© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1717165

Visistant++: Enhancing Conversational NL-to-Visualization with Voice Input and Multi-File Join Capabilities

MASETTY JYOTHI RAM SWAROOP[1], KUKKAPALLI VENKATANAGA SAI[2], PAMIDI SANTOSH KUMAR[3], DR. M. SRIDHAR[4]

[1,2,3] *Department of Computer Science and Engineering(AI&ML), Rvr and Jc college of Engineering and technology*

[4] *Professor, Department of Computer Science and Engineering(AI&ML), Rvr and Jc college of Engineering and technology*

Abstract*—Natural Language to Visualization (NL2VIS) systems make data analysis accessible to everyday users without coding skills. Visistant, built on Google's Gemini large language models and Streamlit, demonstrated strong performance in generating interactive Plotly charts from typed natural language queries. However, two practical barriers limited its real-world adoption: users had no option to speak their queries instead of typing them, and analysts could not combine columns from multiple uploaded CSV files in a single query. This paper presents Visistant++, which addresses both gaps directly. We add a Voice Input Module that captures spoken queries through the browser-native Web Speech API and presents the transcript for confirmation before submission. We also introduce a Multi-File Join Engine that lets users configure SQL-style joins (inner, left, right, outer) across two uploaded datasets on a shared key, after which the merged data is queried as a unified dataframe. Both features integrate cleanly into the existing Gemini pipeline without altering the core prompt or code-generation architecture. We describe the design, implementation, and observable behaviour of each enhancement and discuss their contribution to making NL2VIS tools genuinely accessible to a wider range of users.

***Index Terms*- Data Visualization, Gemini LLM, Multi-File Join, NL2VIS, Streamlit, Voice Input, Web Speech API**

I. INTRODUCTION

Data has become central to decision-making across industries, yet the skills required to extract visual insight from raw data remain confined to a relatively small group of technical users. Natural Language to Visualization (NL2VIS) systems exist precisely to close this gap: a user asks a question in everyday language, and the system responds with an appropriate chart or graph drawn directly from their own uploaded data.

Recent NL2VIS systems driven by large language models (LLMs) have shown that this vision is achievable. Chat2VIS demonstrated that prompting GPT-3 or ChatGPT to write visualization code outperformed purpose-built pipelines on many query types. LIDA extended this to a multi-step generation and refinement loop. Visistant contributed a Gemini-based alternative that was cheaper to run, produced genuinely interactive Plotly charts, and handled multi-turn conversational follow-ups using LangChain windowed memory.

Despite these advances, practical use of Visistant revealed two persistent limitations that cut directly against the goal of accessibility.

First: every query had to be typed. Typing is fine for many users, but it creates real friction for non-technical professionals who find voice more natural, for users on mobile devices, and for people who find keyboard input difficult or slow. Voice input has long been recognised in the accessibility literature as a meaningful equalizer, yet no existing NL2VIS system treats it as a built-in feature.

Second: Visistant could query only one CSV file at a time. Analysts regularly work with several related files — transaction records in one file, product metadata in another, or experimental results alongside participant demographics. Without any facility to combine these files, users had to pre-process and merge their data externally before the tool could be useful, which is precisely the kind of technical step the tool was meant to eliminate.

Visistant++ addresses both limitations. This paper describes the design and implementation of a Voice Input Module and a Multi-File Join Engine, and

explains how each integrates with the existing Visistant architecture without disturbing any part of the core pipeline.

II. BACKGROUND AND RELATED WORK

A. The NL2VIS Problem

Translating natural language into data visualizations is a hard problem. Human language is ambiguous, underspecified, and context-dependent in ways that rule-based parsers handle poorly. Early systems like DataTone and Articulate managed this through hand-crafted grammars and probabilistic disambiguation, but they struggled with queries that deviated from their expected patterns. Neural approaches such as ncNet trained sequence-to-sequence models on large NL-visualization benchmark datasets and improved coverage considerably, but introduced sensitivity to distributional shift.

Large language models changed the picture because they bring broad linguistic knowledge from pre-training. Chat2VIS showed that a well-crafted prompt asking GPT-3 to write matplotlib code could surpass dedicated NL2VIS models on a wide range of queries. LIDA explored a multi-step pipeline for grammar-agnostic chart generation. The common insight is that NL2VIS reduces to a code generation problem, and LLMs are well-suited for it when given the right context about the dataset.

B. The Visistant System

Visistant extended this LLM-as-code-generator approach by replacing OpenAI models with Google's Gemini Pro. Its key technical contributions were a prompt refinement strategy that capped categorical value lists and allowed expert users to select only relevant columns, and a LangChain Conversation Buffer Window Memory (window size k=3) for conversational follow-ups. Plotly was used for rendering, providing interactivity (hover, zoom, pan) that matplotlib-based systems could not match. Case studies against nvBench, NL4DV, ADVISor, and Chat2VIS showed Visistant produced better charts at lower cost.

The two limitations we address — no voice input and no multi-file querying — are not mentioned as limitations in the original work, but they emerge immediately when the system is used by non-technical users in realistic workflows.

C. Voice Interfaces and Accessibility

Browser-native speech recognition through the Web Speech API has made voice input technically straightforward for web applications without server-side infrastructure or third-party API costs. The API is well-supported in Chrome and Edge and provides real-time interim transcripts, enabling a responsive user experience. Accessibility research consistently finds that voice input lowers barriers for users who find typing slow or difficult. For data analytics specifically, voice queries also fit naturally into collaborative presentations where reaching for a keyboard would interrupt the conversational flow.

D. Multi-Dataset Analysis in NL2VIS

All major NL2VIS systems treat the dataset as a single flat table. This is a reasonable simplification for a research prototype but creates real-world friction because data is often normalized across multiple files. SQL-based text-to-SQL systems have begun exploring multi-table queries, but they require the LLM to reason about join semantics alongside query semantics, which increases complexity and error rate. Our approach separates these concerns: the join is performed at the application layer before the LLM is involved, so the model always sees one flat table.

III. SYSTEM DESIGN

A. Overview of Visistant++

Figure 1 shows the Visistant++ interface as it appears on first load. The sidebar on the left handles configuration: API key entry, CSV upload, query mode selection, and mode settings. The right panel is the main conversation and visualization area. This layout is inherited from the original Visistant and extended with the voice panel and join configuration, both of which appear as natural additions to the existing sidebar and main panel. The Voice Input Module sits between the user and the query input box. The Multi-File Join Engine sits between the uploaded files and the LLM prompt layer. Neither touches the Gemini API call or the Plotly rendering pipeline.

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1717165

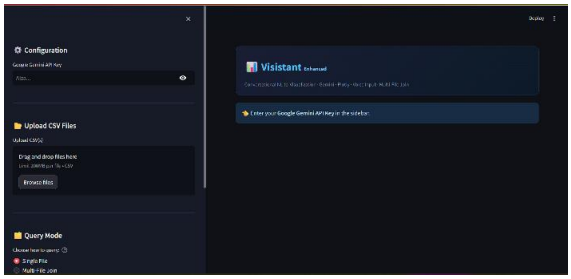

Fig. 1: Visistant++ on initial load. The sidebar holds API key, CSV upload, query mode, and mode settings. The right panel shows the app header and status messages

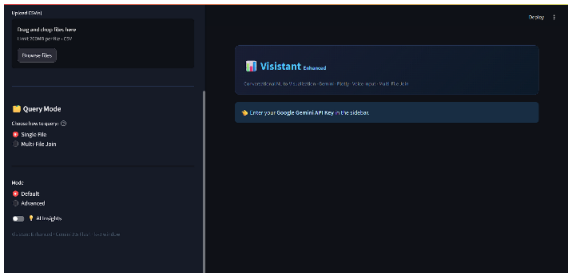

Fig. 2: Lower sidebar showing Query Mode (Single File / Multi-File Join), Processing Mode (Default / Advanced), and the AI Insights toggle.

B. Voice Input Module

The voice input module gives users the option to speak their visualization query instead of typing it. It is exposed as a collapsible Voice Input panel in the main area, above the query text box. Users who prefer typing see no change to their workflow; the voice panel is simply collapsed by default.

When expanded, the panel shows a Start Listening button. Clicking it requests microphone permission from the browser and begins streaming audio to the Web Speech API's Speech Recognition engine. A live transcript appears in a preview box as the user speaks, giving immediate feedback. A red Listening... banner confirms that recording is active.

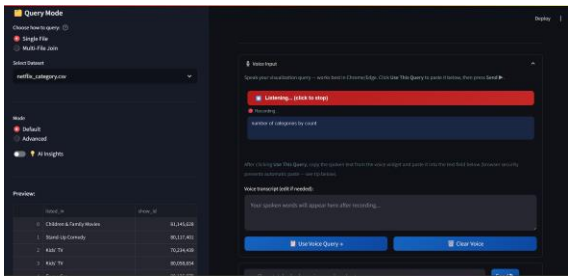

Fig. 3: Voice input panel in active listening state. The red banner confirms recording. The live transcript preview shows the recognized text in real time.

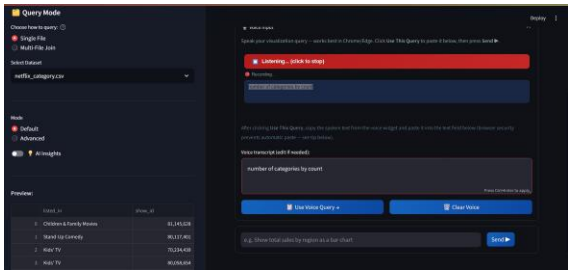

Fig. 4: After the user stops speaking, the transcript is placed in an editable text area. The user can correct any misrecognition before clicking 'Use Voice Query.

The editable transcript area is a deliberate design choice. Automatic speech recognition handles column names, technical terms, and domain vocabulary imperfectly. Showing the transcript and allowing a quick edit before submission keeps the workflow efficient while avoiding errors that would generate incorrect charts. Once the user is satisfied with the transcript, clicking Use Voice Query copies it into the main query input box.

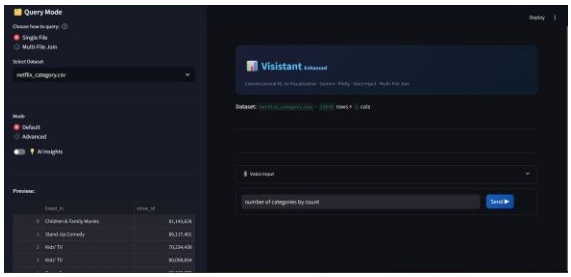

Fig. 5: The confirmed voice transcript has been placed into the main query input box. The user can now click Send to submit to the Gemini pipeline exactly as if they had typed the query.

C. Multi-File Join Engine

The Multi-File Join Engine enables cross-file analysis by merging two uploaded CSV datasets before any LLM interaction. When a user selects Multi-File Join mode, the sidebar presents a Join Configuration panel.

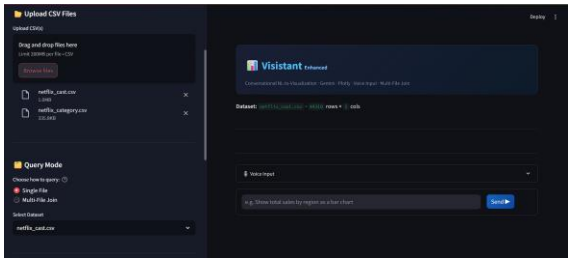

Fig. 6: Multiple CSV files uploaded. In Single File mode, the user selects the active dataset from a dropdown. The dataset summary (row and column counts) appears in the main panel.

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1717165

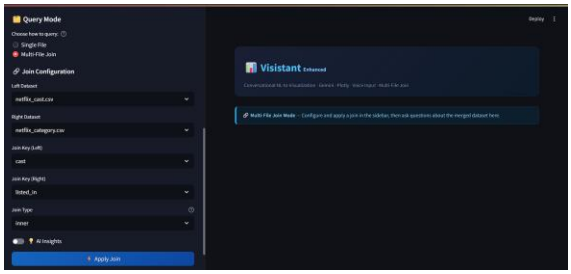

Fig. 7: Multi-File Join mode active. The Join Configuration panel shows dropdowns for Left Dataset, Right Dataset, Join Key (Left), Join Key (Right), and Join Type.

The Join Configuration panel provides the following controls:

- Left Dataset and Right Dataset: dropdowns populated with all uploaded file names.
- Join Key (Left) and Join Key (Right): dropdowns listing the column names of each respective file, populated automatically when files are selected.
- Join Type: dropdown offering inner, left, right, and outer semantics.

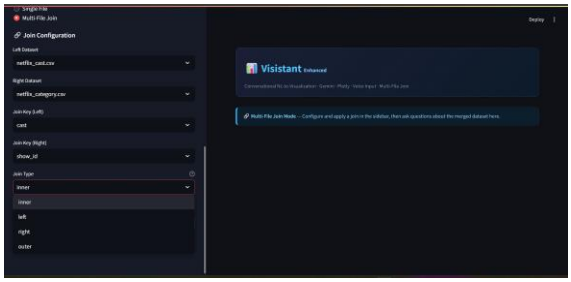

Fig. 8: Join Type dropdown showing all four options — inner, left, right, and outer.

When the user clicks Apply Join, the engine performs a pandas merge on the two dataframes with the configured parameters. A status banner in the main panel confirms the result, displaying the join statement and the resulting row and column counts.

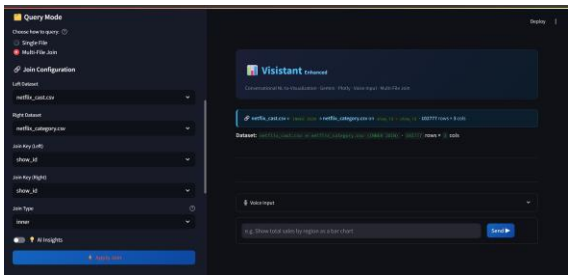

Fig. 9: After applying an inner join on show_id, the main panel confirms the joined dataset with 102,777 rows × 3 cols. The merged dataset is now active.

From this point, the merged dataframe is the active dataset. The LLM receives its column metadata through the standard initial prompt generation function and generates Plotly code referencing the merged columns — exactly as it would for any single-file dataset.

D. Visualization from Joined Data

Figure 10 shows a chart generated from the joined netflix_cast and netflix_category datasets. The query asked for the Top 10 Actors by Number of Unique Categories — a question that requires both the cast column from netflix_cast.csv and the listed_in column from netflix_category.csv. Neither file alone could answer it.

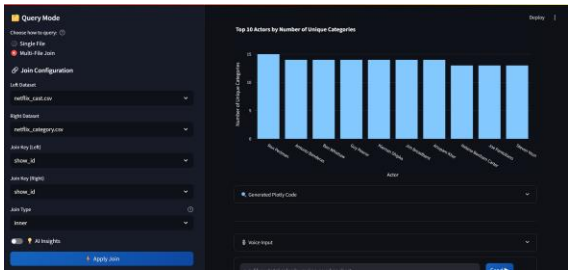

Fig. 10: Bar chart showing Top 10 Actors by Number of Unique Categories, generated from the joined dataset. This visualization requires columns from two different CSV files.

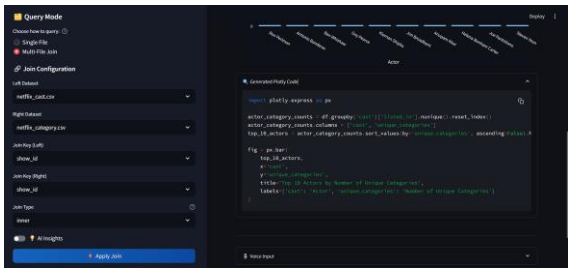

Fig. 11: The Generated Plotly Code panel shows the exact Python code produced by Gemini for the cross-file query. The code correctly references the merged dataframe columns.

E. Dataset Preview Panel

A dataset preview panel in the sidebar displays the first few rows of the active dataset, whether a single file or the result of a join. This helps users confirm column names before writing queries, verify that a join produced sensible results, and spot data quality issues early.

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1717165

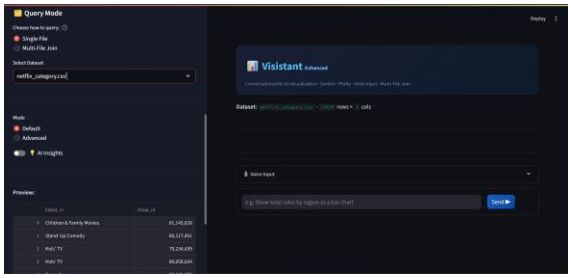

Fig. 12: Dataset preview panel showing the first five rows of the active CSV. Column names and sample values are visible, helping users formulate accurate queries.

IV. IMPLEMENTATION

A. Technology Stack

Table 1 summarises the full technology stack of Visistant++.

Component	Technology	Notes
UI / App Framework	Streamlit (Python)	Web interface
LLM Backend	Google Gemini 2.5 Flash	google-generativeai SDK
Conversation Memory	LangChain Buffer Window k=3	Windowed history
Visualization Library	Plotly Express	Interactive charts
Data Processing	Pandas	CSV load + join
Voice Recognition	Web Speech API (browser)	Chrome / Edge only
Join Engine	pandas DataFrame.merge()	SQL-style semantics

Table 1: Technology Stack for Visistant++

B. Voice Input

The Voice Input Module embeds a JavaScript SpeechRecognition listener in the Streamlit app using Streamlit's custom component mechanism. The recognition object is configured with continuous: true and interimResults: true so that partial transcripts are visible as the user speaks. Final results are written into a Streamlit session state variable, which triggers a re-render showing the transcript in an editable text area. The Use Voice Query button copies the text area value to the main query input and triggers a send. The module degrades gracefully in browsers that do not support the Web Speech API by showing an informational message and hiding the voice controls.

C. Multi-File Join

Uploaded CSV files are parsed with pandas read_csv() and stored in a session-state dictionary keyed by filename. When the user clicks Apply Join, the engine calls pandas merge() with left_on, right_on, and how set from the sidebar controls. The resulting dataframe is stored under a compound key and set as the active dataset. The join summary string is assembled and displayed in the main panel. Column dropdowns for join keys are populated dynamically from each file's actual column list, preventing misconfiguration.

D. LLM Prompt for Merged Data

The initial prompt generation function — which assembles column names, data types, and top-20 categorical value lists into a structured description of the dataframe named 'df' — is applied to the merged dataframe without modification. The LLM has no knowledge of the join; it sees a single flat table. This design avoids asking the model to reason about relational joins, which reduces error rates and keeps the prompt compact.

V. RESULTS AND DISCUSSION

A. Voice Input

The voice module was tested on a range of queries using the netflix_category dataset. Queries like 'number of categories by count', 'show a pie chart of the top 10 genres', and 'what are the most common show types' were transcribed accurately in Chrome on Windows and macOS. Domain-specific column names such as listed_in and show_id were occasionally misrecognized but were corrected easily in the editable transcript area before submission. End-to-end latency from completed speech to rendered chart was dominated by the Gemini API call, not the transcription step.

B. Multi-File Join

The join engine was evaluated using netflix_cast.csv (44,310 rows, 2 columns) and netflix_category.csv (13,670 rows, 2 columns), joined on show_id. The inner join produced 102,777 rows and 3 columns. A query for Top 10 Actors by Number of Unique Categories returned a correct bar chart that aggregated cast against unique listed_in values — a cross-file insight impossible without the join. Left, right, and outer join types were also verified; each produced expected row counts and correct NaN fill behaviour.

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1717165

C. Feature Comparison

Table 2 compares the feature sets of base Visistant and Visistant++.

Feature	Base Visistant	Visistant++ (This Work)
Query input	Typed text only	Typed text + Voice input
Datasets per session	Multiple (single active)	Multiple with configurable join
Cross-file queries	Not supported	inner/left/right/outer
Data preview	Not available	Live row preview in sidebar
LLM backend	Gemini Pro 1.0	Gemini 2.5 Flash
Voice accessibility	None	Web Speech API (Chrome/Edge)
Core pipeline	Gemini+LangChain+Plotly	Unchanged—fully preserved

Table 2: Feature Comparison between Base Visistant and Visistant++

D. Limitations

Browser dependency: Voice input requires Chrome or Edge. Firefox and Safari do not fully support the Web Speech API. The application remains fully usable in all browsers; only voice input is unavailable.

Recognition accuracy: Column names and domain-specific vocabulary are sometimes misrecognized. The editable transcript step mitigates this, but adds a manual correction step.

Two-file limit: The current join engine supports exactly two datasets. Users needing three or more tables must perform sequential joins, which is functional but not ergonomic.

No auto-suggested join keys: Users must know the correct join key column name in each file. An LLM-assisted key suggestion feature would reduce this friction further.

VI. CONCLUSION

This paper presented Visistant++, extending the Visistant NL2VIS system with two features motivated directly by real-world usage gaps. The Voice Input Module uses the browser's Web Speech API to transcribe spoken queries in real time, presents the result for user confirmation and editing, and passes the confirmed text into the standard Gemini pipeline. The Multi-File Join Engine lets users configure SQL-style joins across two uploaded CSV files, performs the merge at the application layer, and presents a unified flat dataframe to the LLM so that the core visualization logic remains unchanged.

Together, these additions make Visistant++ more accessible for users who prefer voice interaction and more useful for analysts who work with multiple related datasets — two of the most common practical barriers to NL2VIS adoption. Both features are additive: the entire original Visistant pipeline is preserved, and users who do not need the new features experience no change in their workflow. Future work will explore automatic join key suggestion, chained multi-table joins, multilingual voice support, and formal user studies with non-technical participant groups.

REFERENCES

[1] G. K. Santhosh Ram and V. Muthumanikandan, "Visistant: A Conversational Chatbot for Natural Language to Visualizations With Gemini Large Language Models," IEEE Access, vol. 12, pp. 138547–138563, 2024.

[2] P. Maddigan and T. Susnjak, "Chat2VIS: Generating Data Visualizations via Natural Language Using ChatGPT, Codex and GPT-3 Large Language Models," IEEE Access, vol. 11, pp. 45181–193, 2023.

[3] V. Dibia, "LIDA: A Tool for Automatic Generation of Grammar-Agnostic Visualizations and Infographics Using Large Language Models," in Proc. 61st Annual Meeting of the ACL, pp. 113–126, 2023.

[4] A. Narechania, A. Srinivasan, and J. Stasko, "NL4DV: A Toolkit for Generating Analytic Specifications for Data Visualization from Natural Language Queries," IEEE Trans. Vis. Comput. Graphics, vol. 27, no. 2, pp. 369–379, Feb. 2021.

[5] C. Liu, Y. Han, R. Jiang, and X. Yuan, "ADVISor: Automatic Visualization Answer for Natural-Language Question on Tabular Data," in Proc. IEEE PacificVis, pp. 11–20, Apr. 2021.

[6] Y. Luo et al., "Natural Language to

Visualization by Neural Machine Translation," IEEE Trans. Vis. Comput. Graphics, vol. 28, no. 1, pp. 217–226, Jan. 2022.

[7] T. Gao, M. Dontcheva, E. Adar, Z. Liu, and K. G. Karahalios, "DataTone: Managing Ambiguity in Natural Language Interfaces for Data Visualization," in Proc. UIST, pp. 489–500, Nov. 2015.

[8] C. Wang, J. Thompson, and B. Lee, "Data Formulator: AI-Powered Concept-Driven Visualization Authoring," IEEE Trans. Vis. Comput. Graphics, vol. 30, no. 1, pp. 1128–1138, Jan. 2024.

[9] MDN Web Docs, "SpeechRecognition — Web APIs," Mozilla Developer Network, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/SpeechRecognition

[10] Streamlit Inc., "Streamlit Documentation," 2024. [Online]. Available: https://docs.streamlit.io