Universal Library of Engineering Technology | Research Article

ISSN: 3064-996X | Volume 2, Issue 1

Open Access | PP: 25-30

DOI: https://doi.org/10.70315/uloap.ulete.2025.0201005

Continuous Testing for State-Based Multistep Voice AI Agents

Vladyslav Budichenko

Chief Technology Officer at Vocaly AI, Miami, FL, USA.

Abstract

This paper presents a strategy for continuous testing of multi-step AI agents based on finite-state machines (FSM) and integrated with large language models (LLMs). The primary focus is on validating the correctness of transitions (both deterministic and LLM-driven) and evaluating the quality of responses, including testing formats such as voice-to-voice, voice-to-text, and text-to-text. It is demonstrated that the state-based approach, where each FSM node is treated as an independent "module" for testing, allows for error localization and effective application of quality metrics (e.g., G-Eval) in a continuous mode. Additionally, methods for integrating a knowledge base (RAG) into test scenarios and organizing integration and end-to-end (e2e) testing are discussed. The proposed practical recommendations are illustrated through the example of "call booking" and can be extended to more complex voice and text-based dialogue systems.

Keywords: *Continuous Testing, Multi-Step LIM Agents, Finite-State Machines, State-Based Testing, Voice-To-Voice / Voice-To-Text / Text-To-Text, G-Eval, Knowledge Base / RAG, Unit Testing, Integration Testing, End-To-End (e2e) Testing.*

INTRODUCTION

Modern dialogue systems that use large language models (LLMs) for generating multi-step responses are becoming increasingly complex and require rigorous quality control (Brown et al., 2020; Zhao et al., 2023). Voice AI agents, which include additional modules for automatic speech recognition (ASR) (Yu et al., 2016) and text-to-speech (TTS) synthesis (Dutoit et al., 1997), hold a particularly prominent place within this domain. However, as interaction scenarios grow more intricate—requiring the system to traverse multiple states (nodes) and consider various dialogue branches—the risks of error accumulation at individual steps increase.

To formalize the logic of such multi-step behavior, finite-state machines (FSMs) are often employed, building on the classical ideas of Mealy (1955) and Moore (1956). This approach has already demonstrated its effectiveness in more modern implementations (Gao et al., 2023), where FSMs are integrated with Retrieval-Augmented Generation (RAG) and LLM-generated responses. However, a persistent challenge has been the continuous evaluation of two key aspects:

- **Correctness of transitions** between states (nodes/edges), including deterministic rules (e.g., a condition like "age > 18") and transitions driven by model outputs (so-called prompt-based transitions).
- **Quality of responses,** since even with the correct selection of the next node, an LLM might generate irrelevant text. In such cases, metrics like G-Eval (Liu, 2023) or simpler binary checks are useful.

Continuous testing is a strategy that involves running a series of tests at every stage of FSM-agent development. This approach helps identify issues promptly without waiting for errors to accumulate and simplifies problem identification in multi-step scenarios.

Hence, for voice AI agents, three testing formats — voice-to-voice, voice-to-text, and text-to-text—are considered; in many cases, text-to-text remains the most practical for frequent checks, as it avoids the overhead of speech modules.

This article explores the general principles of organizing continuous testing for FSM-based agents, combining LLMs with knowledge bases and defined transition logic. We begin by reviewing interaction formats (voice/text) and testing levels (unit, integration, e2e), then outline the key aspects of the state-based testing approach, focused on nodes and transitions. Finally, we present an example scenario that illustrates how to design test cases and evaluate response quality. Our goal is to demonstrate how continuous testing facilitates the maintenance and development of such agents, minimizing the risk of error accumulation at individual states and ensuring dialogue coherence in complex voice interaction scenarios.

OVERVIEW OF TESTING FOR FSM AGENTS

The use of FSMs in building multi-step agents (Mealy, 1955;

Citation: Vladyslav Budichenko, "Continuous Testing for State-Based Multistep Voice AI Agents", Universal Library of Engineering Technology, 2025; 2(1): 25-30. DOI: https://doi.org/10.70315/uloap.ulete.2025.0201005.

Moore, 1956) allows for explicitly regulating transitions between states, thereby enhancing the transparency of dialogue logic (Wu et al., 2024). However, achieving high system reliability requires consideration of specific testing aspects: various input and output formats (voice or text) and several levels of test coverage—from targeted checks of individual nodes to complete end-to-end scenarios (Allouch et al., 2021).

In practice, there are three main types of testing for voice agents:

1. **Voice-to-voice:** Both input and output are in audio format. This method is the most realistic, as it tests not only FSM transitions and LLM-generated responses but also the accuracy of speech modules. However, the high cost of running and maintaining such tests makes supporting these types of tests challenging.
2. **Voice-to-text:** The system receives audio input but returns text output. This method tests the ASR and FSM logic while excluding TTS, thus reducing complexity. However, it still demands careful upkeep and is somewhat less realistic than voice-to-voice.
3. **Text-to-text:** Both input and output are textual, bypassing speech recognition and synthesis. This format is simpler to automate and maintain, making it suitable for continuous integration pipelines, as it reduces execution time and avoids the overhead of speech modules (Allouch et al., 2021).

From the perspective of depth and coverage, testing can be categorized into three classical types (Joshi, 2024):

1. Unit tests

These tests focus on a single node (Node) and/or specific transition (Edge) in isolation from the rest of the dialogue structure. For example, if there is a node responsible for checking available slots, a unit test might send the input "I want to know about available slots" and verify whether the appropriate transition occurs and a correct response is generated. This granular approach simplifies diagnostics and enables frequent test runs in a continuous testing setup.

2. Integration tests

These tests emphasize the interaction between several neighboring nodes and their transition chains. A typical example might involve the sequence: "check schedule" → "offer a list of slots" → "confirm selection". Integration testing ensures that the dialogue progresses from one stage to the next without logical gaps or incompatibilities.

3. End-to-End (e2e) tests

These tests simulate a complete user scenario from start to finish (Mealy, 1955; Moore, 1956). It simulates a specific dialogue testing scenario from start to finish, capturing all transitions and intermediate results to ensure the conversation proceeds as expected.

To illustrate how to combine formats and levels, Table 1 below provides a summary.

Table 1. Formats and evaluation levels

Format / Level	Unit	Integration	End-to-End
Voice-to-voice	Rarely used due to the high cost and complexity of generating/recognizing audio for a single node.	Used for testing multiple voice modules (ASR/TTS) together but requires significant resources.	The most realistic but most expensive scenario is difficult to maintain.
Voice-to-text	Possible for focused ASR analysis (one node with recognition) but has limited application.	A good compromise: tests several connected nodes and ASR functionality, avoiding speech synthesis.	Applicable for realistic dialogue testing, though the final response remains text-based rather than voiced.
Text-to-text	The most convenient mode for local (state-based) node testing. Fast and inexpensive.	Evaluates chains of nodes without speech factors.	The entire dialogue is emulated in text form: fast and simple but lacks testing of speech modules.

In addition to selecting the testing format and level, understanding the internal structure of the FSM is critical. In some systems, state transitions depend on simple conditional deterministic if-else rules, while others rely on model-driven outputs (LLM-driven transitions). The latter requires a specialized testing approach since model responses can vary depending on prompts and inherent randomness (Wu et al., 2024).

Additionally, certain systems incorporate a knowledge base for retrieving factual information. Testing should include scenarios with different knowledge data to validate the correct impact on the agent's dialog processing.

Overall, testing format (voice/text), coverage level (unit/integration/e2e), and presence of a knowledge base create a multidimensional testing space. By prioritizing frequent text-to-text unit and integration tests and using more resource-heavy voice e2e checks only occasionally, teams can efficiently identify issues without overloading infrastructure.

CONTINUOUS TESTING APPROACH

In the context of multi-step agents based on FSM and LLM, continuous testing plays a key role. Its essence lies in quickly running tests at every stage of development (adding new nodes, adjusting transition logic, or updating the model), identifying regressions, and preventing the accumulation of minor errors in complex scenarios.

State-Based Testing

Unlike single-prompt chatbots, which only test "prompt → response," FSM agents decompose logic into multiple nodes (States). In state-based testing, each state is treated as a "module" where we can:

- Define input data: for instance, a fragment of dialogue history in text form (e.g., a user utterance), key variables indicating previously gathered information, and knowledge base data (Wu et al., 2024).
- Specify expected behavior: determining whether the agent should remain in the current state or transition to a specific node.

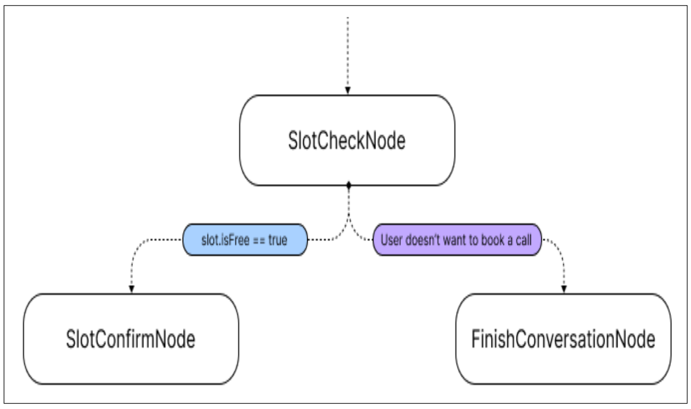

Fig. 1. An example of node testing

Figure Explanation:

- SlotCheckNode is the main state, receiving a user request to book a call for a particular time slot.
- If the condition slot.isFree == true is met (blue condition, representing a deterministic check), and the system transitions to SlotConfirmNode.
- If the user indicates they do not want a booking (purple condition, representing an LLM-based interpretation), the transition leads to FinishConversationNode.

Suppose we have a minimal agent that only does two main tasks: (1) verify the user's desired slot, and (2) either confirm the booking or finish the conversation. The user's utterance might be "Can I book a call for Friday at 5:00 PM?" If slot.isFree == true, we expect a deterministic transition to SlotConfirmNode with a confirmation-like response. Otherwise, the node should provide an alternative recommendation if the user still wants to book another slot.

In a unit test, we simulate precisely this input alongside an internal flag (slot.isFree = true or false) to verify two outcomes:

1. **Transition correctness** — does the agent go to SlotConfirmNode, remain in SlotCheckNode, or proceed to FinishConversationNode?
2. **Response alignment** — does the generated text match the anticipated scenario (e.g., "Yes, 5 PM is free, do you want to confirm?" or "We have no availability at that time.")?

Such testing greatly simplifies error diagnosis and avoids the cost of a full end-to-end check (Joshi, 2024). Since state-based tests only simulate a single entry point (as opposed to the entire conversation), they are easy to integrate into a CI pipeline. Moreover, text-to-text mode is especially useful here: it not only lowers testing overhead by removing ASR/TTS concerns but also makes it simpler for developers to maintain these tests. By varying the input — such as using different user-utterance styles or toggling slot.isFree — we can cover multiple dialogue branches with minimal overhead.

Response Quality Scoring

Even when the FSM correctly selects the next node, the LLM-generated response may be irrelevant, inaccurate, or overly generic. Traditional statistical metrics (BLEU, ROUGE, METEOR) are often inadequate for free-form responses. As a result, LLM-based methods like G-Eval (Liu, 2023) are becoming increasingly popular, where a "judge" model evaluates another model's responses based on criteria such as "accuracy," "relevance," "completeness," and so on.

Simple Binary Check

In straightforward cases, FSM agent developers can define a set of keywords or expected phrases that reliably indicate a correct response. For instance, if a user asks, "When is the call available?" the response should include either a time or a negative statement. Such checks are easy to automate but fail to capture nuances of semantic completeness and are unsuitable for more "creative" responses.

Using G-Eval or Similar Tools

A "judge" prompt might read, "Is the proposed answer factually consistent with the user's request?" The judge model then outputs a numerical score or label (e.g., 1–5). Although this approach is flexible—allowing you to adapt different criteria at each node—it adds cost, given the extra calls to the LLM, and requires carefully crafted judge prompts (Liu, 2023).

This method allows dynamic adaptation of criteria to a specific FSM node, assessing factual correctness (checking for hallucinations), appropriateness of tone, or level of detail.

Typically, scoring is invoked automatically during state-based testing: after generating a response within the node, the system submits it for evaluation and compares the score to a threshold. Failing to meet that threshold marks the test as failed, prompting further investigation.

Integration and End-to-End (e2e) Tests

Even with regular use of state-based testing, there is a possibility that a sequence of nodes may not interact as expected. Therefore, integration and end-to-end (e2e) tests are conducted (Mealy, 1955; Moore, 1956) to emulate a sequence of transitions across multiple nodes.

Integration tests typically encompass a small set of connected nodes, for example: "query available slots → return a schedule → select a slot → confirm." The goal is to ensure that each node in this mini-chain correctly hands off control to the next, without logical gaps or missing data. At this level, a text-to-text format is frequently sufficient, focusing on FSM logic and any knowledge base lookups rather than speech processing.

End-to-end (e2e) tests simulate the entire user conversation, from the first utterance to the final state. In a voice AI scenario, this can involve voice-to-voice interaction, wherein audio input goes through ASR, responses go through TTS, and all relevant FSM transitions are traversed. Such e2e tests are more "realistic" but also significantly more resource-intensive.

Through this two-tiered strategy — selective state-based checks plus comprehensive multi-node (integration/e2e) scenarios — the system achieves rapid feedback on individual nodes while still verifying multi-step coherence. The next section demonstrates these principles in a concrete scenario, focusing on a straightforward "call booking" example.

EXAMPLE TESTING SCENARIO AND PRACTICAL NUANCES

Let us consider a hypothetical scenario in which a user aims to find an available slot for an appointment (a "booking") and, if desired, confirm that booking. The same agent could also handle general inquiries, redirecting such questions to a frequently asked questions (FAQ) node. Below, we first describe the structure of our test cases — inputs, nodes, and expected outcomes — then outline approaches to testing in various formats, followed by considerations for data usage and concluding recommendations.

We begin with the structure of the test cases (fig. 2).

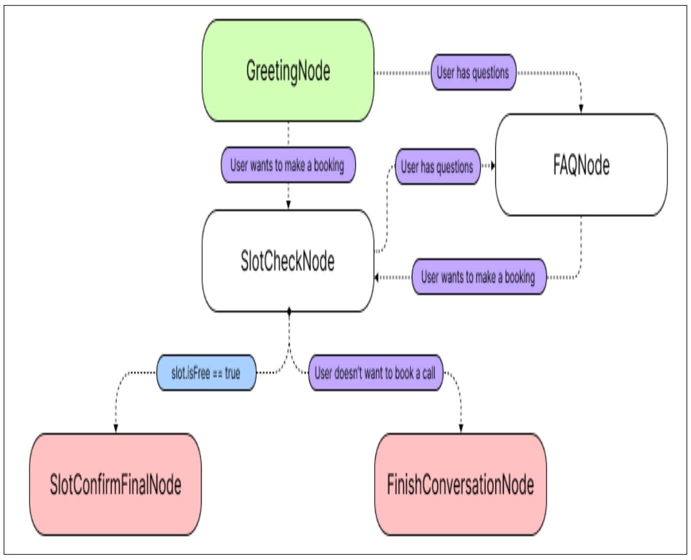

Fig. 2. The structure of the test cases

As shown above:

- **GreetingNode** (in green) is the initial node where the agent greets the user.
- **SlotCheckNode** handles checking if a desired slot is available for the booking. It is reached when the user wants to make a booking (purple arrow: an LLM-based interpretation).
- **FAQNode** handles general questions. It is triggered via a purple arrow from either GreetingNode or SlotCheckNode if the user's query is not about scheduling.
- **SlotConfirmFinalNode** (in pink) is where the agent finalizes a booking if slot.isFree == true (blue arrow: a deterministic check).
- **FinishConversationNode** (in pink) is reached if the user decides not to book.

Following Fig. 2, the primary nodes of interest for the booking flow are:

- **GreetingNode:** The conversation starts here.
- **SlotCheckNode:** The user specifies a desired time for a call; the system checks if it is free.
- **SlotConfirmFinalNode:** If the slot is indeed free, the agent confirms the booking and finalizes.
- **FinishConversationNode:** Ends the dialogue if the user doesn't want a call or has completed their request.

(Note: The FAQNode is present for users who "have questions," but in this example, we focus on booking transitions.)

Transitions occur as follows: from GreetingNode to SlotCheckNode if the user's intent is to make a booking (purple arrow, LLM-based). In SlotCheckNode, a deterministic condition checks whether slot.isFree == true (blue arrow). If so, the system moves to SlotConfirmFinalNode, prompting a confirmation message. Otherwise, it might propose alternatives or move to FinishConversationNode if the user decides against booking. Typical inputs for test cases include:

1. *"I'd like to book a call on Friday at 5 PM."* (The slot is free; user is guided to confirm.)
2. *"Do you have anything on Saturday?"* (No slot available; alternatives or finishing the call might be offered.)
3. *"I changed my mind. I don't need the booking."* (User abandons the request, ending the session.)

Expected outcomes involve verifying (1) transition logic — does the agent arrive at the correct node and (2) response content — does it confirm the slot, decline, or redirect to a different node (like FAQNode or FinishConversationNode)?

To illustrate possible test cases, Table 2 matches each example input with the system's desired response and the node to which it should transition next.

Table 2. Expected outcomes for input data

Input statement	Result/response	Transition (next node)
"I want to book a call for Friday at 5:00 PM."	"Yes, Friday at 5:00 PM is available. Would you like to confirm?"	Slot Confirm Final Node (blue arrow)
"Is there any time on Saturday?"	"Saturday isn't available; we can offer Friday at 4:00 PM instead."	SlotCheckNode (stay in node)
"I changed my mind, I don't need the booking."	"Understood, canceling your request. Have a good day!"	FinishConversationNode

To show how transitions and text responses might be validated:

1. **User**: "Any availability on Saturday?"

   System (slot is not free): "Saturday is unavailable; how about Friday at 4 PM?" *(Remain in SlotCheckNode, user can confirm or decline)*

2. **User**: "Yes, let's do Friday at 4:00 then."

   (The system sees a positive booking request. If slot.isFree == true, the transition goes to SlotConfirmFinalNode with a success response, finalizing the booking.)

To evaluate the agent's responses using a Large Language Model (LLM), we can use a scoring system where the LLM evaluates each response based on predefined criteria like accuracy, relevance, and politeness. For example, consider the scenario where the user asks, "Any availability on Saturday?". The expected response is, "Saturday is unavailable; how about Friday at 4 PM?". However, the agent's actual response is, "Saturday is fully booked. Would Friday at 4 PM work for you?". To assess this response, the LLM is given tailored prompts for each criterion. For accuracy, the prompt could be: "On a scale of 0 to 5, how accurately does this response inform the user about Saturday's availability?" The LLM might assign a score of 5/5 if the response correctly communicates that Saturday is unavailable. For relevance, a prompt such as "On a scale of 0 to 5, how relevant is the suggested alternative time to the user's query?" might result in another 5/5, as the response appropriately suggests Friday at 4 PM. For politeness, the prompt "On a scale of 0 to 5, how polite and professional is the tone of this response?" may yield a 4/5 due to its courteous phrasing.

The total score is then calculated by averaging the individual scores or applying a weighted formula if certain criteria are prioritized. In this case, the response scores (5+5+4)/3 = 4.67, indicating strong overall performance. A threshold score is pre-defined during test setup (e.g., 4.0 out of 5). If the total score is below this threshold, the test fails, signaling that the response requires revision. For instance, if the agent's response instead omitted the suggestion of an alternative time, the relevance score might drop to 2/5, and the overall score could fall to 3.67, triggering a test failure.

By employing these methods — whether focusing on text-based checks, partial flows, or full voice-based interactions — teams can detect logic flaws early and ensure overall coherence in more extensive scenarios.

CONCLUSION

This article examined approaches to testing voice AI agents built on FSM and modern LLMs. Continuous testing plays a crucial role in this context, as even minor changes in node structures or transition logic can lead to unexpected failures at later stages of the dialogue. We demonstrated that the most effective solution combines state-based tests focused on specific nodes with more extensive integration and end-to-end (e2e) scenarios.

Particular emphasis was placed on various interaction formats: voice-to-voice, voice-to-text, and text-to-text. While the full voice format (voice-to-voice) provides the most realistic scenario, text-to-text testing proves optimal for regular CI pipeline runs due to reduced costs and simplified infrastructure.

For response quality evaluation, we discussed basic (template-based) checks and LLM-as-a-judge approaches. These tools enable dynamic assessment of the relevance and accuracy of generative responses but increase testing costs and require careful prompt design. Using the slot booking example, we demonstrated how transition checks, test inputs for different scenarios (available/unavailable slots), and knowledge base integration mechanisms work together.

Thus, the proposed approach provides a practical toolkit for continuous testing of multi-step voice and text dialogue systems. Future developments include the automatic generation of test scenarios based on real logs, expanding the range of quality metrics (e.g., toxicity, unwarranted model "hallucinations"), and systematically addressing security and privacy concerns during testing.

REFERENCES

1. Brown, T. B. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165.

2. Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., ... & Wen, J. R. (2023). A survey of large language models. arXiv preprint arXiv:2303.18223.

3. Yu, D., & Deng, L. (2016). Automatic speech recognition (Vol. 1). Berlin: Springer.

4. Dutoit, T. (1997). An introduction to text-to-speech synthesis (Vol. 3). Springer Science & Business Media.

5. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., ... & Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997.
6. Liu Y. et al. G-eval: Nlg evaluation using gpt-4 with better human alignment //arXiv preprint arXiv:2303.16634. – 2023.
7. Wu, Y., Yue, T., Zhang, S., Wang, C., & Wu, Q. (2024). StateFlow: Enhancing LLM Task-Solving through State-Driven Workflows. arXiv preprint arXiv:2403.11322.
8. Allouch, M., Azaria, A., & Azoulay, R. (2021). Conversational agents: Goals, technologies, vision and challenges. Sensors, 21(24), 8448.
9. Joshi V., Band I. Disrupting Test Development with AI Assistants //arXiv preprint arXiv:2411.02328. – 2024.
10. Wu, B., Chen, G., Chen, K., Shang, X., Han, J., He, Y., ... & Yu, N. (2024). AutoPT: How Far Are We from the End2End Automated Web Penetration Testing?. arXiv preprint arXiv:2411.01236.
11. Mealy, G. H. (1955). A method for synthesizing sequential circuits. The Bell System Technical Journal, 34(5), 1045–1079.
12. Moore, E. F. (1956). Gedanken-experiments on sequential machines. Automata studies, 34, 129–153.