AGENTSOV: VIRTUAL SENSING FOR LLM MULTI-AGENT SYSTEMS VIA STREAM OF VARIATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) multi-agent systems (MASs) solve complex tasks through interacting agents, yet their reliability is difficult to analyze because failures depend on both individual agents and how information and errors propagate across the system. Existing approaches largely treat this as post-hoc failure attribution, but do not provide a common model across heterogeneous MAS architectures or support broader analysis of information influence and observability. We introduce **AgentSoV**, a virtual-sensing framework that models MAS execution as a stochastic dynamic system through Stream of Variation (SoV). AgentSoV maps heterogeneous trajectories into diagnostic stages, constructs virtual measurements from observable outputs, and learns cross-stage variation propagation with a state-space model. Its representation is architecture-aware in stage dependencies yet architecture-agnostic in analytical language, supporting chain-structured, orchestration-based, and directed acyclic graph (DAG)-structured MASs. Without re-executing the MAS, AgentSoV supports three system-level analyses: *failure attribution* identifies where abnormal variation enters; *stage-input influence* quantifies how strongly intermediate inputs affect downstream outcomes; and *virtual-sensor placement* selects complementary observations under a monitoring budget. Experiments across multiple MAS architectures demonstrate a unified approach to fault localization, information influence, and observability. AgentSoV reframes agentic execution as a dynamic system to model rather than merely text to inspect, providing insights for MAS structure and component design.

1 INTRODUCTION

Large language model (LLM) multi-agent systems (MASs) solve complex tasks by distributing work across specialized agents (Li et al., 2024a; Yang et al., 2026a). These agents may plan, call tools, or verify one another's outputs. However, composing capable agents does not necessarily yield a reliable system. Failures can arise from individual agent errors, but also from system design and ineffective information exchange (Cemri et al., 2026). Intermediate outputs may have little effect on the final result or duplicate information already available elsewhere. They may also introduce erroneous variation that persists through subsequent interactions (Zhang et al., 2026d; Li et al., 2025). MAS reliability is therefore a system-level problem. Effective diagnosis should locate where a failure originates and trace how its effect propagates. It should also reveal which intermediate information matters to the outcome and which parts of the execution need to be observed.

A major line of work studies this problem through failure attribution: given a completed failed trajectory, identify the agent or execution step responsible for the failure (Zhang et al., 2026a). Who&When (Zhang et al., 2025) formalized agent- and step-level attribution over failure trajectories from a large collection of MASs and let LLM judges inspect trajectories. Subsequent approaches improve particular parts of this pipeline by narrowing the scope before localization, training specialized failure tracers, refining trajectory inspection, or replacing direct judging with intervention, counterfactual replay, and spectrum analysis (Ma et al., 2026; Yeh et al., 2026; Li et al., 2026c). For example, on the Who&When benchmark, SDBL (Sun et al., 2026) reports step-level accuracy up to 27.78% on the handcrafted subset and 33.33% on the automatically generated subset, and AgenTracer (Zhang et al., 2026a) reports up to 20.68% and 42.86%, respectively. These results represent substantial progress, but they also illustrate that fine-grained MAS diagnosis remains difficult. Additional related work is in Appendix A.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

More fundamentally, many existing approaches are not architecture-invariant by construction. The meaning of an execution step depends on how the MAS is implemented. In an orchestration-based system, a step may represent an orchestrator-worker interaction, while in a directed acyclic graph (DAG)-structured workflow it may correspond to a node with multiple upstream parents. Methods that reason directly over these execution objects are therefore tied to the underlying trajectory format. Intervention-based methods may also require changes to the replay mechanism, while learned approaches may depend on a new training distribution (Zhang et al., 2026a). Replay-based methods further require access to a re-executable MAS and its environment (Yeh et al., 2026). Although such methods can be adapted across systems, their diagnostic representation remains closely coupled to the execution format. Most of these methods are also designed primarily for failure attribution. They answer where or by whom a failure was introduced, but do not provide a common model for broader system analysis. For example, they do not directly characterize how strongly variation persists downstream. They also do not determine which intermediate inputs or outputs are redundant, nor which should be monitored for effective diagnosis.

These questions suggest a different starting point: an MAS execution can be viewed as a stochastic dynamic system. This viewpoint has a close analogue in multistage manufacturing. Manufacturing systems are highly heterogeneous, i.e., different processes contain different machines, operations, and sensor layouts, yet state-space models provide a common abstraction in which a latent process state evolves across stages, process inputs affect that state, and measurements reveal only part of it. One representative framework is Stream of Variation (SoV), which models how variation introduced at one manufacturing stage propagates through downstream stages and appears in distributed measurements (Shi, 2006; Huang et al., 2000; Zhou et al., 2003). Once the process is represented in this form, concepts from control and systems theory become available: observability asks whether internal variation can be inferred from available measurements, diagnosability asks whether different variation sources can be distinguished, and sensor-design methods ask where measurements should be placed to make the process sufficiently informative for diagnosis. The value of this abstraction is not that every manufacturing process has the same dynamics, but that very different processes can be represented using the same language of *stages, propagated state, inputs, disturbances, and measurements*. A detailed illustration is in Section 2.1.

Building on this connection, we introduce **AgentSoV**, a virtual-sensing framework that models LLM MAS execution through SoV (Figure 1). AgentSoV maps an observed execution into a sequence of diagnostic stages, constructs process inputs and virtual measurements from observable execution information, and fits state-space dynamics that characterize cross-stage variation propagation. Once learned, the same compact model supports failure attribution, stage-input influence analysis, and virtual-sensor placement without requiring a separate diagnostic representation for each task. We instantiate this framework for chain-structured, orchestration-based, and DAG-structured MASs, demonstrating that the diagnostic mapping can change with the execution architecture while the underlying analytical framework remains the same. In summary, our contributions are: (1) we introduce AgentSoV, a virtual-sensing framework that represents LLM MAS execution as a multistage dynamic system using SoV, providing a common statistical language for modeling information and variation propagation; (2) we show that the resulting dynamics support multiple system-level analyses within a single model, including failure attribution, stage-input influence, and virtual-sensor placement; (3) we adapt AgentSoV to chain-structured, orchestration-based, and DAG-structured MASs, demonstrating that its diagnostic stages and dependencies can accommodate heterogeneous execution architectures without requiring a one-to-one reproduction of the underlying trajectory; and (4) we validate AgentSoV across these architectures, demonstrating effective diagnosis from a unified learned representation without re-executing the underlying MAS.

2 AgentSoV Framework

2.1 Illustration of the State-Space Representation for MASs

Figure 1 shows an overview of our AgentSoV framework. Consider an MAS in which an Orchestrator asks a WebSurfer to retrieve evidence, passes the result to a Coder for analysis, and then produces a final response. Instead of treating every message as a separate diagnostic object, we represent this execution by three functional stages: s_1 (retrieval), s_2 (reasoning), and s_3 (final response). At stage $k, k \in \{1, 2, 3\}$, the vector $\boldsymbol{u}_k$ contains the information supplied to the stage, $\boldsymbol{x}_k$ is a latent di-

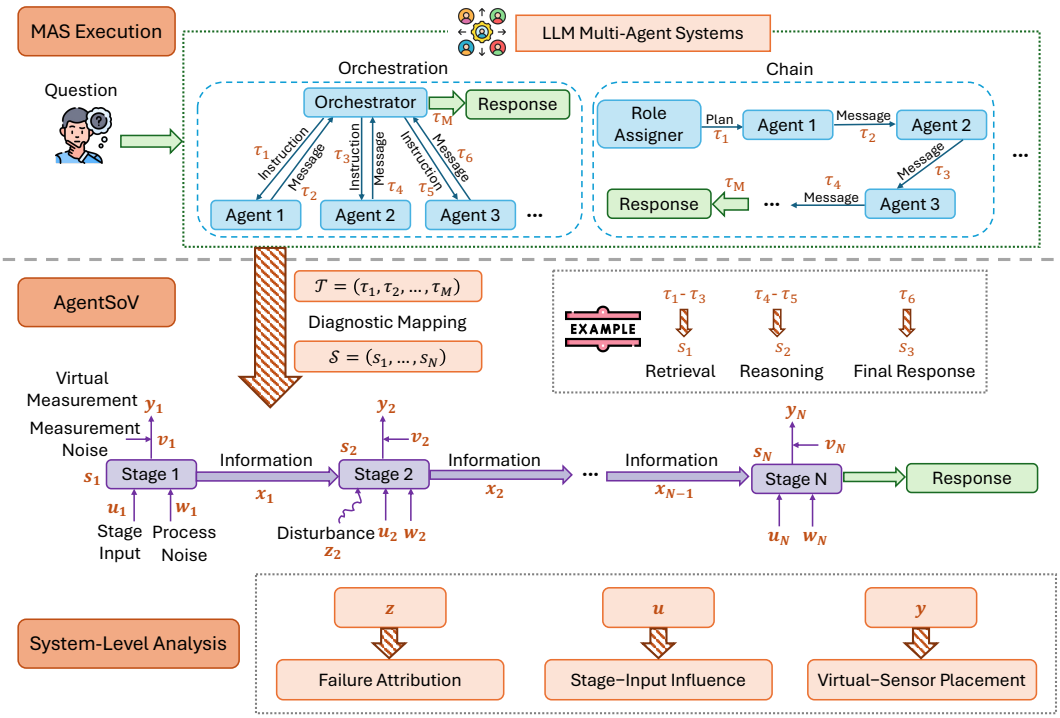


Figure 1: Schematic overview of AgentSoV. The upper panel shows the observed LLM MAS execution trajectory $\mathcal{T}$. AgentSoV maps $\mathcal{T}$ into diagnostic stages $\mathcal{S}$, constructs stage inputs $\boldsymbol{u}$ and virtual measurements $\boldsymbol{y}$, and models cross-stage variation through SoV dynamics. The diagnostic mapping need not preserve the original execution one-to-one: multiple execution steps τ may be grouped into a stage s according to their diagnostic role. The resulting representation supports three system-level analyses through different model quantities: disturbances $\boldsymbol{z}$ for failure attribution, stage inputs $\boldsymbol{u}$ for stage-input influence, and virtual measurements $\boldsymbol{y}$ for virtual-sensor placement.

agnostic state that carries task-relevant variation, $\boldsymbol{z}_k$ is an optional disturbance vector representing abnormal variation introduced at the stage, and $\boldsymbol{y}_k$ is a virtual measurement constructed from the observable stage output, such as a text representation or a tool-execution statistic. Conceptually,

$$\boldsymbol{x}_k = \boldsymbol{F}_k \boldsymbol{x}_{k-1} + \boldsymbol{B}_k \boldsymbol{u}_k + \boldsymbol{H}_k \boldsymbol{z}_k + \boldsymbol{w}_k \qquad \text{(State equation)}, \tag{1}$$

$$\boldsymbol{y}_k = \boldsymbol{C}_k \boldsymbol{x}_k + \boldsymbol{v}_k \qquad \text{(Observation equation)}. \tag{2}$$

Here, the matrices determine how the configured variables are related: $\boldsymbol{F}_k$ propagates upstream variation, $\boldsymbol{B}_k$ maps the current input, $\boldsymbol{H}_k$ maps the abnormal disturbance into the diagnostic state $\boldsymbol{x}_k$, and $\boldsymbol{C}_k$ specifies which components of the diagnostic state are visible through the virtual measurement $\boldsymbol{y}_k$. If the WebSurfer retrieves an incorrect fact, for example, that abnormal variation is represented by the disturbance $\boldsymbol{H}_k \boldsymbol{z}_k$ and may subsequently propagate through the reasoning stage s_2 into the final answer. The process noise $\boldsymbol{w}_k$ captures unmodeled stage-to-stage variation, whereas the measurement noise $\boldsymbol{v}_k$ captures uncertainty in constructing $\boldsymbol{y}_k$ from an observable output (e.g., an imperfect diagnostic feature). Thus, an agent response acts like a location in a manufacturing process at which a virtual sensor can be constructed (Kabadayi et al., 2006). Stage-indexed matrices allow heterogeneous stage operations; when stages share compatible dynamics, AgentSoV instead uses shared matrices such as $\boldsymbol{F}$, $\boldsymbol{B}$, and $\boldsymbol{C}$.

The stages form an analytical abstraction of information transformation and need not reproduce the execution log one-to-one. A stage may combine an assistant turn with its tool calls and responses, several raw operations may be aggregated, and irrelevant operations may be omitted. The dependency structure follows the MAS architecture: sequential stages depend on their predecessors, an Orchestrator's planning state may affect several workers, and a node in a DAG may depend on multiple parents. AgentSoV is therefore architecture-aware when constructing the diagnostic representation while retaining a common analytical language across architectures.

The same dynamics support three system-level analyses. *Failure attribution* identifies the disturbance source $\boldsymbol{z}_k$ whose propagated signature best explains the observed residuals. *Stage-input influence* changes $\boldsymbol{u}_k$ and propagates the resulting change in $\boldsymbol{B}_k\boldsymbol{u}_k$ through the fitted dynamics, revealing whether information supplied to an intermediate stage affects a later output (Li et al., 2025; Zhang et al., 2026d). *Virtual-sensor placement* selects a compact set of complementary intermediate measurements $\boldsymbol{y}_k$ that makes candidate disturbance sources $\boldsymbol{z}_k$ observable and distinguishable when monitoring every output is unnecessary or costly. This correspondence defines the roles of $\boldsymbol{u}_k$, $\boldsymbol{x}_k$, $\boldsymbol{y}_k$, and $\boldsymbol{z}_k$. We next describe how execution operations are mapped into diagnostic stages, after which we specify and estimate the resulting AgentSoV model.

2.2 From Execution Trajectories to Diagnostic Stages

Let an execution trajectory consist of M ordered steps $\mathcal{T} = (\tau_1, \ldots, \tau_M)$, such as agent messages, tool calls, or planning operations. AgentSoV maps these operations into N diagnostic stages,

$$\mathcal{T} \to \mathcal{S} = (s_1, \ldots, s_N), \tag{3}$$

where each s_k, $k \in \{1, \ldots, N\}$, groups execution operations that play a common diagnostic role. The mapping need not preserve the raw execution one-to-one: several operations may be aggregated into one stage, while operations irrelevant to the diagnostic objective may be omitted. Consider again the Orchestrator-WebSurfer-Coder example from Section 2.1, answering the question "Which two ASEAN countries have capitals that are furthest apart?" A simplified trajectory contains τ_1: the Orchestrator constructs a fact sheet and plan; τ_2: the Orchestrator instructs the WebSurfer to retrieve the relevant capitals; τ_3: the WebSurfer returns a candidate pair; τ_4: the Orchestrator passes the retrieved information to the Coder; τ_5: the Coder constructs the answer; and τ_6: the Orchestrator delivers the final response. Rather than treating all six operations as separate diagnostic objects, AgentSoV may represent the execution as

$$\underbrace{(\tau_1, \tau_2, \tau_3)}_{s_1\text{: retrieval}} \to \underbrace{(\tau_4, \tau_5)}_{s_2\text{: reasoning}} \to \underbrace{\tau_6}_{s_3\text{: final response}}. \tag{4}$$

The important modeling decision is therefore not simply which agent produces each raw message, but which execution operations form a meaningful unit of information transformation and variation propagation. Once these stages are chosen, the quantities introduced in Section 2.1(e.g., $\boldsymbol{u}_k$, $\boldsymbol{x}_k$, $\boldsymbol{y}_k$), and, when explicitly modeled, $\boldsymbol{z}_k$ are instantiated from the information entering, carried through, and observed at each stage. Their exact construction can vary with the execution architecture without changing their roles in the AgentSoV representation. More modeling examples are given in Appendix B for different MAS architectures.

2.3 Model Configuration

Eqs. 1-2 define the state-space representation in AgentSoV. Model configuration assigns meanings and dimensions to its variables and determines which parameters are fixed, shared, or learned. We assume isotropic Gaussian noise, $\boldsymbol{w}_k \sim \mathcal{N}(\boldsymbol{0}, \sigma_{w,k}^2\boldsymbol{I})$ and $\boldsymbol{v}_k \sim \mathcal{N}(\boldsymbol{0}, \sigma_{v,k}^2\boldsymbol{I})$, where $\sigma_{w,k}^2$ and $\sigma_{v,k}^2$ are the stage-specific process- and measurement-noise variances, respectively. AgentSoV learns the transition matrices $\boldsymbol{F}_k$, input matrices $\boldsymbol{B}_k$, $\sigma_{w,k}^2$, and $\sigma_{v,k}^2$ from training trajectories. The measurement matrix $\boldsymbol{C}_k$ is fixed when determined by the virtual-measurement design and may otherwise be estimated. The disturbance matrix $\boldsymbol{H}_k$ is specified from system knowledge or estimated from controlled perturbations or labeled failures when structured disturbances are explicitly modeled. Otherwise, we set $\boldsymbol{z}_k = \boldsymbol{0}$ and fit the normal dynamics. Although we use stage-indexed matrices for generality, common matrices such as $\boldsymbol{F}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ can be used when stages perform comparable operations and the available data support parameter sharing.

Returning to the running example makes the distinction among these terms concrete. Suppose $\boldsymbol{x}_k = \begin{bmatrix} x_k^{\text{ret}} & x_k^{\text{reas}} & x_k^{\text{ver}} \end{bmatrix}^\top$ summarizes retrieval, reasoning, and verification variation. At the retrieval stage, the task and Orchestrator instruction form the information represented by $\boldsymbol{u}_1$, while the WebSurfer response provides the observable signal from which $\boldsymbol{y}_1$ is constructed. If the WebSurfer incorrectly returns Jakarta-Bangkok, the resulting abnormal retrieval variation can be represented

through the disturbance channel $\boldsymbol{H}_1\boldsymbol{z}_1$. For illustration, let

$$\boldsymbol{z}_k = \begin{bmatrix} z_k^{\text{ret}} \\ z_k^{\text{reas}} \end{bmatrix}, \qquad \boldsymbol{H}_k = \begin{bmatrix} 1 & 0 \\ 0.3 & 1 \\ 0.2 & 0 \end{bmatrix}, \quad \boldsymbol{F}_2 = \begin{bmatrix} 0.9 & 0 & 0 \\ 0.3 & 0.5 & 0 \\ 0.2 & 0 & 0.6 \end{bmatrix}. \tag{5}$$

Under this specified disturbance basis, the two coordinates of $\boldsymbol{z}_k$ define retrieval and reasoning disturbances, and the corresponding columns of $\boldsymbol{H}_k$ specify the directions in diagnostic-state space along which those disturbances enter. At s_2, the Coder receives its current instruction and retrieved evidence through $\boldsymbol{u}_2$. Its state depends on this current-stage information through $\boldsymbol{B}_2\boldsymbol{u}_2$ and on variation inherited from the retrieval stage through $\boldsymbol{F}_2\boldsymbol{x}_1$. The matrix $\boldsymbol{F}_2$ in (5) describes strong persistence of retrieval variation while allowing it to propagate into the reasoning and verification components. A downstream stage that independently checks and corrects the retrieved information would instead have transition dynamics that attenuate this variation. Thus, $\boldsymbol{F}_k$ captures whether upstream variation is preserved, amplified, or suppressed, whereas $\boldsymbol{B}_k$ captures how current-stage information affects the diagnostic state. This separation is later used to distinguish variation propagation from stage-input influence. Finally, $\boldsymbol{C}_k$ connects the diagnostic state $\boldsymbol{x}_k$ to the retained virtual measurement $\boldsymbol{y}_k$. When the measurement directly represents the state, $\boldsymbol{C}_k = \boldsymbol{I}$; when only selected features of an agent response are retained, $\boldsymbol{C}_k$ selects or combines the corresponding state components. This distinction is especially important for the virtual-sensor analysis in Section 3.3, where different observable stage outputs provide different amounts of diagnostic information.

2.4 Model Estimation

Once the diagnostic stages $\mathcal{S}$, variable representations, dependency structure, and parameter configuration have been specified, estimating AgentSoV is a standard state-space identification problem. For latent-state instantiations, we fit the parameters designated as learnable using expectation-maximization (EM). In the E-step, Kalman filtering and Rauch-Tung-Striebel smoothing infer the latent states given the current parameter estimates. In the M-step, these parameters are updated from the inferred state transitions and measurement residuals. When the diagnostic state $\boldsymbol{x}_k$ is directly observable, for example when $\boldsymbol{x}_k = \boldsymbol{y}_k$ is a projected representation of an agent response, estimation reduces to fitting the corresponding cross-stage regression. We use normal, i.e., fault-free, training executions to characterize the normal dynamics, corresponding to $\boldsymbol{z}_k = \boldsymbol{0}$.

Thus, the contribution of AgentSoV is not a new estimator for linear state-space models. Rather, it is the construction of a diagnostic representation of MAS execution for which established state-space estimation and systems-analysis tools become applicable.

2.5 Instantiating AgentSoV Across MAS Architectures

Our formulation in AgentSoV is not tied to a particular MAS architecture. What changes across systems is how the diagnostic stages and dependencies are instantiated, not the roles of the AgentSoV variables themselves. In a chain-structured system, variation propagates primarily from the preceding stage; in an orchestration-based system, a persistent planning state may influence multiple downstream stages; and in a DAG-structured system, a stage may aggregate variation from several upstream parents. We provide concrete AgentSoV instantiations for these three settings, including BFCL (Patil et al., 2025), Magentic-One (Zhang et al., 2025), and AEGIS (Kong et al., 2026), in Appendix B. These examples show that the trajectory-to-stage mapping and transition structure can adapt to the underlying execution architecture while retaining the same analytical language of inputs $\boldsymbol{u}$, diagnostic states $\boldsymbol{x}$, virtual measurements $\boldsymbol{y}$, disturbances $\boldsymbol{z}$, and variation propagation.

3 System-Level Analysis with AgentSoV

Once the AgentSoV dynamics in Eqs. 1-2 have been fitted, the model can be used not only to predict how an MAS normally evolves, but also to reason about how variation enters, propagates, and becomes observable. We consider three complementary questions: where does abnormal variation originate, how strongly does input supplied at a stage affect downstream outcomes, and which intermediate outputs must be observed for effective diagnosis? All three analyses use the same fitted state-space dynamics and therefore do not require re-executing the underlying MAS.

3.1 FAILURE ATTRIBUTION

Failure attribution asks which stage introduced the abnormal variation. Under normal execution, we set $\boldsymbol{z}_k = \mathbf{0}$ in Eq. 1 and use the fitted model to predict the stage outputs. A disturbance $\boldsymbol{z}_k$ introduced at stage k enters the diagnostic state through $\boldsymbol{H}_k \boldsymbol{z}_k$. Recursive application of Eq. 1 shows that its effect on the virtual measurement $\boldsymbol{y}_l$ at a downstream stage $\ell > k$ is

$$\underbrace{\boldsymbol{C}_\ell \boldsymbol{F}_\ell \boldsymbol{F}_{\ell-1} \cdots \boldsymbol{F}_{k+1} \boldsymbol{H}_k}_{\boldsymbol{G}_{\ell,k}} \boldsymbol{z}_k. \tag{6}$$

$\boldsymbol{G}_{\ell,k} = \boldsymbol{C}_\ell \boldsymbol{F}_\ell \boldsymbol{F}_{\ell-1} \cdots \boldsymbol{F}_{k+1} \boldsymbol{H}_k$ is the propagated disturbance signature from stage k to stage ℓ, a term established in state-space model-based diagnosis (Gertler, 1998; Isermann, 2006). It combines three elements of the AgentSoV model: $\boldsymbol{H}_k$ specifies where abnormal variation enters, the transition matrices $\boldsymbol{F}_{k+1}, \ldots, \boldsymbol{F}_\ell$ describe how it propagates, and $\boldsymbol{C}_\ell$ specifies how that propagated variation becomes observable. Accordingly, $\boldsymbol{G}_{\ell,k} \boldsymbol{z}_k$ is the observable effect of disturbance $\boldsymbol{z}_k$ at stage ℓ.

In the Orchestrator-WebSurfer-Coder example, an incorrect retrieval introduced at s_1 should influence both the Coder response at s_2 and, unless corrected, the final response at s_3. A reasoning error introduced only at s_2 cannot explain abnormal behavior already visible at s_1. AgentSoV therefore attributes failure by comparing the observed deviations from normal predictions with the downstream patterns implied by candidate $\boldsymbol{H}_k \boldsymbol{z}_k$ terms. When several stages may contribute, the candidate disturbances can be estimated jointly with a sparsity penalty for accurate identification (see Eq. 14 in Appendix C.3). This differs from local anomaly scoring since an attributed source must explain how its variation propagates through the execution, not merely exhibit a large local error. The discussion describes the attribution principle at the system level. For completeness, Appendix C gives the corresponding computational formulation.

3.2 STAGE-INPUT INFLUENCE

Failure attribution concerns the disturbance channel $\boldsymbol{H}_k \boldsymbol{z}_k$ in Eq. 1. The process-input channel $\boldsymbol{B}_k \boldsymbol{u}_k$ supports a different question: how strongly does the information entering at stage k affect a later measurement? This question applies to successful as well as failed trajectories. For example, the WebSurfer may return correct information that the Coder largely ignores, or information that is redundant with evidence available elsewhere (Li et al., 2025; Zhang et al., 2026d). To quantify this effect, we set the observed input $\boldsymbol{u}_k$ to zero in the fitted model while holding the other stage inputs fixed. Let $\hat{\boldsymbol{y}}_\ell^{-k}$ denote the predicted measurement at a later stage ℓ under this intervention. The linear state and observation equations, Eqs. 1-2, give

$$\hat{\boldsymbol{y}}_\ell - \hat{\boldsymbol{y}}_\ell^{-k} = \boldsymbol{C}_\ell \boldsymbol{F}_\ell \boldsymbol{F}_{\ell-1} \cdots \boldsymbol{F}_{k+1} \boldsymbol{B}_k \boldsymbol{u}_k, \qquad k < \ell. \tag{7}$$

$\|\hat{\boldsymbol{y}}_\ell - \hat{\boldsymbol{y}}_\ell^{-k}\|_2$ is the influence magnitude of the predicted change at stage ℓ caused by removing the modeled input at stage k. Taking $\ell = N$ measures its effect on the final measurement. A larger influence $\|\hat{\boldsymbol{y}}_\ell - \hat{\boldsymbol{y}}_\ell^{-k}\|_2$ means that the input is more consequential. No MAS replay is required.

3.3 VIRTUAL-SENSOR PLACEMENT

The preceding analyses assume that intermediate outputs $\boldsymbol{y}_k$ are available for diagnosis. In practice, monitoring every stage may be unnecessary: different outputs can reveal nearly the same propagated variation, while omitting another output may make important failure sources difficult to distinguish. AgentSoV therefore treats the choice of which $\boldsymbol{y}_k$ to retain as a virtual-sensor placement problem. The propagated disturbance signature $\boldsymbol{G}_{\ell,k}$ gives the key quantity. Observing $\boldsymbol{y}_\ell$ is useful for diagnosing a disturbance at stage k when $\boldsymbol{G}_{\ell,k} \boldsymbol{z}_k$ is large relative to normal variation. More importantly, two candidate measurements are redundant when they expose essentially the same disturbance directions. AgentSoV therefore selects a limited set of observation stages $\mathcal{S}$ using the standard log-determinant information criterion, $\mathcal{S}^\star = \arg\max_{|\mathcal{S}| \le b} \log\det\left(\sum_{\ell \in \mathcal{S}} \mathcal{I}_\ell + \eta \boldsymbol{I}\right)$, where $\mathcal{I}_\ell$ is the information about the candidate propagated disturbances contained in $\boldsymbol{y}_\ell$ relative to normal residual variation, b is the monitoring budget, and $\eta > 0$ stabilizes the calculation. $\mathcal{I}_\ell$ is computed from the propagated effects in $\boldsymbol{G}_{\ell,k}$ and the residual covariance estimated from fault-free trajectories.

Returning to the toy example, observing only the final response may not distinguish an incorrect WebSurfer retrieval from an error introduced by the Coder, because both may produce the same

Table 1: Failure attribution performance on BFCL and Magentic-One.

Dataset	Method	Step Accuracy	Top-3 Step Accuracy	MRR	Agent Accuracy
BFCL	Uniform Random	11.7%	35.2%	31.7%	11.7%
	GPT-5.5 (All-at-Once)	$16.5\% \pm 1.2\%$	$58.3\% \pm 1.2\%$	$39.4\% \pm 1.4\%$	$16.5\% \pm 1.2\%$
	GPT-5.5 (Step-by-Step)	$9.7\% \pm 0.9\%$	$49.8\% \pm 2.8\%$	$24.9\% \pm 1.1\%$	$9.7\% \pm 0.9\%$
	GPT-5.5 (Binary Search)	$2.9\% \pm 1.1\%$	$48.2\% \pm 3.0\%$	$23.7\% \pm 1.4\%$	$2.9\% \pm 1.1\%$
	MASC	$5.2\% \pm 1.0\%$	$33.8\% \pm 1.4\%$	$29.5\% \pm 0.4\%$	$5.2\% \pm 1.0\%$
	OAT	$8.4\% \pm 0.5\%$	$29.5\% \pm 1.3\%$	$27.9\% \pm 0.3\%$	$8.4\% \pm 0.5\%$
	AgentSoV-S (F=0)	$35.1\% \pm 1.1\%$	$58.1\% \pm 1.7\%$	$52.6\% \pm 1.0\%$	$35.1\% \pm 1.1\%$
	AgentSoV-S	$36.9\% \pm 0.6\%$	$\mathbf{62.7\% \pm 2.2\%}$	$53.6\% \pm 0.7\%$	$36.9\% \pm 0.6\%$
	AgentSoV (F=0)	$28.5\% \pm 1.8\%$	$46.0\% \pm 1.0\%$	$43.4\% \pm 1.6\%$	$28.5\% \pm 1.8\%$
	AgentSoV	$\mathbf{37.5\% \pm 2.0\%}$	$62.5\% \pm 1.8\%$	$\mathbf{54.4\% \pm 0.8\%}$	$\mathbf{37.5\% \pm 2.0\%}$
Magentic-One	Uniform Random	13.6%	40.8%	34.4%	46.1%
	GPT-5.5 (All-at-Once)	$31.5\% \pm 1.9\%$	$51.6\% \pm 1.5\%$	$42.8\% \pm 0.8\%$	$46.3\% \pm 2.2\%$
	GPT-5.5 (Step-by-Step)	$3.7\% \pm 0.7\%$	$46.9\% \pm 1.7\%$	$28.3\% \pm 0.8\%$	$51.9\% \pm 3.1\%$
	GPT-5.5 (Binary Search)	$5.6\% \pm 1.4\%$	$43.2\% \pm 1.7\%$	$20.8\% \pm 0.5\%$	$46.3\% \pm 2.2\%$
	MASC	$3.7\% \pm 0.1\%$	$28.1\% \pm 2.2\%$	$26.0\% \pm 2.0\%$	$46.3\% \pm 2.3\%$
	OAT	$27.2\% \pm 0.9\%$	$54.1\% \pm 0.7\%$	$45.0\% \pm 0.5\%$	$53.1\% \pm 1.7\%$
	AgentSoV-S (F=0)	$25.2\% \pm 2.8\%$	$54.5\% \pm 1.4\%$	$47.5\% \pm 1.8\%$	$45.9\% \pm 5.5\%$
	AgentSoV-S	$36.3\% \pm 3.6\%$	$\mathbf{65.6\% \pm 1.9\%}$	$\mathbf{53.0\% \pm 2.4\%}$	$\mathbf{63.0\% \pm 3.9\%}$
	AgentSoV (F=0)	$37.0\% \pm 3.3\%$	$53.0\% \pm 1.5\%$	$45.9\% \pm 1.4\%$	$60.4\% \pm 5.3\%$
	AgentSoV	$\mathbf{39.3\% \pm 3.8\%}$	$54.8\% \pm 1.5\%$	$47.3\% \pm 1.9\%$	$61.9\% \pm 4.2\%$

final answer. Adding the WebSurfer output can expose the retrieval error before reasoning occurs, whereas adding another downstream output $\boldsymbol{y}$ carrying nearly identical information may provide little additional diagnostic value. Virtual-sensor placement therefore asks not simply where can we observe the MAS, but which observations make its sources of variation most distinguishable?

4 Experiments

4.1 Experimental Setup

We evaluate AgentSoV on BFCL (Liu et al., 2026; Patil et al., 2025), Magentic-One (Liu et al., 2026; Zhang et al., 2025), and AEGIS (Kong et al., 2026). These datasets represent chain-structured tool use, centralized orchestration, and DAG-structured information flow, respectively. For failure attribution, we follow Zhang et al. (2025) and report step- and agent-level accuracy, together with Top-3 step accuracy and mean reciprocal rank (MRR). The step- and agent-level accuracy are identical on BFCL because its steps do not distinguish agent types. Because an AEGIS trajectory may contain multiple faulty agents, we report exact-set accuracy and top-1 hit accuracy. Results are reported as mean $\pm$ standard deviation over five random seeds. For each system-level analysis, AgentSoV is compared only with applicable baselines that provide sufficient implementation details and support the corresponding task. We compare AgentSoV with six baselines spanning four categories: (1) Uniform Random, (2) three GPT-5.5 prompting strategies from Zhang et al. (2025), (3) MASC (Shen et al., 2026), and (4) OAT (Yeh et al., 2026). We evaluate two variants of our method: AgentSoV-S uses a supervised residual-localization head and step-level fault labels described in Appendix B.3, whereas AgentSoV follows Section 2. Additional details are described in Appendix D.

4.2 Failure Attribution

Tables 1 and 2 show that AgentSoV provides effective failure attribution across different MAS architectures. On BFCL, AgentSoV achieves the highest step accuracy (37.5%). On Magentic-One, AgentSoV performs best for exact step localization (39.3%), whereas AgentSoV-S achieves higher agent accuracy, top-3 accuracy, and MRR. This suggests that supervised residual-to-fault mappings are useful for identifying the responsible agent when multiple interactions from the same agent exhibit related failure patterns, while AgentSoV's propagation model provides finer discrimination of the exact failure stage. On AEGIS, AgentSoV achieves 43.6% exact-set and 90.4% top-1 accuracy, outperforming AgentSoV-S, which indicates that explicit modeling of cross-agent propagation is beneficial when multiple faults interact through graph dependencies. Finally, the $\boldsymbol{F}$=0 ablations generally reduce performance, with large drops for AgentSoV on BFCL and AgentSoV-S on Magentic-One, confirming that cross-stage propagation does provide crucial diagnostic information.

Table 2: Failure attribution performance on AEGIS.

Method	Exact-Set Accuracy	Top-1 Accuracy	Top-3 Accuracy	MRR
Uniform Random	9.0%	44.9%	91.7%	67.6%
GPT-5.5 (All-at-Once)	$7.7\% \pm 2.1\%$	$59.0\% \pm 4.0\%$	$99.1\% \pm 0.3\%$	$91.5\% \pm 0.3\%$
GPT-5.5 (Step-by-Step)	$8.3\% \pm 2.2\%$	$60.9\% \pm 2.6\%$	$99.8\% \pm 0.3\%$	$83.0\% \pm 0.5\%$
GPT-5.5 (Binary Search)	$12.2\% \pm 2.6\%$	$71.9\% \pm 2.7\%$	$99.6\% \pm 0.3\%$	$92.9\% \pm 0.4\%$
MASC	$0.1\% \pm 0.3\%$	$76.7\% \pm 2.6\%$	$95.5\% \pm 0.0\%$	$86.53\% \pm 1.4\%$
OAT	$4.9\% \pm 1.1\%$	$41.7\% \pm 1.0\%$	$87.5\% \pm 1.2\%$	$62.5\% \pm 0.6\%$
AgentSoV-S (F=0)	$32.1\% \pm 0.7\%$	$76.9\% \pm 0.0\%$	$98.1\% \pm 0.2\%$	$87.1\% \pm 0.4\%$
AgentSoV-S	$34.0\% \pm 1.1\%$	$78.9\% \pm 0.2\%$	$99.4\% \pm 0.0\%$	$92.4\% \pm 0.0\%$
AgentSoV (F=0)	$41.6\% \pm 0.7\%$	$87.2\% \pm 0.3\%$	$\mathbf{100.0\% \pm 0.0\%}$	$95.1\% \pm 0.1\%$
AgentSoV	$\mathbf{43.6\% \pm 0.9\%}$	$\mathbf{90.4\% \pm 0.7\%}$	$\mathbf{100.0\% \pm 0.0\%}$	$\mathbf{95.6\% \pm 0.3\%}$

We measure abnormal variation using the state-prediction error, $\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k\|_2^2$, where $\hat{\boldsymbol{x}}_k$ is defined in Eq. 9. For each verified AEGIS repair, in which a non-faulty downstream receiver corrects a faulty parent's output, we compute the relative anomaly reduction from the parent to the receiver. A repair is detected when this reduction exceeds the specified threshold. Table 3 shows that AgentSoV achieves the highest recall at every threshold, detecting 74.7% of repairs with any reduction. The results indicate that AgentSoV consistently captures substantial anomaly attenuation during downstream repair.

Table 3: Recall of repaired AEGIS executions under different anomaly-reduction thresholds.

Method	$> 0\%$	$\geq 10\%$	$\geq 20\%$
Uniform Random	65.5%	45.1%	24.8%
MASC	67.1%	67.1%	60.8%
OAT	48.1%	21.5%	8.9%
AgentSoV	$\mathbf{74.7\%}$	$\mathbf{68.4\%}$	$\mathbf{63.3\%}$

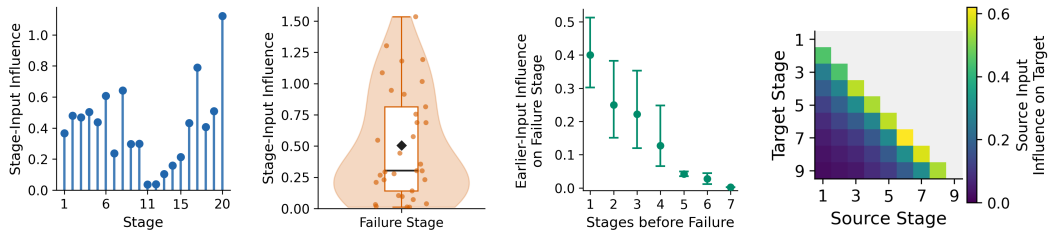
Table 4 shows that AgentSoV has low one-time training costs and inference overhead. AgentSoV-S is the fastest method at inference (0.0001s), while AgentSoV requires 0.0242s per trajectory, about $13.6\times$ faster than OAT and more than $110\times$ faster than the fastest GPT-5.5 strategy. MASC is faster than AgentSoV at inference but requires more training time and LLM encoding. AgentSoV avoids LLM inference and token consumption, yielding low costs without API latency and token usage. Additional results are in Appendix E.

Table 4: Computational costs of BFCL.

Method	Training (s)	Inference (s)	LLM Tokens (In/Out)
GPT-5.5 (All-at-Once)	0	5.3427	618/440
GPT-5.5 (Step-by-Step)	0	16.4219	2,586/1,026
GPT-5.5 (Binary Search)	0	2.6835	1,221/27
MASC	23.86	0.0054	5,102/0
OAT	7.82	0.3282	4,937/0
AgentSoV-S	10.92	0.0001	–
AgentSoV	10.89	0.0242	–

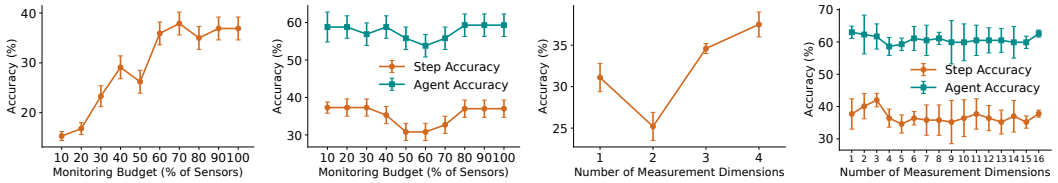
4.3 Stage-Input Influence

Figure 2 evaluates the stage-input influence as described in Eq. 7. Figure 2a shows substantial heterogeneity in how strongly individual stage inputs $\boldsymbol{u}$ affect the final outcome $\boldsymbol{y}_N$. Influence ranges from below 0.1 for several middle stages to around 1.1 at Stage 20, with other relatively influential stages such as Stages 6, 8, and 17 reaching roughly 0.6, 0.65, and 0.8, respectively. Thus, the learned dynamics do not assign comparable influence to all orchestrator instructions: some inputs produce little downstream change when removed, whereas others substantially alter the predicted final representation. Figure 2b shows a similarly heterogeneous distribution of failure-stage input influence on $\boldsymbol{y}_N$. The mean influence is approximately 0.5, while many cases lie below 0.3 and a smaller number exceed 1.0, reaching about 1.5. This indicates that failure stages are not necessarily the stages whose inputs exert the strongest downstream influence. Figure 2c further reveals a strong temporal decay in this influence. The input immediately preceding a failure stage has mean influence of approximately 0.40, which decreases to 0.25, 0.22, and 0.13 at two, three, and four stages earlier, respectively. In other words, the immediately preceding input is roughly three times as influential as an input four stages earlier. Figure 2d shows the largest influence concentrate near the diagonal, where source and target stages are close, while entries become progressively weaker as their separation increases. Together, these results show that AgentSoV can identify non-uniform and predominantly local information propagation, revealing how stage inputs differ in downstream influence and how their effects attenuate across successive agent interactions.



(a) Input influence by stage. (b) Failure-stage input influence. (c) Influence of earlier inputs on failure stage. (d) Source-to-target stage influence.

Figure 2: Stage-input influence on Magentic-One.



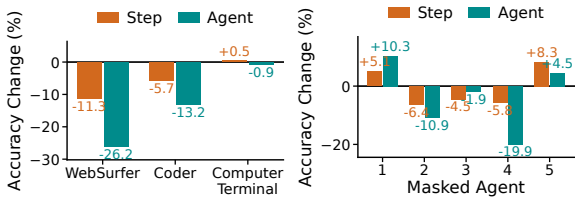
(a) Sensor: BFCL. (b) Sensor: MO. (c) $\boldsymbol{y}$ dimensions: BFCL. (d) $\boldsymbol{y}$ dimensions: MO.

Figure 3: Virtual-sensor placement and $\boldsymbol{y}$-dimension selection under different observation budgets: (a & c) BFCL and (b & d) Magentic-One (MO).

4.4 Virtual-Sensor Placement

Figure 3 evaluates virtual-sensor placement and $\boldsymbol{y}$-dimension selection on BFCL and Magentic-One. Figures 3a and 3b vary the percentage of monitored stages following Section 3.3, while Figures 3c and 3d similarly vary the monitored dimensions in each $\boldsymbol{y}_k$. BFCL benefits from broader stage coverage and all four dimensions, whereas Magentic-One matches full-observation performance with only 10% of stages and achieves its highest step accuracy with three dimensions, indicating substantial measurement redundancy. These results show that AgentSoV can identify compact monitoring configurations and reveal when additional measurements are necessary.

Figure 4 evaluates agent-specific effects by masking all call-node $\boldsymbol{y}$ from one agent type. Masking WebSurfer on Magentic-One and Agent 4 on AEGIS causes the largest accuracy losses, showing that their outputs provide important diagnostic evidence. In contrast, masking AEGIS Agents 1 and 5 improves attribution, showing that their $\boldsymbol{y}$ obscure fault signatures. Diagnostic value depends on an agent's role and position, and additional monitoring does not always improve attribution.



(a) Magentic-One. (b) AEGIS.

Figure 4: Agent-masking effects on failure-attribution accuracy relative to full observation.

5 Conclusion

We introduced AgentSoV, a virtual-sensing framework that represents heterogeneous LLM MAS trajectories as diagnostic stages in a stochastic dynamic system. Its architecture-aware yet unified model captures cross-stage variation in chain-structured, orchestration-based, and DAG-structured systems, supporting failure attribution, stage-input influence, and virtual-sensor placement without MAS replay. Experiments on BFCL, Magentic-One, and AEGIS show improved failure localization, differentiated downstream influence across stage inputs, and preserved diagnostic performance with fewer measurements. Future work can extend AgentSoV beyond linear state-space dynamics and automate stage and measurement construction.

AI USE STATEMENT

In this work, we used generative AI tools for running LLM-required baselines in experiments and writing assistance, including language editing and polishing of author-written text. We have not used generative AI tools for other tasks with required disclosure, and the rest of the required disclosure tasks are not applicable to this work. We have reviewed all AI-assisted work and manually verified the resulting text for technical accuracy, consistency with our methods and experiments, and appropriate attribution. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work develops a framework for diagnosing failures in LLM-based multi-agent systems. All datasets used in this study are publicly available, and no private, personally identifiable, or otherwise sensitive data were used. We followed responsible research practices throughout the study.

REPRODUCIBILITY STATEMENT

Detailed instructions for dataset access, pre-processing steps and methods implementation are provided in Section 4 and Appendix. Our code will be released publicly upon acceptance.

REFERENCES

Kaveh Bastani, Zhenyu Kong, Wenzhen Huang, and Yingqing Zhou. Compressive sensing–based optimal sensor placement and fault diagnosis for multi-station assembly processes. *IIE Transactions*, 48(5):462–474, 2016.

Dariusz Ceglarek, W Huang, S Zhou, Yu Ding, R Kumar, and Y Zhou. Time-based competition in multistage manufacturing: Stream-of-variation analysis (sova) methodology. *International Journal of Flexible Manufacturing Systems*, 16(1):11–44, 2004.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *Advances in Neural Information Processing Systems*, 38, 2026.

Yurui Chang, Bochuan Cao, Yujia Wang, Jinghui Chen, and Lu Lin. Jopa: Explaining large language model's generation via joint prompt attribution. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22106–22122, 2025.

Yu Ding, Jianjun Shi, and Dariusz Ceglarek. Diagnosability analysis of multi-station manufacturing processes. *J. Dyn. Sys., Meas., Control*, 124(1):1–13, 2002.

Liming Dong, Qinghua Lu, and Liming Zhu. Agentops: Enabling observability of llm agents, 2024. URL https://arxiv.org/abs/2411.05285.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.

Yu Ge, Linna Xie, Zhong Li, Yu Pei, and Tian Zhang. Spectrum-based failure attribution for multi-agent systems. *Proceedings of the ACM on Software Engineering*, 3(FSE):1888–1909, 2026.

Janos Gertler. *Fault Detection and Diagnosis in Engineering Systems*. Marcel Dekker, New York, 1998. ISBN 9780824794279.

Carlos Guestrin, Andreas Krause, and Ajit Paul Singh. Near-optimal sensor placements in gaussian processes. In *Proceedings of the 22nd international conference on Machine learning - ICML '05*, 2005. doi: 10.1145/1102351.1102385. URL https://doi.org/10.1145/1102351.1102385.

Yun Hua, Haosheng Chen, Shiqin Wang, Wenhao Li, Xiangfeng Wang, and Jun Luo. Shapley-coop: Credit assignment for emergent cooperation in self-interested llm agents, 2025. URL https://arxiv.org/abs/2506.07388.

Xun Huan, Jayanth Jagalur, and Youssef Marzouk. Optimal experimental design: Formulations and computations. *Acta Numerica*, 2024. doi: 10.1017/s0962492924000023. URL https://doi.org/10.1017/s0962492924000023.

Qiang Huang, Nairong Zhou, and Jianjun Shi. Stream of variation modeling and diagnosis of multi-station machining processes. In *ASME International Mechanical Engineering Congress and Exposition*, volume 19166, pp. 81–88. American Society of Mechanical Engineers, 2000.

Rolf Isermann. *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. Springer, Berlin, Heidelberg, 2006. ISBN 9783540303688. doi: 10.1007/3-540-30368-5.

Sanem Kabadayi, Adam Pridgen, and Christine Julien. Virtual sensors: Abstracting data from physical sensors. In *2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'06)*, pp. 6–pp. IEEE, 2006.

Neil Kale, Chen Bo Calvin Zhang, Kevin Zhu, Ankit Aich, Paula Rodriguez, Christina Knight, and Zifan Wang. Reliable weak-to-strong monitoring of llm agents. In *International Conference on Learning Representations*, volume 2026, pp. 78083–78138, 2026.

Fanqi Kong, Ruijie Zhang, Huaxiao Yin, Guibin Zhang, Xiaofei Zhang, Ziang Chen, Zhaowei Zhang, Xiaoyuan Zhang, Song-Chun Zhu, and Xue Feng. Aegis: Automated error generation and attribution for multi-agent systems. In *International Conference on Learning Representations*, volume 2026, pp. 52994–53028, 2026.

Ao Li, Yuexiang Xie, Songze Li, Fugee Tsung, Bolin Ding, and Yaliang Li. Agent-oriented planning in multi-agent systems. In *International Conference on Learning Representations*, volume 2025, pp. 19495–19517, 2025.

Junzhi Li, Peng He, Qirui Ji, Wei Wang, Lixiang Liu, and Chuxiong Sun. Discovering efficient and explainable communication topologies for llm-based multi-agent systems via causal inference, 2026a. URL https://arxiv.org/abs/2608.12921.

Shiyuan Li, Yixin Liu, Qingsong Wen, Chengqi Zhang, and Shirui Pan. Assemble your crew: Automatic multi-agent communication topology design via autoregressive graph generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026b. doi: 10.1609/aaai.v40i28.39481. URL https://doi.org/10.1609/aaai.v40i28.39481.

Ting-Wei Li, Yuanchen Bei, Xiao Lin, and Hanghang Tong. Beyond llm-based reasoning: Lightweight gnns for agent failure attribution. *arXiv preprint arXiv:2608.18575*, 2026c.

Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9, 2024a.

Yunxuan Li, Yibing Du, Jiageng Zhang, Longxiang Hou, Peter Grabowski, Yeqing Li, and Eugene Ie. Improving multi-agent debate with sparse communication topology. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024b. doi: 10.18653/v1/2024.findings-emnlp.427. URL https://doi.org/10.18653/v1/2024.findings-emnlp.427.

Zhibo Liang, Tianze Hu, Zaiye Chen, and Mingjie Tang. Sieve: Selective integrity verification and escalation for defending llm agents against indirect prompt injection, 2025. URL https://arxiv.org/abs/2512.06716.

Chin Qinyan Liu, Yu Ding, and Yong Chen. Optimal coordinate sensor placements for estimating mean and variance components of variation sources. *IIE transactions*, 37(9):877–889, 2005.

Xuannan Liu, Xiao Yang, Zekun Li, Pei Pei Li, and Ran He. Agenthallu: Benchmarking automated hallucination attribution of LLM-based agents. In *Agentic AI in the Wild: From Hallucinations to Reliable Autonomy*, 2026. URL https://openreview.net/forum?id=qtQkhE9zqF.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic LLM-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=XII0Wp1XA9.

Weidi Luo, Shenghong Dai, Xiaogeng Liu, Suman Banerjee, Huan Sun, Muhao Chen, and Chaowei Xiao. Agrail: A lifelong agent guardrail with effective and adaptive safety detection, 2025. URL https://arxiv.org/abs/2502.11448.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Dover: Intervention-driven auto debugging for llm multi-agent systems. In *International Conference on Learning Representations*, volume 2026, pp. 28059–28087, 2026.

Yutao Mou, Zhangchi Xue, Lijun Li, Peiyang Liu, Shikun Zhang, Wei Ye, and Jing Shao. Toolsafe: Enhancing tool invocation safety of llm-based agents via proactive step-level guardrail and feedback, 2026. URL https://arxiv.org/abs/2601.10156.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Xu Shen, Yixin Liu, Yiwei Dai, Yili Wang, Rui Miao, Yue Tan, Shirui Pan, and Xin Wang. Understanding the information propagation effects of communication topologies in llm-based multi-agent systems. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025. doi: 10.18653/v1/2025.emnlp-main.623. URL https://doi.org/10.18653/v1/2025.emnlp-main.623.

Xu Shen, Qi Zhang, Song Wang, Zhen Tan, Xinyu Zhao, Laura Yao, Vaishnav Tadiparthi, Hossein Nourkhiz Mahjoub, Ehsan Moradi Pari, Kwonjoon Lee, et al. Metacognitive self-correction for multi-agent system via prototype-guided next-execution reconstruction. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 23320–23337, 2026.

Jianjun Shi. *Stream of variation modeling and analysis for multistage manufacturing processes*. CRC press, 2006.

Kai Sun, Wenqiang Li, Bo Dong, Yuxin Lin, Jingyao Zhang, and Bin Shi. Scope delineation before localization: A two-stage framework for enhancing failure attribution in multi-agent systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33108–33116, 2026.

Haoyu Wang, Christopher M. Poskitt, Jiali Wei, and Jun Sun. Probguard: Proactive runtime monitoring for llm agent safety via probabilistic prediction. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2025a. doi: 10.48550/arxiv.2508.00500. URL https://arxiv.org/abs/2508.00500.

Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. Agentdropout: Dynamic agent elimination for token-efficient and high-performance llm-based multi-agent collaboration. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025b. doi: 10.18653/v1/2025.acl-long.1170. URL https://doi.org/10.18653/v1/2025.acl-long.1170.

Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. Agentnet: Decentralized evolutionary coordination for llm-based multi-agent systems. *Advances in Neural Information Processing Systems*, 38:107309–107336, 2026a.

Zihao Yang, Mosh Levy, Yoav Goldberg, and Byron C. Wallace. Compared to what? baselines and metrics for counterfactual prompting. In *Conference on Language Modeling (COLM)*, 2026b. doi: 10.48550/arxiv.2605.01048. URL https://arxiv.org/abs/2605.01048.

Samuel Yeh, Yiwen Zhu, Shaleen Deep, and Sharon Li. Tracing agentic failure from the flow of success. *arXiv preprint arXiv:2607.12747*, 2026. URL https://arxiv.org/abs/2607.12747.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for llm-based multi-agent systems, 2024. URL https://arxiv.org/abs/2410.02506.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. Agentracer: Who is inducing failure in the llm agentic systems? In *International Conference on Learning Representations*, volume 2026, pp. 11377–11399, 2026a.

Jiamu Zhang, Lingxi Zhang, Pengjun Lu, Qiyue Zhang, Yu-Neng Chuang, Zhengchen Li, Shuai Xu, Vipin Chaudhary, and Hanjie Chen. Rethinking the evaluation of efficiency methods for multi-agent systems. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2026b. doi: 10.48550/arxiv.2609.05933. URL https://arxiv.org/abs/2609.05933.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? On automated failure attribution of LLM multi-agent systems. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76583–76599. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/zhang25cq.html.

Zhen Zhang, Wanjing Zhou, Juncheng Li, Hao Fei, Jun Wen, and Wei Ji. Radar: Redundancy-aware diffusion for multi-agent communication structure generation. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026c. doi: 10.48550/arxiv.2605.09907. URL https://arxiv.org/abs/2605.09907.

Zhiwei Zhang, Xiaomin Li, Yudi Lin, Hui Liu, Ramraj Chandradevan, Linlin Wu, Minhua Lin, Fali Wang, Xianfeng Tang, Qi He, et al. Unlocking the power of multi-agent llm for reasoning: From lazy agents to deliberation. In *International Conference on Learning Representations*, volume 2026, pp. 60547–60579, 2026d.

Shiyu Zhou, Yu Ding, Yong Chen, and Jianjun Shi. Diagnosability study of multistage manufacturing processes based on linear mixed-effects models. *Technometrics*, 45(4):312–325, 2003.

A Related Work

A.1 SoV

SoV was a branch of state-space models developed for multistage manufacturing systems to model how variation introduced at one process stage propagates to and is observed at downstream stages (Shi, 2006; Ceglarek et al., 2004). Early work formulated this propagation using knowledge-based state-space models, enabling process-level diagnosis of variation sources across multiple stations (Huang et al., 2000). Subsequent studies connected the propagation model to system diagnosability, analogous to observability in control systems, and examined whether different variation sources can be distinguished from available measurements (Ding et al., 2002; Zhou et al., 2003). The same structure also provides a principled basis for sensor placement: measurements can be allocated across stages according to how variation is transmitted and detected, rather than treating each station independently (Liu et al., 2005; Bastani et al., 2016). AgentSoV extends this SoV perspective to address the challenges in LLM MASs. LLM MASs often fail because local errors (such as faulty specifications, misaligned messages, or invalid intermediate results) propagate downstream, causing symptoms far from their source and evading sparse or output-only checks. AgentSoV explicitly models this propagation, by constructing virtual measurements from observable execution signals and modeling how task-relevant variation propagates across diagnostic stages, enabling fault attribution, identification of redundant stage contributions, and cost-aware placement of monitors where they provide the greatest diagnostic value.

A.2 Failure Attribution in LLM MASs

Failure attribution for MAS has been approached through four main methodologies, including LLM-based trajectory judging and decomposition, supervised attribution models, causal or counterfactual intervention, and structure-based methods such as spectrum analysis and anomaly detection.

The widely adopted Who&When benchmark (Zhang et al., 2025) formalizes agent and step level attribution for single agent and decisive step. AEGIS scales supervision through synthetically injected and labeled errors (Kong et al., 2026). These formulations may not capture naturally occurring failures with multiple agents and multiple interacting causes, and their dependence on failure annotations limits transfer to new systems. AgentSoV fits normal cross-stage dynamics from fault-free trajectories and uses group-sparse disturbance estimation, identifying multiple abnormal stages without requiring labeled failures.

On the representative methods, SDBL reduces the reasoning burden by separating failure-scope detection from step localization (Sun et al., 2026), and AgenTracer trains a specialized model using counterfactual-replay supervision (Zhang et al., 2026a). However, these methods remain tied to trajectory representations and training distributions whose semantics change across sequential, orchestrated, and DAG-based architectures. AgentSoV relocates this architecture-specific adaptation into explicit definitions of diagnostic stages, inputs, states, and measurements, after which attribution is performed through a common state-space formulation.

The intervention based methods, on the other hand, provide stronger causal evidence. DoVer tests candidate explanations by modifying executions (Ma et al., 2026), while FAMAS estimates suspiciousness from repeated trajectory replay (Ge et al., 2026). Their principal limitation is operational cost and the need for a re-executable system and environment. AgentSoV compares an observed trajectory with its normal model prediction and attribute the resulting residual without replay or per-trace LLM inferences in MAS.

Finally, OAT learns the dynamics of successful trajectories and detects deviations without failure labels, but primarily returns anomaly scores rather than an explicit account of where and how abnormal variation entered (Yeh et al., 2026). AgentSoV extends this dynamical perspective by decomposing residuals into stage-specific disturbance directions and propagating their signatures across downstream observations, producing more interpretable attribution while exposing when candidate failure sources are observationally difficult to distinguish.

A.3 Information Analysis in LLM MASs

The information analysis is primarily studied through communication pruning, topology optimization, heuristic influence scoring, and cooperative credit assignment. AgentPrune (Zhang et al., 2024) and AgentDropout (Wang et al., 2025b) learn masks that remove redundant communication edges or agents while preserving task performance. Sparse-debate and topology-generation methods optimize communication structures using task-level reward (Li et al., 2024b; Zhang et al., 2026c; Li et al., 2026b). These methods demonstrate that substantial communication can be omitted, but their learned structures provide limited insight into why a particular message was redundant or not. Also, reported gains can depend strongly on the initial topology, prompts, and token-cost accounting (Li et al., 2026a; Zhang et al., 2026b). AgentSoV addresses this interpretability gap by representing each message or stage input, replacing it with a fault-free reference, and propagating the resulting counterfactual change through an identified dynamical model to measure its effect on final output quality.

Moreover, DyLAN directly estimates individual-agent importance from downstream effects (Liu et al., 2024), while causal analyses examine how correct and erroneous information spreads through alternative communication topologies (Shen et al., 2025). However, DyLAN relies on an unsupervised propagation heuristic, and topology-level analyses do not isolate the contribution of a specific stage input. AgentSoV instead produces a stage-level influence score, that distinguishes information while accounting for its modeled downstream propagation.

As a principled alternative, Shapley-based methods distribute credit across agents or messages, but evaluate many possible coalitions grows exponentially with system size (Hua et al., 2025). Under its linear approximation, AgentSoV computes each counterfactual contribution in closed form from the fitted model, avoiding coalition enumeration and repeated MAS execution.

Evaluation remains an open challenge: existing studies typically combine general capability benchmarks such as MMLU, GSM8K, MATH, and HumanEval with token or API-cost measurements, rather than using a dedicated benchmark for information contribution (Yang et al., 2026b; Chang et al., 2025). AgentSoV offers a common model-based quantity that could support such evaluation across architectures, although its fault-free reference inputs and one-at-a-time perturbations must be carefully calibrated because message content and stage inputs may interact.

A.4 Monitoring for MASs

The runtime monitoring is currently addressed through observability frameworks, proactive guardrails, selective verification, and trace-sampling methods. AgentOps catalogs the artifacts that agents should emit for debugging (Dong et al., 2024), but treats monitoring locations as predetermined and does not determine which observations are necessary for fault attribution. Runtime guardrails such as ProbGuard and AGrail predict unsafe states or adapt safety checks over time (Wang et al., 2025a; Luo et al., 2025), while ToolSafe uses TS-Bench to evaluate step-level tool-invocation monitoring (Mou et al., 2026). Although these approaches improve detection, their checks are generally attached to fixed locations or applied at every relevant step, which can be expensive when intermediate evaluations require LLM judges or process supervision. AgentSoV reduces this cost by selecting a budgeted subset of stages whose measurements distinguish the largest range of possible disturbance sources.

Weak-to-strong monitoring studies on SHADE-Arena and CUA-SHADE-Arena further show that monitor scaffolding, such as where and how observations are collected, can matter more than monitor-model strength (Kale et al., 2026). However, these benchmarks evaluate robustness of a given monitoring configuration rather than optimizing the configuration itself. AgentSoV formalizes this missing selection problem using stage-specific disturbance signatures and a log-determinant objective that favors measurements that are both informative and non-redundant.

SIEVE similarly avoids exhaustive verification by escalating only when a heuristic suspicion signal is triggered (Liang et al., 2025), but such triggers may overlook failures that remain locally inconspicuous while propagating downstream. Drawing on established optimal sensor-placement and Bayesian D-optimal design methods (Guestrin et al., 2005; Huan et al., 2024), AgentSoV adapts this principle to stage-level MAS monitoring. Because no dedicated benchmark currently evaluates

budgeted monitor placement in MAS, AgentSoV's partial-observation experiments also provide a foundation for measuring how much attribution capability is retained as monitoring cost decreases.

B Instantiating AgentSoV Across MAS Architectures

The preceding AgentSoV formulation provides a common interface, i.e., diagnostic stages s_k, inputs $\boldsymbol{u}_k$, states $\boldsymbol{x}_k$, virtual measurements $\boldsymbol{y}_k$, disturbances $\boldsymbol{z}_k$, and cross-stage propagation, without prescribing a single trajectory decomposition for every MAS. Different execution architectures instantiate these quantities differently. We illustrate this with three representative settings: chain-structured tool use, centralized orchestration, and DAG-structured information flow.

B.1 Chain-Structured MAS

In a chain-structured system, the basic dependency in Eq. 1 applies directly: each stage primarily inherits variation from the immediately preceding stage. We instantiate this setting using BFCL data (Patil et al., 2025; Liu et al., 2026), where an assistant repeatedly issues tool calls and incorporates the resulting tool responses over multiple turns.

Each diagnostic stage s_k corresponds to one assistant turn together with its tool calls and returned results. The process input summarizes the action issued before observing those results:

$$\boldsymbol{u}_k = \begin{bmatrix} n_k^{\text{call}} & n_k^{\text{arg}} & n_k^{\text{write}} & \rho_k \end{bmatrix}^\top \in \mathbb{R}^4,$$

where n_k^{call}, n_k^{arg}, and n_k^{write} denote the numbers of tool calls, supplied arguments, and environment-modifying calls, respectively, and ρ_k is the fraction of calls repeated from earlier stages. All four quantities are calculated directly from the execution trace.

BFCL contains four corresponding failure types: incorrect tool arguments, missing required calls, unnecessary tool calls, and parallel-call conflicts. Because the measurement coordinates and failure mechanisms have explicit semantics, we choose to configure

$$\boldsymbol{y}_k = \begin{bmatrix} y_k^{\text{schema}} & y_k^{\text{inaction}} & y_k^{\text{exec}} & y_k^{\text{accum}} \end{bmatrix}^\top,$$

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0.5 & 1 \end{bmatrix}, \quad \boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0.5 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The first three rows of $\boldsymbol{C}$ directly observe their corresponding state coordinates, while its final row allows the accumulated-error measurement to reflect both current execution error and previously accumulated variation. The four columns of $\boldsymbol{H}$ correspond, respectively, to the four failure types. For example, an unnecessary call decreases the inaction component while increasing execution-error variation, whereas a parallel-call conflict affects both current and accumulated execution errors.

Tool-calling trajectories provide structured execution signals, so following the definition of the $\boldsymbol{C}$ and $\boldsymbol{y}$, the four interpretable diagnostic coordinates can be written as:

$$\boldsymbol{x}_k = \begin{bmatrix} x_k^{\text{schema}} & x_k^{\text{inaction}} & x_k^{\text{exec}} & x_k^{\text{accum}} \end{bmatrix}^\top.$$

These coordinates represent schema defects, inaction, current execution errors, and accumulated unresolved errors. Specifically, y_k^{schema} is the fraction of argument keys at stage k that occur only once in the trajectory, y_k^{inaction} is one if no tool call is made and zero otherwise, y_k^{exec} counts failed tool calls, and y_k^{accum} counts unresolved execution errors accumulated through stage k.

No change to the general state-space logic is required. The learned $\boldsymbol{F}$ characterizes how diagnostic variation persists across tool-use turns, while $\boldsymbol{B}$ characterizes how the current tool-use pattern changes the diagnostic state. Thus, the BFCL instantiation differs from the running textual example primarily in how $\boldsymbol{u}_k$, $\boldsymbol{x}_k$, $\boldsymbol{y}_k$, $\boldsymbol{C}$, and $\boldsymbol{H}$ are materialized from a structured execution log.

B.2 Orchestration-Based MAS

An orchestration-based MAS introduces a dependency that is absent from a simple chain: persistent task-level information maintained by a central controller may influence many downstream stages.

Magentic-One (Fourney et al., 2024; Liu et al., 2026) provides a representative example and closely resembles the Orchestrator-WebSurfer-Coder system used throughout our illustration.

A Magentic-One execution contains an initial orchestrator fact sheet, repeated orchestrator-agent interactions, and a final response. We therefore map the trajectory as

$$\underbrace{\tau_1}_{s_0} \to \underbrace{(\tau_2, \tau_3)}_{s_1} \to \cdots \to \underbrace{(\tau_{2K}, \tau_{2K+1})}_{s_K} \to \underbrace{\tau_{2K+2}}_{s_{K+1}},$$

where s_0 represents the persistent orchestration information, $s_1, \ldots, s_K$ are orchestrator-worker stages, and s_{K+1} represents the delivered response. Relative to Eq. 4, this simply retains the individual orchestrator-worker interactions rather than aggregating them into a smaller number of functional stages.

For an execution stage $s_k (k \geq 1)$, the information dispatched by the orchestrator forms the process input, while the corresponding worker response provides the virtual measurement. In our Magentic-One instantiation,

$$\boldsymbol{u}_k = \boldsymbol{P}\,\mathcal{E}(\tau_{2k}), \qquad \boldsymbol{y}_k = \boldsymbol{P}\,\mathcal{E}(\tau_{2k+1}),$$

where $\mathcal{E}(\cdot)$ is a frozen sentence encoder and $\boldsymbol{P}$ is a PCA projection fitted using the training split. Thus, the numerical representations differ from those used in BFCL, but $\boldsymbol{u}_k$ and $\boldsymbol{y}_k$ retain exactly the roles defined by Eqs. 1-2.

The initial orchestration stage requires separate treatment because its information is available to many later stages rather than only to s_1. Magentic-One's fact sheet explicitly separates verified facts, information to derive or guess, and planned actions. We therefore construct

$$\boldsymbol{y}_0 = \begin{bmatrix} n_{\text{verified}} & n_{\text{derive/guess}} & n_{\text{action}} \end{bmatrix}^\top,$$

where the three terms summarize the plan's grounding, unresolved information, and scope. The corresponding orchestration state $\boldsymbol{x}_0$ is retained as a shared upstream state for the downstream execution.

The architectural change can then be represented by adding one term to the basic state equation:

$$\boldsymbol{x}_k = \boldsymbol{F}\boldsymbol{x}_{k-1} + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{G}\boldsymbol{x}_0 + \boldsymbol{H}\boldsymbol{z}_k + \boldsymbol{w}_k, \qquad \boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{v}_k, \quad k = 1, \ldots, K,$$

where $\boldsymbol{G}$ captures the direct influence of the persistent orchestration state on each downstream stage. The term $\boldsymbol{G}_k\boldsymbol{x}_0$ admits two equivalent reformulations to the original formula: augmenting the state as $\tilde{\boldsymbol{x}}_k = [\boldsymbol{x}_k^\top, \boldsymbol{x}_0^\top]^\top$ with transition $\begin{bmatrix} \boldsymbol{F}_k & \boldsymbol{G}_k \\ 0 & \boldsymbol{I} \end{bmatrix}$, or appending $\boldsymbol{x}_0$ to each $\boldsymbol{u}_k$. We maintain this term explicitly for two reasons. First, in orchestrator-worker designs, the task specification and global plan are re-supplied to every worker rather than relayed sequentially by preceding stages; thus, $\boldsymbol{x}_0$ acts as an exogenous input across all stages. Second, separating these channels preserves the interpretation of $\boldsymbol{B}_k\boldsymbol{u}_k$ as stage-specific control information, which is necessary for the intervention analysis in Section 3.2 folding $\boldsymbol{x}_0$ directly into $\boldsymbol{u}_k$ would mean that "removing a stage input" inadvertently wipes out the global task definition.

This decomposition separates two qualitatively different propagation mechanisms. Variation in $\boldsymbol{x}_0$, such as an incorrect global plan or poorly grounded fact sheet, acts as a common upstream source and can influence many later stages through $\boldsymbol{G}$. By contrast, abnormal variation introduced by one worker enters through $\boldsymbol{H}\boldsymbol{z}_k$ and subsequently propagates to later stages through $\boldsymbol{F}$. Consequently, an incorrect plan and an incorrect intermediate agent response can produce different cross-stage signatures even when they eventually yield the same wrong final answer. If an orchestration-based MAS has no explicit persistent planning state, the $\boldsymbol{G}\boldsymbol{x}_0$ term can simply be omitted, recovering the basic formulation.

B.3 DAG-Structured MAS

A DAG-structured MAS differs from the previous two cases because a stage may depend on several upstream stages rather than on a unique predecessor. We instantiate this setting on AEGIS (Kong et al., 2026), where each trajectory contains ten LLM calls connected by a trajectory-specific directed acyclic graph. Each call forms one diagnostic stage, and an edge from stage j to stage k exists when the response produced at j is included in the input to k.

For each call, the process input $\boldsymbol{u}_k \in \mathbb{R}^{16}$ is obtained by encoding the complete input-message list with a frozen encoder and projecting it to 16 dimensions using PCA. The response is represented analogously to construct $\boldsymbol{x}_k \in \mathbb{R}^{16}$. PCA is fitted using non-injected training calls. Because the response representation is itself directly observed, this instantiation uses $\boldsymbol{y}_k = \boldsymbol{x}_k, \boldsymbol{C}_k = \boldsymbol{I}$.

The only substantive change to Eq. 1 is the cross-stage propagation term. Let $\mathrm{pa}(k)$ denote the set of diagnostic stages whose outputs directly enter stage k. Instead of receiving propagated variation from one preceding stage, stage k aggregates it across its parents:

$$\boldsymbol{x}_k = \boldsymbol{B}\boldsymbol{u}_k + \frac{1}{|\mathrm{pa}(k)|} \sum_{j \in \mathrm{pa}(k)} \boldsymbol{F}\boldsymbol{x}_j + \boldsymbol{w}_k, \tag{8}$$

with an intercept included in the implementation through the usual augmented input representation. The term $\boldsymbol{B}\boldsymbol{u}_k$ captures the call-local relationship between the complete information supplied to the current call and its response, while $\boldsymbol{F}\boldsymbol{x}_j$ captures variation inherited from an upstream call. The parameters $\boldsymbol{F}$ and $\boldsymbol{B}$ are shared across call positions and trajectories and are fitted using non-injected training calls.

The formulation above defines the normal DAG dynamics used by AgentSoV. Because AEGIS also provides injected-fault labels, we use the same dynamics to construct AgentSoV-S and examine whether a supervised residual-to-fault mapping improves localization. AgentSoV-S fits a local predictor using only $\boldsymbol{B}\boldsymbol{u}_k$ and a graph-aware predictor using the full dynamics in Eq. 8, both on non-injected training calls. For each call, it combines the standardized local residual $\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^{\mathrm{local}}$ with the graph-explained residual difference $\hat{\boldsymbol{x}}_k^{\mathrm{full}} - \hat{\boldsymbol{x}}_k^{\mathrm{local}}$, and a linear classifier shared across call positions produces the call-level fault probability. Agent scores are obtained by taking the maximum probability over their calls, after which a validation-selected decoding rule predicts the faulty-agent set. Thus, AgentSoV-S retains the graph-aware representation of AgentSoV but learns how to map its residual evidence to fault predictions.

For both variants, Eq. 8 is the DAG analogue of the $\boldsymbol{F}_k\boldsymbol{x}_{k-1}$ term in Eq. 1, rather than a separate diagnostic model. For a chain there is one propagation path from an upstream stage to a downstream stage; for a DAG, variation may reach the same downstream stage through several directed paths. Accordingly, the multi-stage propagation quantities used in Section 3 are obtained by aggregating the effects of all directed paths connecting the source and observation stages. The meanings of input, state, measurement, and disturbance remain unchanged.

Taken together, these three instantiations illustrate the distinction between the architecture-aware construction and the architecture-independent analytical language of AgentSoV. A tool-calling chain, an orchestrated system, and a DAG expose different execution structures and therefore require different stage mappings and dependency patterns. Once represented, however, the same quantities retain the same interpretation: $\boldsymbol{u}_k$ captures information entering a stage, $\boldsymbol{x}_k$ captures task-relevant variation carried through the execution, $\boldsymbol{y}_k$ captures what is virtually observed, $\boldsymbol{H}_k\boldsymbol{z}_k$ represents abnormal variation when modeled explicitly, and the transition terms describe how upstream variation reaches downstream stages. The analyses in Section 3 build directly on this common representation.

C Computational Details for Residual-Based Failure Attribution

Section 3.1 describes failure attribution in terms of the propagated disturbance signature $\boldsymbol{G}_{\ell,k} = \boldsymbol{C}_\ell\boldsymbol{F}_\ell\boldsymbol{F}_{\ell-1}\cdots\boldsymbol{F}_{k+1}\boldsymbol{H}_k$, which characterizes how abnormal variation introduced at stage k becomes observable at a downstream stage ℓ. This appendix gives the corresponding residual-based computational formulation used to identify the source of abnormal variation from an observed execution.

C.1 Normal Prediction and Diagnostic Residuals

We first define the execution expected under the fitted normal dynamics. Setting $\boldsymbol{z}_k = \boldsymbol{0}$ in Eq. 1, the one-step state and measurement predictions are

$$\hat{\boldsymbol{x}}_k = \boldsymbol{F}_k\hat{\boldsymbol{x}}_{k-1} + \boldsymbol{B}_k\boldsymbol{u}_k, \tag{9}$$

$$\hat{\boldsymbol{y}}_k = \boldsymbol{C}_k\hat{\boldsymbol{x}}_k. \tag{10}$$

where $\hat{\boldsymbol{x}}_k$ and $\hat{\boldsymbol{y}}_k$ denote the state and virtual measurement predicted in the absence of an explicit abnormal disturbance. For an observed execution, we define the stage-wise diagnostic residual

$$\boldsymbol{e}_k = \boldsymbol{y}_k - \hat{\boldsymbol{y}}_k. \tag{11}$$

Thus, $\boldsymbol{e}_k$ measures the part of the observed virtual measurement that is not explained by the fitted normal AgentSoV dynamics.

Importantly, a nonzero residual by itself does not identify where abnormal variation originated. The residual at stage k may reflect a disturbance introduced locally, variation propagated from an earlier stage, ordinary process and measurement noise, or model mismatch. The state-space structure allows these possibilities to be separated through their different cross-stage patterns.

Suppose an abnormal disturbance $\boldsymbol{z}_k$ is introduced at stage k, with no additional abnormal disturbance introduced elsewhere. As shown in Section 3.1, its contribution to the virtual measurement at a downstream stage $\ell \geq k$ is $\boldsymbol{G}_{\ell,k}\boldsymbol{z}_k$. By causality, the same disturbance cannot affect measurements at stages preceding k. The diagnostic residual therefore decomposes as

$$\boldsymbol{e}_\ell = \begin{cases} \boldsymbol{\epsilon}_\ell, & \ell < k, \\ \boldsymbol{G}_{\ell,k}\boldsymbol{z}_k + \boldsymbol{\epsilon}_\ell, & \ell \geq k, \end{cases} \tag{12}$$

where $\boldsymbol{\epsilon}_\ell$ collects the residual variation expected under normal execution, including propagated process noise, measurement noise, and model mismatch.

Equation 12 provides the key diagnostic principle. A disturbance is not identified merely because stage k has a large residual. Rather, a candidate source at stage k is plausible when the residuals observed from k onward are consistent with the downstream pattern predicted by $\boldsymbol{G}_{\ell,k}\boldsymbol{z}_k$.

For example, in the Orchestrator-WebSurfer-Coder system, an abnormal retrieval disturbance at s_1 can generate residuals at the retrieval, reasoning, and final-response stages. In contrast, a disturbance introduced by the Coder at s_2 cannot explain an abnormal residual already present at s_1. This temporal and structural constraint is what distinguishes AgentSoV attribution from independently scoring each stage for anomalousness.

C.2 Trajectory-Level Residual Representation

To use the residual pattern jointly across the execution, we stack the stage-wise residuals into

$$\boldsymbol{e} = \begin{bmatrix} \boldsymbol{e}_1^\top & \boldsymbol{e}_2^\top & \cdots & \boldsymbol{e}_N^\top \end{bmatrix}^\top.$$

For a candidate disturbance source at stage k, we similarly collect its predicted effects over the complete trajectory into the trajectory-level disturbance signature

$$\boldsymbol{G}_k = \begin{bmatrix} \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{G}_{k,k}^\top & \boldsymbol{G}_{k+1,k}^\top & \cdots & \boldsymbol{G}_{N,k}^\top \end{bmatrix}^\top.$$

The leading zero blocks encode causality: abnormal variation introduced at stage k cannot influence observations made before that stage. The remaining blocks describe how that same disturbance should appear throughout its downstream execution.

Let

$$\boldsymbol{\epsilon} = \begin{bmatrix} \boldsymbol{\epsilon}_1^\top & \cdots & \boldsymbol{\epsilon}_N^\top \end{bmatrix}^\top.$$

Then Eq. 12 becomes

$$\boldsymbol{e} = \boldsymbol{G}_k\boldsymbol{z}_k + \boldsymbol{\epsilon} \tag{13}$$

when abnormal variation originates from a single stage.

This representation makes explicit why the downstream structure is useful for diagnosis. Two candidate stages may both exhibit large local residuals, but their trajectory-level signatures $\boldsymbol{G}_k$ generally differ because they enter the system at different locations and propagate through different sequences of $\boldsymbol{F}_\ell$ matrices before becoming visible through the corresponding $\boldsymbol{C}_\ell$ matrices.

To account for the different scales and correlations of residual components, we estimate the covariance of normal residuals from fault-free training trajectories. If the stage-wise residual covariance is denoted by $\boldsymbol{\Sigma}$, a simple implementation assumes a common covariance across stages and uses $\boldsymbol{\Sigma}_{\boldsymbol{e}} = \boldsymbol{I}_N \otimes \boldsymbol{\Sigma}$.

More general empirical covariance models can be used when sufficient fault-free trajectories are available. In either case, the covariance serves to discount residual directions that commonly fluctuate during normal execution and to give greater weight to deviations that are unusual relative to the fitted normal dynamics.

C.3 Joint Attribution with Sparse Disturbances

Although some benchmarks assume a single responsible stage, an MAS execution may contain abnormal variation introduced at multiple stages. The residual model then generalizes naturally to $\boldsymbol{e} = \sum_{k=1}^{N} \boldsymbol{G}_k \boldsymbol{z}_k + \boldsymbol{\epsilon}$. Each $\boldsymbol{z}_k$ is a vector associated with one candidate source stage, while $\boldsymbol{G}_k \boldsymbol{z}_k$ is the complete downstream residual pattern predicted from that source.

Because failures are typically expected to originate from only a small subset of the diagnostic stages, we estimate the candidate disturbances using a group-sparse objective:

$$\{\hat{\boldsymbol{z}}_k\}_{k=1}^{N} = \arg\min_{\{\boldsymbol{z}_k\}} \left\| \boldsymbol{e} - \sum_{k=1}^{N} \boldsymbol{G}_k \boldsymbol{z}_k \right\|_{\boldsymbol{\Sigma}_{\boldsymbol{e}}^{-1}}^{2} + \lambda \sum_{k=1}^{N} \|\boldsymbol{z}_k\|_2, \tag{14}$$

where $\|\boldsymbol{a}\|_{\boldsymbol{\Sigma}_{\boldsymbol{e}}^{-1}}^{2} = \boldsymbol{a}^\top \boldsymbol{\Sigma}_{\boldsymbol{e}}^{-1} \boldsymbol{a}$ and $\lambda > 0$ controls the sparsity of the attribution.

The first term in Eq. 14 measures how well the candidate disturbances jointly explain the observed trajectory-wide residual, after accounting for residual variation expected during normal execution. The second term is an ℓ_2/ℓ_1 group penalty: the ℓ_2 norm treats all components of the disturbance at one stage as a group, while the outer sum encourages entire stage-level disturbance vectors to be zero. Hence, stages with $\hat{\boldsymbol{z}}_k \neq \boldsymbol{0}$ are selected as candidate sources of abnormal variation.

This formulation is useful because attribution is performed through explanation of the complete residual trajectory. A stage is selected only when the disturbance entering through its $\boldsymbol{H}_k$ direction and propagating through the learned $\boldsymbol{F}$ dynamics explains the observed downstream deviations. A stage with a large local residual but an incompatible downstream signature can therefore receive less attribution than an earlier stage whose propagated effect explains several later residuals simultaneously. When a benchmark requires a single responsible stage, the group-sparse solution can be reduced to a stage-level ranking using $\hat{k} = \arg\max_k \|\hat{\boldsymbol{z}}_k\|_2$. When multiple causes are allowed, the nonzero groups themselves provide the multi-stage attribution.

C.4 Relationship to the Analyses in Section 3

The residual formulation above is one computational realization of the system-level ideas in Section 3. Failure attribution uses the disturbance channel $\boldsymbol{H}_k \boldsymbol{z}_k$ and asks which propagated disturbance signatures best explain the observed residuals. Stage-input influence instead perturbs the process-input channel $\boldsymbol{B}_k \boldsymbol{u}_k$ and propagates that change through the same learned transition matrices. Virtual-sensor placement asks which observation matrices $\boldsymbol{C}_\ell$, and therefore which virtual measurements $\boldsymbol{y}_\ell$, make the candidate propagated disturbance signatures most distinguishable relative to normal residual variation.

Thus, these analyses do not require separate execution models. They are different uses of the same fitted AgentSoV dynamics: $\boldsymbol{H}_k$ specifies where abnormal variation can enter, $\boldsymbol{F}_k$ determines how it propagates, $\boldsymbol{B}_k$ describes how stage inputs influence the system, and $\boldsymbol{C}_k$ determines what parts of that variation become observable.

D Additional Experimental Setup

Datasets. BFCL (Liu et al., 2026; Patil et al., 2025) contains 164 tool-calling trajectories from a single-agent system: 61 are fault-free and 103 contain a scorable failure. The failures include incorrect tool arguments, missing required calls, parallel-call conflicts, and unnecessary invocations. We use five-fold cross-validation over all 164 trajectories.

Magentic-One, drawn from AgentHallu (Liu et al., 2026), contains 94 trajectories from an orchestrator-with-specialists system comprising an Orchestrator, WebSurfer, Coder, and Comput-

erTerminal. Of these, 40 are fault-free and 54 contain a labeled failure. We use five-fold cross-validation stratified by fault-free, planning-failure, and execution-failure trajectories.

AEGIS (Kong et al., 2026) provides official training, validation, and test splits containing 1749, 454, and 156 MacNet trajectories, respectively. After removing task overlap across splits and excluding one training trajectory with a different MacNet configuration, we use 856 training, 364 validation, and 156 test trajectories. Each trajectory is a topologically ordered DAG with ten calls executed by five agents and may contain multiple synthetically injected faulty agents.

Baselines. Uniform Random ranks the candidate steps or agents in random order. The GPT-5.5 baselines follow the three prompting strategies in Zhang et al. (2025): All-at-Once supplies the complete trajectory with attribution instructions, Step-by-Step analyzes each step sequentially, and Binary Search iteratively narrows the candidate failure window. MASC (Shen et al., 2026) trains a history-conditioned multilayer perceptron (MLP) on clean trajectories and uses prediction error as its anomaly score. OAT (Yeh et al., 2026) re-encodes trajectory steps with an LLM and identifies deviations from normal dynamics modeled by a neural controlled differential equation. We include a baseline only when its released method provides sufficient implementation detail and supports the corresponding analysis task.

AgentSoV variants and computation. AgentSoV fits its normal dynamics using normal training executions and performs attribution from the resulting residuals and their propagation. AgentSoV-S uses the same diagnostic representation but, for each stage, forms features from the standardized local prediction residual and the change in that residual after adding cross-stage propagation. A class-balanced logistic classifier shared across stage positions is trained with fault labels to map these features to the probability that each stage is faulty. For specific datasets, we encode text using the all-MiniLM-L6-v2 encoder and reduce the resulting embeddings to 16 dimensions using PCA. The sparsity parameter λ in Eq. 14 is selected independently from $\{0.03, 0.1, 0.3, 1.0, 3.0\}$ by maximizing step accuracy. In virtual-sensor placement, we fix $\eta = 10^{-6}$ for numerical stabilization. All AgentSoV experiments ran on one Apple M3 Pro CPU thread, requiring no LLM.

E Additional Results

E.1 Failure Attribution

The impact of $\boldsymbol{F}$. Tables 5 and 6 examine whether the transition dynamics can be shared across stages rather than using stage-specific $\boldsymbol{F}_k$ across three datasets. On BFCL, the two parameterizations perform nearly identically: sharing $\boldsymbol{F}$ changes AgentSoV-S from 36.9% to 37.7% accuracy and leaves AgentSoV essentially unchanged at 37.5%. This suggests that a common transition model is sufficient for the relatively homogeneous tool-calling stages.

Table 5: Failure attribution on BFCL and Magentic-One using a shared transition matrix $\boldsymbol{F}$ across all stages, i.e., $\boldsymbol{F}_k = \boldsymbol{F}$ for all k.

Dataset	Method	Step Accuracy	Top-3 Step Accuracy	MRR	Agent Accuracy
BFCL	AgentSoV-S	$\mathbf{37.7\% \pm 1.3\%}$	$\mathbf{65.2\% \pm 2.7\%}$	$\mathbf{56.1\% \pm 1.4\%}$	$\mathbf{37.7\% \pm 1.3\%}$
	AgentSoV	$37.5\% \pm 1.0\%$	$63.1\% \pm 1.7\%$	$54.6\% \pm 0.5\%$	$37.5\% \pm 1.0\%$
Magentic-One	AgentSoV-S	$32.2\% \pm 1.9\%$	$58.5\% \pm 1.5\%$	$46.4\% \pm 1.0\%$	$54.8\% \pm 5.1\%$
	AgentSoV	$\mathbf{37.8\% \pm 2.1\%}$	$\mathbf{61.5\% \pm 1.8\%}$	$\mathbf{51.0\% \pm 1.5\%}$	$\mathbf{62.6\% \pm 0.8\%}$

Table 6: Failure attribution on AEGIS using a shared transition matrix $\boldsymbol{F}$ across all stages, i.e., $\boldsymbol{F}_k = \boldsymbol{F}$ for all k.

Method	Exact-Set Accuracy	Top-1 Accuracy	Top-3 Accuracy	MRR
AgentSoV-S	$38.5\% \pm 0.0\%$	$77.6\% \pm 0.3\%$	$98.2\% \pm 0.3\%$	$87.4\% \pm 0.3\%$
AgentSoV	$\mathbf{46.2\% \pm 1.3\%}$	$\mathbf{91.7\% \pm 0.5\%}$	$100.0\% \pm 0.0\%$	$95.4\% \pm 0.3\%$

On Magentic-One, AgentSoV remains robust to sharing, with step accuracy changing from 39.3% to 37.8% and agent accuracy from 61.9% to 62.6%. In contrast, AgentSoV-S decreases from 36.3% to 32.2% in step accuracy and from 63.0% to 54.8% in agent accuracy, suggesting that its supervised attribution benefits more from stage-specific transition dynamics.

On AEGIS, sharing $\boldsymbol{F}$ improves exact-set accuracy for both variants: AgentSoV-S increases from 34.0% to 38.5%, while AgentSoV increases from 43.6% to 46.2%. AgentSoV also improves in top-1 accuracy from 90.4% to 91.7%, whereas AgentSoV-S changes slightly from 78.9% to 77.6%. These results indicate that parameter sharing can regularize propagation estimation when stages exhibit similar transition structure, while stage-specific $\boldsymbol{F}_k$ can be advantageous when propagation patterns are more heterogeneous.

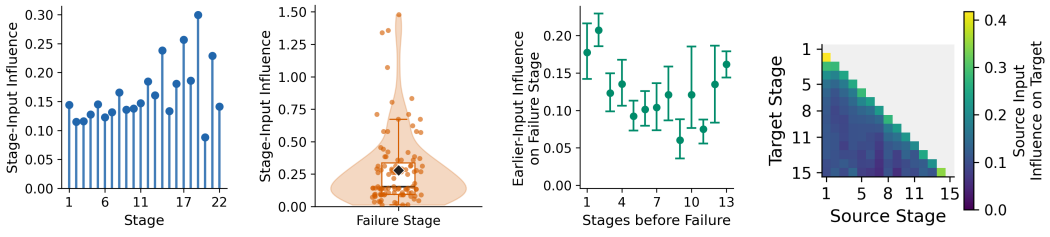
The impact of $\boldsymbol{H}$. Table 7 evaluates the impact of $\boldsymbol{H}$ by replacing failure-type-aware disturbance directions ($\boldsymbol{H}$) with an identity mapping $\boldsymbol{H}{=}\boldsymbol{I}$. AgentSoV reduces step accuracy from 37.5% to 35.3% (-2.2%), indicating that failure-type-specific disturbance signatures provide additional discrimination beyond the learned propagation dynamics. Together, these ablations show that both components contribute to failure attribution: $\boldsymbol{F}$ provides the cross-stage propagation structure, while $\boldsymbol{H}$ refines how different failure mechanisms enter the diagnostic state.

Table 7: Failure attribution accuracy on BFCL when replacing the failure-aware disturbance matrix $\boldsymbol{H}$ with $\boldsymbol{H}{=}\boldsymbol{I}$. Δ denotes the resulting change in accuracy.

Method	**Accuracy**	**Δ vs. Failure-Specific $\boldsymbol{H}$**
AgentSoV ($\boldsymbol{F}{=}\boldsymbol{0}$, $\boldsymbol{H}{=}\boldsymbol{I}$)	$26.2\% \pm 1.4\%$	-2.3%
AgentSoV ($\boldsymbol{H}{=}\boldsymbol{I}$)	$35.3\% \pm 2.0\%$	-2.2%

E.2 Stage-Input Influence

Figure 5 shows that stage inputs have heterogeneous influence on BFCL. In Figure 5a, the mean influence on the final measurement ranges from approximately 0.11 to 0.18 across the well-represented early stages, while several later stages have larger values. The failure-stage distribution in Figure 5b is strongly right-skewed, with a mean of 0.278 and a median of 0.153, showing that a small number of failure stages exert large downstream effects while many have modest influence. Figures 5c and 5d further show that propagation is largely local: inputs one and two stages before failure have mean influences of 0.177 and 0.207, respectively, whereas most longer-lag means are lower, and the largest source-to-target effects concentrate near the matrix diagonal. The decay is not strictly monotonic, indicating that stage separation alone does not determine influence on BFCL. Overall, AgentSoV reveals which tool-calling inputs remain consequential downstream and how rapidly their effects are attenuated across stages.



(a) Input influence by stage. (b) Failure-stage input influence. (c) Influence of earlier inputs on failure stage. (d) Source-to-target stage influence.

Figure 5: Stage-input influence on BFCL.