Stagent: A Finite-State Machine Architecture for Traceable LLM Agents

Lucas T. Bandeira[1], **Daniel Alves da Rocha**[1], **Eryck Silva**[1], **Julio C. dos Reis**[1]

[1]Universidade Estadual de Campinas (UNICAMP)

{l322979,d247024}@dac.unicamp.br,
eryck@unicamp.br, jreis@ic.unicamp.br

Abstract. *Autonomous agents extend LLMs to multi-step tasks by combining reasoning, intermediate decisions, and tool use. However, many LLM-based workflows remain implicit execution chains, making failure localization, recovery control, and intermediate evaluation difficult. This investigation presents* ***Stagent****, a finite-state machine architecture that models agent execution as explicit states and transitions, enabling workflow steps to be inspected and redirected after failures. For experimental purposes, we specialize Stagent for tabular document diagnosis as CsvStagent, obtaining valid Finite State Machine (FSM) traces in all executions and reaching* 0.963 *task success. Our findings show that deterministic diagnostic context and FSM-based verification support traceable diagnosis and future self-healing.*

1. Introduction

Large Language Models (LLMs) have been increasingly used to address tasks across different domains [Bahroun et al. 2023, Jo et al. 2023], largely due to their ability to process and generate text. When integrated with external tools, LLMs can be applied to complex, multi-step tasks [Bendinelli et al. 2025, Cho et al. 2026], motivating the use of agents as autonomous entities designed to understand and interact with an environment to reach a specific goal [Konolige and Nilsson 1980]. In this context, LLM-based agents use LLMs as their "brain" [Sumers et al. 2024], extending their memory, reasoning, and decision-making capabilities through Generative AI [Mohamed et al. 2025, Silva et al. 2025].

However, LLM-based systems still face challenges in complex tasks that require fault tolerance [Lampert et al. 2019, Zhou et al. 2025] and robust tabular data processing [Choudhury et al. 2025, Cho et al. 2026]. Recent studies corroborate that strategies like single-prompting (e.g., the ReAct model) [Yao et al. 2023b] are insufficient, especially given that generated errors are propagated and difficult to trace to their origin. In this context, Finite State Machines (FSMs) [Gomaa 2011] provide a promising abstraction for structuring LLM-based tasks into explicit states, enabling sub-task orchestration to either guide their solution via states [Wu et al. 2024] or use each state to store and learn past reasoning processes [Liu et al. 2024].

This study presents **Stagent**, an LLM-based agent architecture that employs the concepts of FSM. Our work was inspired by StateFlow [Wu et al. 2024] and State Machine of Thoughts [Liu et al. 2024]. Our approach complements these studies by expanding the intersection between FSM and LLM-based agents. By combining the autonomous capabilities of agents with the structured architecture of states, Stagent offers a deeper

understanding of how complex tasks are solved and provides ways to ensure traceability in task execution.

To conduct a preliminary evaluation, we designed a Stagent instance to support structural diagnosis and recoverability assessment for tabular data processing, a recurrent problem in the literature [Choudhury et al. 2025, Bendinelli et al. 2025, Cho et al. 2026]. This work addresses the following research questions: **RQ1**: *How can the concepts of a Finite State Machine be integrated into an LLM-based agent architecture?* and **RQ2**: *How can an agent instance developed in RQ1 support structural diagnosis and recoverability assessment in tabular data processing?*

The evaluation was conducted on Pollock-derived CSV corruptions [Vitagliano et al. 2023] under different answer-generation temperatures. CsvStagent is compared with two single-call baselines: a pure-LLM setting, in which the model receives only the raw CSV content and the diagnostic question, and a same-context baseline, in which the model receives the same deterministic diagnostic context produced by CsvStagent's nodes but without the finite-state execution mechanism. This design isolates the effects of diagnostic context and FSM-based execution, enabling separate evaluation of trace validity, diagnostic success, robustness, and token cost.

This study makes three main contributions: 1) proposes Stagent as a finite-state architecture for structuring LLM-agent execution through explicit states, transitions, verification, and error handling; 2) instantiates this architecture as CsvStagent for controlled structural diagnosis over tabular documents; 3) provides empirical evaluation results showing that FSM-based execution improves traceability and diagnostic robustness relative to single-call LLM baselines, while introducing a measurable computational cost. These results indicate that explicit diagnostic traces are a necessary foundation for future self-healing mechanisms in LLM-based agents.

The remainder of this article is organized as follows. Section 2 presents related work. Section 3 details the Stagent architecture. Section 4 presents the methods employed to evaluate Stagent. Section 5 reports the obtained results while Section 6 discusses them. Lastly, Section 7 presents the conclusion remarks.

2. Related Work

Motivated by the fact that LLMs fail to leverage the current process status before deciding the next step, Wu *et al.* [Wu et al. 2024] developed StateFlow, which models LLMs workflows using an FSM. The goal is to separate the current process step from the actual task-solving, with state transitions defined by predefined heuristics or determined by an LLM. The authors reported that StateFlow achieved higher success rates than other approaches, such as ReAct [Yao et al. 2023b], on SQL and textual game benchmarks.

State Machine of Thoughts (SMOT) [Liu et al. 2024], on the other hand, employs an FSM to register and learn past reasoning processes. To achieve this, each state represents a subtask, while transitions describe how subtasks were solved. Transitions can be bidirectional, indicating successful or failed steps. SMOT was also evaluated in a game scenario, with an observed success rate of 56% over Tree-of-Thoughts [Yao et al. 2023a] approach.

Lampert *et al.* [Lampert et al. 2019] proposed a conceptual fault tolerance model

for multi-agent systems by abstracting multidimensional programming. The authors motivated their study by the fact that these systems are naturally susceptible to faults, given their autonomous nature. In a recent study, Zhou *et al.* [Zhou et al. 2025] further investigated that exception-handling in LLM workflows is often superficial, leading to issues in tracing exceptions that occurred in the execution phase to their roots.

Regarding tabular data, Bendinelli *et al.* [Bendinelli et al. 2025] evaluated an LLM agent that interacts with an IPython shell to detect and correct data corruptions, showing that such agents can handle inconsistencies but often rely on brute-force strategies and struggle with complex multi-line errors. Similarly, TASER [Cho et al. 2026] targets errors arising from tabular data extraction in financial documents, although its recursive prompting and occasional need for human intervention may introduce additional overhead.

Our approach further advances this line of work by treating the FSM not only as a task-solving scaffold or as a memory structure for past reasoning trajectories, but as an execution architecture for LLM-based agents. In Stagent, states define operational responsibilities, while the finite-state controller coordinates transitions, bounds execution, records diagnostic traces, and redirects the workflow when verification or execution failures occur. The evaluation scenario in our experiments explores this design in tabular data processing through structural diagnosis and recoverability assessment, distinct from previous approaches that did not leverage LLM-based agents with FSM.

3. Stagent: A Finite-State Machine Architecture for LLM Agents

Finite-state machines (FSMs) provide an abstraction for modeling systems whose behavior evolves through a finite set of states and event-dependent transitions [Gomaa 2011]. In this model, each state represents a well-defined execution condition, while transitions determine how the system progresses when specific inputs, events, or control conditions are observed. Stagent applies this abstraction to LLM-based agents by externalizing workflow control into an explicit finite-state structure, allowing task execution to be organized as a sequence of bounded operational states rather than as a single implicit generation process.

3.1. Formal Model

For a fixed runtime configuration, we formalize Stagent's control structure as

$$\mathcal{M} = \langle S, s_0, S_F, W, F, \delta \rangle, \tag{1}$$

where S is a finite set of operational states, $s_0 \in S$ is the initial state, $S_F \subseteq S$ is the set of terminal states, and W is the working-memory space. The set $F = \{f_s : W \to W \mid s \in S\}$ contains the state-specific execution functions, while $\delta : (S \setminus S_F) \times W \to S \times W$ defines the transition function. A run is initialized with an input-dependent working-memory configuration $w_0 \in W$.

At execution step t, the current state s_t applies its corresponding function f_{s_t} to the current working memory w_t. For non-terminal states, execution proceeds according to

$$\begin{aligned} \widehat{w}_{t+1} &= f_{s_t}(w_t), \\ (s_{t+1}, w_{t+1}) &= \delta(s_t, \widehat{w}_{t+1}), \end{aligned} \qquad s_t \in S \setminus S_F. \tag{2}$$

The intermediate configuration $\widehat{w}_{t+1}$ represents the state-local update before transition-related metadata and control decisions are applied. The transition function δ evaluates this updated working memory, enforces the allowed transitions and their control conditions, and produces both the next operational state and the resulting working-memory configuration. If $s_t \in S_F$, execution terminates after f_{s_t} produces the final working-memory configuration.

In the Stagent implementation, the *router* is the runtime implementation of δ, rather than an operational state of the FSM. It inspects the updated working memory, applies transition guards and execution limits, records transition-related metadata, and selects a valid next state.

Each f_s defines a well-scoped operational responsibility. Depending on the state, the function may be deterministic or may invoke an LLM using a state-specific prompt, model configuration, and set of supporting tools. For a realized execution trace, LLM and tool outcomes are treated as fixed, so that f_s denotes the corresponding working-memory update. The LLM therefore performs only the reasoning required by the current state. When a state emits a routing recommendation, the router retains final transition selection under explicit finite-state control.

Concretely, the working-memory space W is realized as an execution-scoped typed state shared across the `StateGraph` nodes. It combines task artifacts, such as the query, answer, and verification report, with control metadata, such as state identifiers, error context, counters, and token usage. Each node reads the fields required by its responsibility and emits a partial update that is incorporated into the shared state, making working memory an explicit representation of the evolving execution state rather than relying on implicit LLM context or long-term memory.

3.2. State Structure and Execution

Figure 1 presents the state-transition topology of the base Stagent workflow. The workflow begins at `INIT`, which normalizes the user request into the initial working-memory configuration. Under nominal execution, the agent then progresses through `PLAN`, `CHECK_FILE`, `EXTRACT_CONTENT`, `SEARCH`, `ANSWER`, and `VERIFY` before reaching the terminal `END`. These states define the main processing stages of the base architecture, while the actual transition between them is determined by the transition function δ introduced in Section 3.1.

Conditional transitions allow execution to depart from the nominal path. State or verification failures may trigger transitions to `ERROR`, from which execution can be redirected to an allowed operational state, to `RECOVERY` when task-specific recovery is available, or to termination when continuation is not permitted. Therefore, the FSM supports conditional branches and repeated execution of operational states rather than constraining the agent to a fixed linear sequence. Transition conditions and failure-handling decisions are represented by the corresponding edges in Figure 1. The router validates and applies these transitions but is not itself an operational state of the FSM.

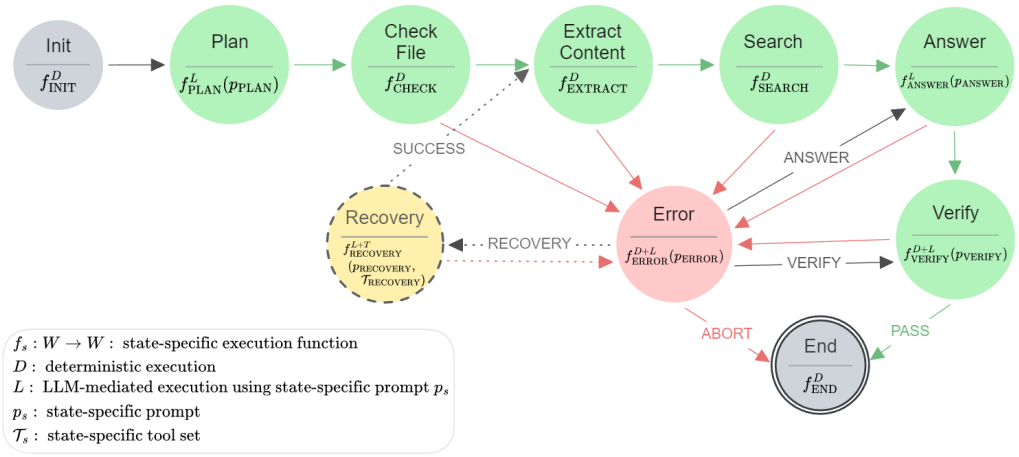

Figure 1. Stagent state-transition model. Each state s executes $f_s : W \to W$ through deterministic (D), LLM-mediated (L), or tool-supported ($\mathcal{T}$) processing. Dashed edges denote optional recovery; the router implements δ and is omitted as a state.

Table 1 summarizes the generic responsibility associated with each state in the base workflow. These responsibilities define Stagent's architectural contract, allowing specialized implementations for different downstream tasks or document formats to replace selected state behaviors without changing the underlying transition model.

The state set was manually designed by decomposing a generic document-agent life cycle into stages with distinct operational responsibilities and meaningful control boundaries. This decomposition was chosen to balance traceability and reuse: a coarser graph decomposition would make intermediate artifacts and failure locations less explicit, whereas a finer decomposition would introduce task-specific operations into the base architecture and reduce its portability. Therefore, the proposed state set is not intended as the only possible FSM decomposition or as a structure learned automatically from data, but as a reusable architectural contract for specialized agents.

Table 1. Base Stagent states and generic responsibilities.

State	Responsibility
`INIT`	Normalize the user request into working memory.
`PLAN`	Produce a high-level task plan.
`CHECK_FILE`	Validate input availability and file constraints.
`EXTRACT_CONTENT`	Extract task-relevant content from the artifact.
`SEARCH`	Select or construct context for answer generation.
`ANSWER`	Generate the task output from the selected context.
`VERIFY`	Check whether the output satisfies task constraints.
`ERROR`	Diagnose failures and propose a handling transition.
`RECOVERY`	Optionally execute task-specific recovery actions.
`END`	Consolidate the final result or terminal error report.

Although the base Stagent architecture includes both `ERROR` and `RECOVERY` states to support conditional deviations from the nominal path, these states serve distinct purposes. The `ERROR` state is part of the control mechanism for handling execution-time failures, aligning with recent work arguing that exception handling in LLM workflows should be explicitly traced rather than treated as superficial post-processing

[Zhou et al. 2025]. In contrast, the RECOVERY state represents an optional capability intended for tasks that require tool-guided correction or deeper artifact processing.

3.3. Traceability and Control

A Stagent execution produces a structured trace that records the evolution of both control state and working memory. For an execution reaching a terminal state at step T, we denote this trace as

$$\tau = \langle (s_0, w_0), (s_1, w_1), \ldots, (s_T, w_T) \rangle. \tag{3}$$

The trace captures the sequence of operational states and their corresponding entry working-memory configurations. In the implementation, the control-state projection of this trace is materialized by state_history, while the returned working memory preserves the accumulated task artifacts, verification outcomes, execution metrics, and failure-handling information. A trace is considered valid when its observed transitions conform to the transition function δ defined in Section 3.1, thereby capturing conformance to the finite-state control model.

Bounded execution is enforced through global transition guards implemented by the router as part of δ. The runtime parameters max_error_count and max_transition_count limit accumulated node failures and excessive state transitions, preventing uncontrolled execution cycles. In addition, the VERIFY node converts outputs that violate the task-specific schema or validation constraints into explicit control events, allowing such failures to be represented and handled within the FSM rather than remaining observable only in the final response.

When execution terminates successfully, the returned working memory retains both the generated answer and the most recent structured verification report, which records the verification checks, final verdict, failed dimensions, and retryability decision. If execution terminates unsuccessfully, the working memory instead retains a structured terminal error report containing the failure context available at termination. Consequently, each run can be evaluated in terms of its task outcome and recorded control-state history.

4. Evaluating CsvStagent: A Tabular Document Specialization

CsvStagent is a Stagent-based implementation that serves as an agent for structural diagnosis of tabular documents (CSV files). CsvStagent treats the file as a raw textual artifact and derives a diagnostic report from observable structural properties, which is then used to produce a structured judgment about whether a structural error is present and whether it appears recoverable.

4.1. CsvStagent Execution Workflow

In the structural diagnosis configuration, CsvStagent preserves the nominal Stagent workflow while adapting the behavior of selected states to CSV-specific analysis. At the implementation level, CsvStagent specializes the state-specific functions associated with CHECK_FILE, EXTRACT_CONTENT, SEARCH, and ANSWER, while preserving the remaining state functions and the transition function δ defined by Stagent. The specialization changes the operational behavior of selected states without modifying the underlying finite-state control model.

The first adaptation occurs in CHECK_FILE, which performs only minimal input validation by checking file availability and input type compatibility. It does not attempt to parse, normalize, or repair the document, since malformed CSV files are themselves the object of analysis. Rejecting a file because it cannot be loaded as a regular table would reduce the task to a data-loading failure and prevent the agent from inspecting the structural evidence needed for diagnosis.

The main specialization occurs in EXTRACT_CONTENT. Rather than converting the CSV file into a dataframe, CsvStagent reads it as a raw textual artifact and derives a structural diagnostic report from observable document properties. This report exposes structural signals that can support the diagnosis, while avoiding oracle information such as the expected pollution type, ground-truth class, or external labels. The SEARCH state then packages this report as the context passed to ANSWER, which produces a structured JSON diagnosis covering the presence of a structural error, its predicted type, its apparent recoverability, and the supporting evidence. Recoverability is modeled as a diagnostic assessment, while file repair remains outside the scope of CsvStagent in this evaluation.

VERIFY checks whether the generated diagnosis follows the expected output format and is supported by the available diagnostic context. If verification succeeds, the terminal artifact combines the diagnosis with the execution trace and associated metadata, allowing each run to be inspected as a finite-state diagnostic trajectory. Otherwise, if a node or verification step fails, the router redirects execution to ERROR, where the failure source and handling decision are recorded before the workflow either retries an allowed state or terminates according to the configured control policy.

4.2. Structural Error Diagnosis Task

We evaluate CsvStagent on a structural diagnosis task over CSV documents. Given an input file and a diagnostic question, the agent must produce a structured JSON diagnosis indicating whether a structural error is present, its predicted category, the supporting evidence, and whether the file appears structurally recoverable. Recoverability is interpreted as the ability to safely correct the structure using regularities already observable in the file, without inventing missing content or altering the intended data semantics. Here, regularities denote recurring structural patterns observable across lines, such as dominant delimiter and field-count distributions, consistent quoting behavior, and stable relationships between header and data rows.

The task focuses exclusively on the organization of the CSV artifact itself. Accordingly, CsvStagent is not evaluated on semantic anomalies, factual inconsistencies, numerical correctness, or downstream performance after loading the file. This delimitation separates structural diagnosis from broader data-cleaning and table-reasoning tasks.

4.3. Dataset and Experimental Protocol

The structural cases are derived from Pollock [Vitagliano et al. 2023], a benchmark designed to evaluate data-loading robustness under systematic CSV pollutions. We use Pollock as a controlled testbed in which known structural corruptions provide reproducible conditions for evaluating diagnostic performance, state transitions, verification failures, and error-handling trajectories.

From this benchmark, we selected a representative subset of cases and mapped each pollution to a two-level diagnostic taxonomy. At the group level, the taxonomy includes no error, file/table-level errors, header delimiter errors, dialect mismatch, row-shape errors, quoting errors, and a fallback category. The fine-grained level preserves the corresponding Pollock-derived structural type for each case, such as missing headers, multirow headers, inconsistent row lengths, delimiter mismatches, or malformed quoted cells. Each selected case is converted into an anonymized input file and diagnostic question, processed by CsvStagent, and evaluated offline using the JSON diagnosis, FSM trace, and execution metadata.

All experiments used the `gpt-oss:20b` model served through Ollama. In CsvStagent, LLM calls were limited to the `PLAN`, `ANSWER`, `VERIFY`, and `ERROR` states. The `PLAN` temperature was fixed at 0.1, whereas `VERIFY` and `ERROR` used 0.0. The `ANSWER` temperature was the only parameter varied across runs, with $T_A \in \{0.1, 0.4, 0.7\}$. Each of the 16 test cases was executed five times at each T_A, yielding 80 executions per temperature, 240 executions per approach, and 720 executions overall. The remaining states, namely `CHECK_FILE`, `EXTRACT_CONTENT`, and `SEARCH`, were deterministic and did not invoke an LLM. Execution was constrained to at most three accumulated errors, 18 FSM transitions, and a LangGraph recursion limit of 50.

We compare CsvStagent with two single-call baselines implemented as direct calls to the same LLM configuration used by CsvStagent's `ANSWER` state. The pure-LLM baseline receives the raw anonymized CSV, diagnostic question, and output schema, without a structural report or FSM control. The same-context baseline receives the same deterministic diagnostic report and `ANSWER` prompt as CsvStagent, but omits the FSM, router, verification, and error-handling stages. Together, they provide minimal and context-controlled reference points for assessing the effects associated with diagnostic context construction and FSM-mediated execution.

4.4. Evaluation Metrics

We evaluate the structural diagnosis task along complementary dimensions: a) diagnostic correctness; b) recoverability; c) execution reliability; and d) computational cost. All metrics are computed at the execution level and then aggregated across repeated runs.

Diagnostic correctness. It measures whether the generated diagnosis matches the expected interpretation of each case. We define structural detection as the ability to determine whether a structural error is present. Classification is then assessed at two complementary levels, the fine-grained Pollock-derived structural type and the broader diagnostic group defined in the experimental taxonomy. We report standard classification metrics, summarizing diagnostic correctness through structural detection accuracy and F1 scores at the specific and group levels.

Recoverability. It is evaluated only for polluted cases with an oracle annotation, measuring whether the agent's estimate matches the expected recoverability policy. Evidence quality is assessed through evidence presence and an automatic evidence-support proxy. Evidence presence checks whether polluted-case diagnoses include a non-empty evidence list, while the proxy additionally requires valid JSON, correct structural error detection, and assignment to the expected diagnostic group. This proxy captures group structural grounding, but does not validate the semantic correctness of individual evidence

statements.

Execution reliability. It measures whether CsvStagent completes the diagnostic workflow in a valid form. A run is considered reliable when it finishes without exception, reaches a successful terminal status, produces a parseable JSON diagnosis, and follows a valid FSM trace as defined in Section 3.3. We also define a composite diagnostic task success metric that is satisfied only when the run is reliable, correctly detects the structural condition, assigns the case to the expected group, and satisfies the automatic evidence-support proxy.

Computational cost. It is summarized by the average number of FSM transitions and total token usage collected from the execution metadata.

5. Experimental Results

This section reports the experimental results according to the evaluation dimensions defined in Section 4.4. Subsection 5.1 discusses workflow completion and FSM trace validity. Subsection 5.2 compares diagnostic and recoverability performance against baselines. Subsection 5.3 analyzes the transition and token overhead introduced by FSM.

5.1. Execution Reliability Results

Table 2 presents CsvStagent execution reliability across the three answer generation temperatures. Across all 240 executions, CsvStagent completed the diagnostic workflow and preserved a valid FSM trace in every run. Although all executions reached a successful terminal status, 20 runs were redirected through `ERROR`, corresponding to intermediate verification failures rather than terminal failures. These runs produced 21 error activations, all originating in `VERIFY`, with one execution at $T_A = 0.7$ requiring two verification and regeneration cycles.

Table 2. CsvStagent execution reliability by answer temperature.

T_A	Runs	Completion	Valid FSM	Error-routed
0.1	80	100%	100%	6 (7.5%)
0.4	80	100%	100%	4 (5.0%)
0.7	80	100%	100%	10 (12.5%)

5.2. Diagnosis and Recoverability Results

Table 3 compares CsvStagent with the two baselines in terms of structural diagnosis, recoverability, and task success. No approach produced false positives for the clean control. CsvStagent achieved perfect structural detection at $T_A = 0.1$ and $T_A = 0.7$, with only one false negative at $T_A = 0.4$, and obtained the best task success at $T_A = 0.4$ and $T_A = 0.7$.

5.3. Computational Cost Results

Table 4 presents that CsvStagent incurs a higher computational cost than the single-call baselines because its diagnosis is produced through multiple finite-state steps rather than a single generation call. Compared with the diagnostic-report LLM baseline, which receives the same deterministic context as CsvStagent but does not execute the FSM workflow, CsvStagent required approximately 2.2 to 2.5 times more tokens. The highest average transition and token counts occurred at $T_A = 0.7$, which is consistent with the larger number of error-routed executions reported in Section 5.1.

Table 3. Structural diagnosis effectiveness by each approach evaluated and answer temperature.

Approach	T_A	Detection	Specific F_1	Group F_1	Recoverability	Task success
CsvStagent	0.1	**1.000**	**0.786**	0.898	**0.933**	0.938
Pure LLM	0.1	0.900	0.570	0.625	0.827	0.750
LLM + Diagnostic Report	0.1	**1.000**	0.776	**0.943**	**0.933**	**0.950**
CsvStagent	0.4	0.988	**0.805**	**0.936**	0.933	**0.950**
Pure LLM	0.4	0.963	0.658	0.765	0.920	0.863
LLM + Diagnostic Report	0.4	**1.000**	0.798	0.910	**0.973**	0.938
CsvStagent	0.7	**1.000**	**0.807**	**0.946**	**0.933**	**0.963**
Pure LLM	0.7	0.963	0.711	0.888	0.920	0.875
LLM + Diagnostic Report	0.7	0.975	0.747	0.881	0.893	0.900

Table 4. Average computational cost per execution.

Approach	T_A	FSM trans.	Tokens
CsvStagent	0.1	7.23	22,912
	0.4	7.15	22,284
	0.7	7.41	25,137
Pure LLM	0.1	–	12,344
	0.4	–	11,255
	0.7	–	10,324
LLM + Diagnostic Report	0.1	–	9,979
	0.4	–	10,062
	0.7	–	9,913

6. Discussion

Stagent addresses RQ1 by separating state-specific execution from finite-state control. Each operational state implements a bounded function f_s, while the router realizes the transition function δ, enforcing transition guards, failure redirection, and execution limits independently of the LLM. This separation makes intermediate artifacts and control decisions explicit and allows task-specific behavior to be specialized without redefining the underlying execution model. Traceability therefore follows from the architecture itself, since executions can be reconstructed from their state sequence and associated working-memory configurations rather than assessed only through the final output.

Regarding RQ2, CsvStagent specializes this model for structural CSV diagnosis while preserving the finite-state control structure. Across all tested temperatures, every execution completed with a valid FSM trace, and verification failures were represented as explicit transitions through `ERROR` rather than as terminal exceptions. This demonstrates that the architecture can expose and handle intermediate failures within the evaluated configuration, although trace validity captures conformity to the FSM and does not imply diagnostic correctness.

The baseline comparisons indicate that deterministic diagnostic context accounts for an important part of the observed effectiveness. At $T_A = 0.1$, replacing raw CSV input with the diagnostic report increased task success from 0.750 to 0.950 and group-level F_1 from 0.625 to 0.943 in the single-call setting, suggesting that compact structural

observations reduce the burden of deriving CSV regularities directly from malformed raw text. The LLM + Diagnostic Report baseline further shows that the additional effects of CsvStagent are associated with finite-state execution, verification, and error handling rather than with richer diagnostic context alone. Although this baseline achieved slightly higher task success at $T_A = 0.1$, CsvStagent performed better at $T_A = 0.4$ and $T_A = 0.7$, which is consistent with verification-driven retries becoming more useful as answer generation becomes more variable.

Remaining errors were concentrated in boundary cases, particularly missing delimiters, row-shape inconsistencies, and malformed quoting, where models often detected the anomaly but confused neighboring fine-grained categories. Recoverability showed a different pattern, with CsvStagent remaining more stable across temperatures than the baselines, although this robustness comes at a higher token cost due to multi-step execution. These findings are limited to controlled structural diagnosis over Pollock-derived CSV corruptions; future work should extend the evaluation to semantic and mixed structural-semantic failures, while also using the generated diagnosis and FSM trace as contextual signals for self-healing and replanning. In this setting, a `RECOVERY` state could apply tool-guided corrections when repair is considered safe, or redirect execution whenever repair cannot be reliably performed.

7. Conclusion

Although LLMs enable autonomous agents to tackle complex multi-step tasks, their implicit execution chains often obscure error tracking. To address this, we proposed Stagent, an LLM-based agent with an FSM architecture that models execution through explicit states and transitions. For an initial assessment, we created an instance, the CsvStagent, for the structural diagnosis of tabular documents. Our evaluation demonstrated that FSM-based execution enhances robustness, achieving 100% execution reliability with valid traces and up to a task success rate of 0.963 by explicitly routing validation failures through an `ERROR` state rather than failing terminally. However, this architectural rigor introduces a tradeoff in resource consumption alongside challenges in fine-grained categorization of borderline structural anomalies. Future work can expand the generic Stagent architecture with traceable diagnostics to achieve dynamic capabilities, tool-guided artifact correction and replanning within a bounded and verifiable execution loop.

Acknowledgments

This study was sponsored by Petróleo Brasileiro S.A. (PETROBRAS) within the project "*Application of Large Language Models (LLMs) for online monitoring of industrial processes*" conducted in partnership with the University of Campinas. This work was also supported by CNPq, Brazil (grant #301337/2025-0).

References

Bahroun, Z., Anane, C., Ahmed, V., and Zacca, A. (2023). Transforming education: A comprehensive review of generative artificial intelligence in educational settings through bibliometric and content analysis. *Sustainability*, 15(17):12983.

Bendinelli, T., Dox, A., and Holz, C. (2025). Exploring LLM agents for cleaning tabular machine learning datasets. In *ICLR 2025 Workshop on Foundation Models in the Wild*.

Cho, N., Fielding, K., Watson, W., Ganesh, S., and Veloso, M. (2026). TASER: Table agents for schema-guided extraction and recommendation. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 5: Industry Track)*, pages 226–252. Association for Computational Linguistics.

Choudhury, M. R., Iyengar, A. I. K. N., Siingh, S., Puranam, S., and Gupta, V. (2025). TABARD: A novel benchmark for tabular anomaly analysis, reasoning and detection. In Christodoulopoulos, C., Chakraborty, T., Rose, C., and Peng, V., editors, *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 21783–21817, Suzhou, China. Association for Computational Linguistics.

Gomaa, H. (2011). *Finite State Machines*, page 151–176. Cambridge University Press.

Jo, E., Epstein, D. A., Jung, H., and Kim, Y.-H. (2023). Understanding the benefits and challenges of deploying conversational AI leveraging large language models for public health intervention. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–16. ACM.

Konolige, K. and Nilsson, N. J. (1980). Multiple-agent planning systems. In *AAAI*, volume 80, pages 138–142.

Lampert, L., Hübner, J., and Zatelli, M. (2019). Tolerância a faltas em sistemas multiagentes multidimensionais. In *Anais do XIII Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações*, pages 230–235, Porto Alegre, RS, Brasil. SBC.

Liu, J., Shuai, J., and Li, X. (2024). State machine of thoughts: Leveraging past reasoning trajectories for enhancing problem solving. *arXiv:2312.17445*.

Mohamed, A. H., Santos, F. A., and dos Reis, J. C. (2025). Enhancing llm agent effectiveness via reflective multi-agent system. In *Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações (WESAAC)*, pages 206–217. SBC.

Silva, E., Santos, F. A., Thompson, P., and dos Reis, J. C. (2025). Llm-powered conversational multi-agent cognitive system for collaborative task solving. In *Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações (WESAAC)*, pages 59–70. SBC.

Sumers, T., Yao, S., Narasimhan, K., and Griffiths, T. (2024). Cognitive architectures for language agents. *Transactions on Machine Learning Research*.

Vitagliano, G., Hameed, M., Jiang, L., Reisener, L., Wu, E., and Naumann, F. (2023). Pollock: A data loading benchmark. *Proc. VLDB Endow.*, 16(8):1870–1882.

Wu, Y., Yue, T., Zhang, S., Wang, C., and Wu, Q. (2024). Stateflow: Enhancing llm task-solving through state-driven workflows. *arXiv:2403.11322*.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., and Narasimhan, K. (2023a). Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023b). React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Zhou, J., Chen, J., Lu, Q., Zhao, D., and Zhu, L. (2025). Shielda: Structured handling of exceptions in llm-driven agentic workflows. *arXiv:2508.07935*.