligence of desktop assistants.

Overall, the proposed approach provides a scalable and efficient framework for developing *next-generation context-aware desktop AI assistants,* contributing to advancements in intelligent desktop automation and conversational AI systems.

REFERENCES

[1] Chintal, et al. – Next-Gen Desktop Automation with JARVIS: Zero-Wait Voice Control, IEEE Access (2024).

[2] Adha, et al. – Jarvis: Virtual Voice Command Desktop Assistant, Springer Conference Paper (2022).

[3] Natarajan, et al. – An AI Powered Voice Assistant for Enhanced User Interaction (Voice-Bot), Elsevier Procedia (2024).

[4] Gupta, et al. – Development of an AI-Powered Voice Assistant: Enhancing Speech Recognition and User Interaction, IEEE Access (2025).

[5] Anusha M., et al. – Voice Based Virtual Assistant Using AI and ML, Springer Journal (2025).

[6] Huang – Research on the Development of Voice Assistants in the Era of Artificial Intelligence, International Journal of AI Research (2021).

[7] Sinha & Siegert – Improving the Accuracy for Voice-Assistant Conversations in German by Combining Different Online ASR-API Outputs, IEEE Conference Paper (2013).

[8] Klein, et al. – Exploring Voice Assistant Risks and Potential with Technology-Based Users, ACM Digital Library (2025).