Workflow Instability Decomposition: Per-Node Variance and a DAG-Depth Amplification Law for Multi-Step LLM Workflows

Andrew Sheng-Han Huang*
National Taiwan University
johnshhuang1111@gmail.com

Abstract

Production LLM workflows—chains and graphs of model calls, parsers, and routers built on frameworks such as LangGraph, n8n, and DSPy—increasingly return a *different answer on every run* for the *same* input. Practitioners treat this run-to-run instability as a monolithic nuisance and either tolerate it or bluntly set the temperature to zero everywhere, sacrificing useful diversity. We give it structure. We introduce WID (Workflow Instability Decomposition): a per-node variance-components decomposition for a workflow directed acyclic graph (DAG) that, *under a linear additive structural model with independent per-node noise, a fixed DAG, and a finite scalar readout*, attributes end-to-end output variance to each node and node *type* through an exact law-of-total-variance identity, together with a *depth-amplification law*—a fitted per-edge coefficient g describing how instability injected deeper in a homogeneous positive-gain chain lands on the output with larger weight. The estimator is *observational* (it fits local input–output maps from repeated runs and never uses any ground-truth parameters), so the *procedure* transfers from our controlled simulator to real pipelines. Our contribution is primarily a *method and an identifiability result*: on a ground-truth simulation sweep (200 random workflows) the estimator *recovers the injected per-node variance shares* to a median 1.7% relative error, the interaction (non-additivity) residual it leaves is a measured 0.05% of total variance, and it recovers the per-edge coefficient g to within its 95% confidence interval across an amplification grid. Because fitters notoriously invent slopes on noise, g is reported only when it survives a significance gate; a node-level permutation gate is *mis-calibrated* on a strict-uniform flat null (it fires at 9.3% against a nominal 5% because depth points are clustered within workflows and heteroscedastic in depth), so we gate instead with a *cluster (hierarchical) bootstrap over whole workflows*, whose flat-null false-positive rate is 2.7% (realistic null) and 4.0% (strict-uniform null), both within the ±5.3-point Monte-Carlo band around 5% at 150 trials. We are explicit that the synthetic node-type budget—LLM nodes carrying the majority share—is *recovery of an injected modeling assumption*, not a discovered fact; the only non-baked evidence that LLM nodes dominate is a small live llama3.1:8b pipeline (600 logged runs over 30 fixed inputs), on which the two LLM nodes carry 100.0% of the measured variance and the deterministic router 0.0%—corroborating the node-*type* budget but, because the formatting node is folded into the router readout, too shallow to test the depth law. Intervention accounting (how much variance pinning removes) is reported on the simulator only. WID turns "it gives a different answer every run" into a per-node accounting method with a calibrated gate, enabling targeted rather than indiscriminate stabilization.

1 Introduction

A modern LLM application is rarely a single model call. It is a *compound AI system* [19]: a graph whose nodes extract fields, call tools, reason, route on conditions, and format structured output, wired

*National Taiwan University. Code, the synthetic simulator, the live-pipeline logs, and end-to-end reproduction scripts are released with the paper; all result numbers are regenerated from `results/*.json` by the analysis scripts.

together by orchestration frameworks [1, 11]. These systems inherit a property that single calls have but that compounds badly across a graph: **run-to-run instability**. Run the same workflow twice on the *same* input and the end-to-end output can differ—because LLM nodes sample at temperature > 0, because a parser or router takes a different branch, and because even at temperature 0 the inference stack is not bit-reproducible (floating-point non-associativity under dynamic batching [6]). "It gives a different answer every run" is among the most common practitioner complaints about agentic pipelines.

The field's two reflexes are both unsatisfying. The first is to tolerate the variance and report a single run. The second is to set temperature to 0 globally, which destroys the diversity that generation nodes are *for* and still does not remove non-LLM instability (branch flips, dictionary/set ordering, tie-breaks). Neither tells you *where* the instability lives or *how much* you would recover by intervening. What is missing is a *decomposition*: a principled attribution of end-to-end variance to the workflow's parts, and a model of how that variance propagates through the graph.

This paper. We provide that decomposition and that model, and we are deliberately conservative about what is *recovered* versus *discovered*.

- **A per-node variance-components decomposition over a workflow DAG** (Section 3), stated under explicit assumptions (a linear additive structural model, independent per-node noise, a fixed DAG with fixed parents, and a finite scalar readout; Assumption 1). Under these, the law of total variance gives an *exact* additive attribution of end-to-end output variance to the nodes, correctly handling the fact that downstream variance is *conditional* on upstream draws. We estimate it observationally—fitting each node's local input–output map from repeated runs—so the *procedure* runs on a real pipeline's logged intermediates, and we report the non-additivity it leaves behind as an explicit interaction residual rather than assuming it away.
- **An identifiability (recovery) result for the node-type budget.** On a controlled simulator where we *inject* the per-node variances, the estimator recovers the injected per-node shares (median 1.7% relative error) and hence the per-type budget. *We stress that the synthetic finding "LLM nodes carry the majority share" (*71.7% *in our sweep) is recovery of an injected modeling choice* (we set LLM-node injection variance largest, reflecting unpinned sampling), *not* an empirical discovery. The empirical question—do LLM nodes in fact dominate?—is answered only by the live pipeline below.
- **A DAG-depth amplification law with a calibrated significance gate** (Section 4). For a homogeneous positive-gain chain, instability injected d edges upstream of the output lands on it with a contribution that scales like γ^{2d} for a per-edge gain γ; we fit the per-edge *amplification coefficient* $g = \log \gamma$. g is reported only when a significance gate rejects the flat null. We show the obvious node-level permutation gate is mis-calibrated under depth-clustered, heteroscedastic points and replace it with a *cluster bootstrap over whole workflows* that returns the false-positive rate to nominal on *both* a realistic and a strict flat null—reported honestly with their Monte-Carlo bands.
- **Intervention accounting (simulator only).** On the simulator the decomposition *predicts* how much variance a given pin/cache intervention removes, and we validate the prediction against ground truth. This is a synthetic identifiability check; we do *not* claim a validated real-pipeline pin/cache forecast.

Why now, and what survives a scoop. Production agentic workflows are exploding, and reliability is becoming a first-class evaluation axis. Our contribution is a *measurement method with an identifiability guarantee and a calibrated gate*, not a leaderboard number: the decomposition identity, the recovery result, the gate's flat-null calibration, and the live-pipeline node-type budget are reproducible artifacts on a public setup. If a competing method appears, the method, the recovery result, and the calibrated gate still stand.

2 Related Work

We position WID against the three literatures that each own one of its pillars; none does all three, and—decisively—all of them measure *correctness* (failures, errors, success) whereas WID measures

run-to-run output variance under a fixed input. A node can be perfectly correct on every run yet contribute large variance (e.g. a JSON node that reorders keys), so this is a different quantity, not a stricter one.

Variance amplification with scale. The reliability-science framework of Khanal et al. [10] defines a *Variance Amplification Factor* (VAF) that grows with task horizon and even argues high VAF is a capability signature. VAF is a *scalar, whole-agent, black-box* ratio keyed to task *duration*; it decomposes nothing, has no DAG topology, and fits no per-edge coefficient. WID's law is *node-level and topology-aware*: a fitted per-edge g on a graph whose variance has been *attributed* to nodes, not a single number for the whole agent.

Variance components of LLM output. Haase et al. [9] run an ANOVA-style partition of LLM output variance into prompt, model, and sampling effects—but for a *single call, single step*. There is no workflow, no node graph, and no depth. WID lifts variance components to a *multi-node DAG* and contributes the conditional (nested) variance treatment and the depth law that the single-call setting cannot express.

Per-node attribution in workflow DAGs. A growing body localizes *failures* in agent graphs: step-level evaluation with error-propagation tracking [7], causal failure attribution [17], lifecycle failure taxonomies [13], and error-cascade dynamics on dependency graphs [18]. All operate on *single traces* and attribute *correctness defects*. WID runs each workflow *many times on a fixed input* and attributes *variance*; the depth-amplification coefficient is a property of the variance flow, not of an error cascade. Reliability benchmarks such as Gupta [8] measure pass^k consistency and perturbation robustness—again correctness, with no variance decomposition. "Hallucinations live in variance" [5] shares our intuition that instability is variance-dominated, but for a single model's paraphrase consistency, not a per-node workflow decomposition.

Adjacent. Harm amplification in multi-agent systems [14] and prompt-sensitivity robustness [15] share vocabulary ("amplification", "brittle") but target safety and single-step robustness, respectively; semantic-entropy [12], durable-workflow governance [3], and speculative execution [4] optimize correctness, lifecycle, and latency—none decomposes run-to-run variance. Our statistical machinery is classical variance components [16] and the bootstrap/permutation [2] applied to a new object.

3 Method: Per-Node Variance Decomposition

Setup. A workflow is a DAG with nodes V in topological order and a single sink out $\in V$. We run it R times on a *fixed* input; the only randomness between runs is each node's stochastic *injection*. Let X_v be a scalar *readout* of node v's output (for structured nodes, a parsed value; for a router, the branch indicator; for free text, a canonicalized or embedding-clustered value reduced to a discrete code). The readout map ϕ is a disclosed modeling choice (Section 6).

Definition 1 (Standing assumptions)**.** *The exact decomposition and the depth law below are guarantees only under:* ***(A1) linear additive structure****—each node's readout is its parents' readouts combined linearly plus an independent additive injection (Eq. (1));* ***(A2) independent per-node noise****—the injections ε_v are mutually independent, with* ***no hidden common cause*** *driving two nodes' run-to-run variation;* ***(A3) a fixed DAG and fixed parents****—the topology and each node's parent set do not change across runs (a router that changes the graph violates this and is treated as a node whose branch indicator is a readout);* ***(A4) a finite scalar readout****—ϕ has a finite second moment. Every guarantee in this paper is scoped to (A1)–(A4); we report what breaks when they fail (Section 6) and measure the residual non-additivity directly rather than assuming it is zero.*

Generative form. For the controlled backbone we use a linear-Gaussian structural model: each node combines its parents and adds an independent injection,

$$X_v \;=\; b_v + \sum_{u \in \mathrm{pa}(v)} \beta_{vu}\, X_u \;+\; \varepsilon_v, \qquad \varepsilon_v \sim \mathcal{N}(0, \sigma_v^2) \text{ independent across } v. \tag{1}$$

This is the simplest form for which ground truth is *analytic*, so recovery and conservation are checkable; the *estimator* below never uses $\{\beta, \sigma\}$.

Algorithm 1 WID per-node variance decomposition (observational)

Require: workflow structure (nodes, parents, sink); readouts $\{X_v^{(r)}\}_{r=1}^{R}$

1: **for** v in topological order **do**
2: **if** v has no parents **then** $\hat{\sigma}_v^2 \leftarrow \widehat{\mathrm{Var}}(X_v)$
3: **else** OLS $X_v \sim X_{\mathrm{pa}(v)}$; set $\hat{\beta}_{v\cdot} \leftarrow$ slopes, $\hat{\sigma}_v^2 \leftarrow$ residual variance
4: **end if**
5: **end for**
6: $\hat{a}_{\mathrm{out}} \leftarrow 1$; for v in reverse topo order: $\hat{a}_v \leftarrow \sum_{c \in \mathrm{ch}(v)} \hat{\beta}_{cv}\, \hat{a}_c$
7: **return** $\widehat{\mathrm{contribution}}(v) = \hat{a}_v^2\, \hat{\sigma}_v^2$ for all v; node-type budget = $\sum_{v \in \mathrm{type}} \widehat{\mathrm{contribution}}(v) \,/\, \sum_v(\cdot)$

Proposition 1 (Exact per-node decomposition)**.** *Under* (1)*, the sink is a linear combination of injections,* $X_{\mathrm{out}} = \sum_v a_v\, \varepsilon_v + \mathit{const}$*, where the path gain* $a_v = \sum_{p:\, v \to \mathrm{out}} \prod_{e \in p} \beta_e$ *sums the products of edge gains over all directed paths from* v *to the sink (*$a_{\mathrm{out}} = 1$*). Because the* ε_v *are independent,*

$$\mathrm{Var}(X_{\mathrm{out}}) \;=\; \sum_{v \in V} a_v^2\, \sigma_v^2, \qquad \mathit{contribution}(v) := a_v^2\, \sigma_v^2. \tag{2}$$

Under Assumption 1 this is the law of total variance specialized to a linear DAG: an *exact* additive split with no cross terms, where node v's contribution is its own injection variance scaled by how strongly the graph propagates it to the output. Conservation, $\sum_v \mathrm{contribution}(v) = \mathrm{Var}(X_{\mathrm{out}})$, is then an identity. It is *not* an identity when (A1)–(A4) fail: a non-linear node, a hidden common cause, or a changing parent set introduces cross terms. We therefore do not assume conservation; we report the gap. Define the **interaction (non-additivity) residual**

$$\Delta \;:=\; \mathrm{Var}(X_{\mathrm{out}}) \;-\; \sum_v \widehat{\mathrm{contribution}}(v) \;=\; \mathrm{total}_{\mathrm{obs}} - \mathrm{total}_{\mathrm{est}}, \tag{3}$$

the part of the observed output variance the additive split fails to attribute (positive: super-additive interaction; negative: the fitted local maps over-explain the total). Δ is an *observed accounting line*, reported alongside the budget, not a quantity assumed to vanish; on the linear-Gaussian backbone it is Monte-Carlo small (Section 5), and on a non-linear pipeline it makes the model's mis-fit visible rather than hidden.

Observational estimator. Given the R runs we estimate the decomposition *without* the true parameters (Algorithm 1). For each node we OLS-fit its readout on its parents' readouts across runs; the residual variance estimates σ_v^2 and the slopes estimate β_{vu}. We then propagate estimated path gains $\hat{a}_v$ through the fitted DAG and report $\widehat{\mathrm{contribution}}(v) = \hat{a}_v^2 \hat{\sigma}_v^2$. The procedure uses only the graph *structure* (who feeds whom) plus logged intermediates—exactly what a real pipeline can emit—and reduces to (2) in the linear-Gaussian case.

4 The DAG-Depth Amplification Law

Estimand and its scope. Let d_v be the longest-path depth from v to the sink and m_v the number of directed $v \to$ out paths. The law holds for a *homogeneous positive-gain chain*: a *common, positive* per-edge gain $\gamma > 0$ and roughly equal path lengths, so $a_v \approx m_v\, \gamma^{d_v}$, and by (2)

$$\log \mathrm{contribution}(v) \;\approx\; \mathrm{const} \;+\; 2 \log m_v \;+\; 2g\, d_v, \qquad g := \log \gamma. \tag{4}$$

The slope of (multiplicity-adjusted) log-contribution on depth is $2g$; we report the *per-edge amplification coefficient* $g = \mathrm{slope}/2$ and $\gamma = e^g$. Subtracting $2 \log m_v$ *isolates* per-edge amplification from the purely structural fan-out: a node feeding many downstream paths has a large a_v for reasons unrelated to γ, and without this adjustment flat workflows look amplified (Section 5). *This is the law's load-bearing assumption.* With *signed* or *heterogeneous* path gains the monotone relation (4) can fail: edges with $|\beta| < 1$ *attenuate* (the depth coefficient inverts, $g < 0$), and mixed-sign paths can cancel so that a deep node contributes *less* than a shallow one. The reported g is therefore a fitted regularity for positive-gain chains, not a universal monotonicity of depth; we scope every depth

claim to this regime and treat the gate's output as "no significant amplification" whenever the data do not support a positive slope.

Significance gate (anti-numerology). A least-squares slope is always non-zero; reporting it as "amplification" on flat data is the failure mode that sequential and benchmark-decay analyses have repeatedly hit. The pooled depth points have *two* nuisance structures that break a naive test: (i) points from the *same workflow* are correlated—they are not exchangeable across realizations—and (ii) per-node estimation noise *grows with depth* (deep nodes compound estimation error), i.e. the points are heteroscedastic in depth. A node-level permutation test (shuffle depth labels, refit) assumes exchangeable points and is consequently *mis-calibrated*: on a strict-uniform flat null it over-rejects well above nominal (we measure this; Section 5). The honest unit of analysis is the *workflow*, so we gate with a **cluster (hierarchical) bootstrap**: resample whole workflow *blocks* with replacement, refit the pooled slope on each replicate, and report g only if the bootstrap 95% confidence interval for g excludes zero from below. Resampling at the workflow level preserves both the within-workflow clustering and the depth-variance structure, so the false-positive rate returns to nominal on both flat nulls. (We retain the node-level permutation test only as the mis-calibrated comparator that motivates this choice.)

Definition 2 (WID depth-amplification report)**.** *Pool* $\{(d_v, m_v, \widehat{contribution}(v))\}$ *as per-workflow blocks; regress* $\log \widehat{contribution} - 2\log m$ *on* d *for the point estimate. Cluster-bootstrap whole workflow blocks for the CI on* $g = slope/2$*. If the slope is positive and the bootstrap CI lower bound exceeds* 0 *at level* $\alpha = 0.05$*, report* $\hat{g} = slope/2$*,* $\hat{\gamma} = e^{\hat{g}}$*, and the CI; otherwise report "no significant amplification" (*g=0, γ=1*).*

5 Experiments

All statistics are pure-stdlib Python (no `numpy`/`scipy`); seeds are fixed and printed; every number below is a macro generated from `results/*.json`. The validity suite (27/27 checks pass, seed 20260617) is run before the analysis and is described first because it is what licenses the rest.

5.1 Lie-detectors: what the tests caught

We wrote executable tests that *can fail*, alongside (not after) the estimator.

- **Conservation as a measured residual.** The interaction residual Δ (Eq. (3)) is reported, not assumed: on the linear-Gaussian backbone its median magnitude is 0.05% of observed $\mathrm{Var}(X_{\mathrm{out}})$ (0.37% at the 90th percentile), i.e. the additive split accounts for the total up to Monte-Carlo error. A deliberately broken variant that drops the path-gain weighting blows the residual up by $> 4\times$, confirming the test bites.
- **Recovery (the core identifiability result).** On chains and random DAGs with *injected* per-node variance, the estimator recovers per-node contributions to a median 1.7% relative error (5.9% at the 90th percentile). This is what licenses reading the recovered budget; it is a statement about the estimator, not about real workflows.
- **Break-test / gate calibration.** This caught a real, still-present pitfall and forced the gate design. A node-level permutation gate (shuffle depth labels) is mis-calibrated because the pooled points are clustered within workflows and heteroscedastic in depth: on a *strict-uniform* flat null it fires at 9.3%, and a per-workflow standard-error gate over-rejects at 22.8% (both far above nominal). Replacing the gate with a *cluster bootstrap over whole workflows* brings the flat-null false-positive rate to 2.7% (realistic null) and 4.0% (strict-uniform null), both inside the ±5.3-point Monte-Carlo band around 5% at 150 trials (Figure 2). We report the mis-calibrated comparators *and* the fixed numbers, on *both* nulls, so nothing is cherry-picked.
- **Reduction.** A single-node workflow's contribution equals $\mathrm{Var}(X_{\mathrm{out}})$ and a deterministic sink contributes ≈ 0, as required.

5.2 Synthetic sweep (a recovery / identifiability result)

Recovered node-type variance budget. Across 250 random 10-node workflows (3000 runs each) in which we *inject* per-node variances with LLM nodes loudest, the estimator *recovers the injected*

shares: unpinned-sampling LLM nodes are recovered at a mean 71.7% of end-to-end output variance, LLM + parser/router at 99.3%, and deterministic nodes are near-silent (mean share 0.007); LLM nodes are the recovered top contributor in 73.6% of workflows (Figure 1). **We emphasize that this is parameter recovery, not a discovered empirical fact: we set the LLM injection variance largest, so "LLM nodes dominate" here is baked into the simulator.** The result that *is* non-trivial is identifiability—that the observational estimator recovers whatever injected budget is present, to median 1.7% error—which is what makes the budget trustworthy *when read on real logs*, where the shares are not set by us (the live experiment below is that test).

Depth-amplification law. Sweeping the true per-edge gain $\gamma \in \{1.0, 1.05, 1.10, 1.20, 1.35\}$ on homogeneous positive-gain chains (pools of 12 depth-8 chains, 60 trials each), the cluster bootstrap gate's detection rate rises with γ: it is 6.7% at γ=1 (the false-positive rate), 96.7% at γ=1.20, and 100.0% at γ=1.35. Where amplification is detected, the recovered coefficient is accurate: $\hat{g} = 0.187$ versus a true 0.182 at γ=1.20, with 94.8% confidence-interval coverage of the truth. The flat case is the headline guard: a 2.7% false-positive rate (realistic null; 4.0% strict-uniform) means the law does not fire on workflows that have none.

Intervention accounting (simulator only). *On the simulator*, pinning all LLM nodes (modeled as zeroing their injection, i.e. temperature→0 / cache) removes a mean 80.6% of end-to-end variance in ground truth; the decomposition *predicts* 80.5%, a median absolute error of 0.2 percentage points, and pinning LLM+router nodes removes 99.3%. This shows the additive budget is *internally* consistent with the intervention it implies. It is a synthetic identifiability check using the *same* simulator; we do *not* claim a validated real-pipeline pin/cache forecast, and we did not run a live pinning experiment.

5.3 Live mini-pipeline (the only non-baked evidence that LLM nodes dominate)

The synthetic budget bakes in LLM dominance; the live pipeline does not, so it carries the empirical weight of the "LLM nodes dominate" claim—and we report its limits plainly. We built a four-stage llama3.1:8b pipeline—extract a rating (N_1), transform to a polarity (N_2), format as JSON (N_3), validate/route (N_4)—and ran it 600 times over 30 fixed, general-domain inputs at temperature> 0 (complete run, not partial: false). **The runner logged scalar readouts for N_1, N_2, and N_4 only; the JSON-format node N_3 was not logged as a separate scalar**, so we estimate the decomposition on the *collapsed* graph $N_1 \to N_2 \to N_4$ (3 nodes, 3 depth levels), with N_3's formatting folded into the $N_2 \to N_4$ edge. On this input set the JSON-validity rate was 100.0%, so the router N_4 added *no* run-to-run variance and simply forwarded N_2's value; 14 inputs show non-trivial run-to-run variation. Estimating the decomposition on the real logs, the two LLM nodes carry 100.0% of the measured variance (N_1: 61.9%, N_2: 38.1%), with the deterministic router at 0.0%, and the measured interaction residual is −2.7% of total. This corroborates the node-*type* budget—LLM nodes carry essentially all the run-to-run variance, the router none—on a real model where the shares were *not* injected by us. We are deliberately conservative about what this does *not* show. (i) Because N_3 is folded into N_4 and the router is silent, the live graph has only 3 depth levels from a single chain—far too few, and not a population of workflows—so we **do not fit the depth law on live data**; the depth law rests entirely on the synthetic backbone. (ii) The router's 0% share is partly an artifact of a 100.0% validity rate on this easy input set; a harder input distribution that trips the router would shift the budget. (iii) One 8B model on 30 inputs is a corroboration, not a population study. What survives is narrow and honest: *on this real pipeline, the LLM nodes, not the deterministic router, carry the run-to-run variance*, which is the empirical analogue of the recovered synthetic budget.

6 Limitations

- **The exact decomposition is scoped to Assumption 1.** Exactness of Proposition 1 requires (A1) linear additive structure, (A2) independent per-node noise with no hidden common cause, (A3) a fixed DAG and fixed parents, and (A4) a finite scalar readout. Each can fail in a real pipeline: a non-linear node breaks (A1); a shared random seed, a common context window, or a shared retrieval corpus breaks (A2); a router that rewires the graph or a dynamically constructed subgraph breaks (A3). When they fail the additive split is an approximation, and the interaction residual

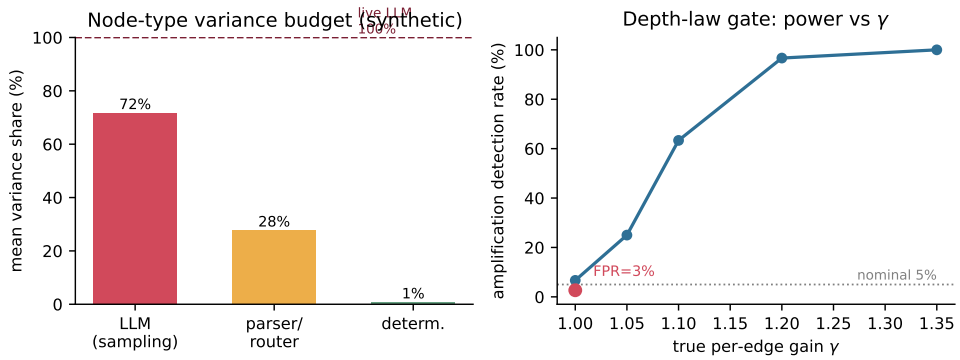

Figure 1: Recovered node-type variance budget and the depth-amplification law. Left: the *recovered* share of end-to-end output variance by node type across the synthetic sweep—LLM dominance here is recovery of an injected modeling choice (we set LLM injection variance largest), not a discovered fact; the dashed line marks the non-baked live-pipeline LLM share. Right: detection rate of the cluster-bootstrap depth gate versus the true per-edge gain γ; the point at γ=1 is the false-positive rate, held near nominal. All numbers from `results/analysis.json`.

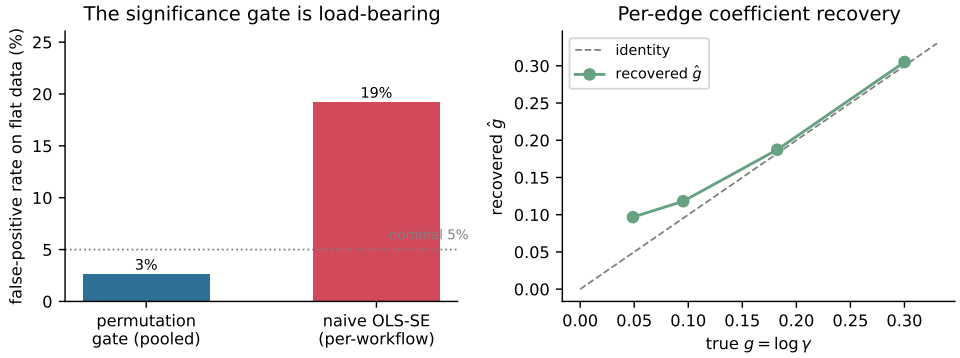

Figure 2: The significance gate is load-bearing and must respect clustering. On flat (no-amplification) workflows, the naive node-level slope test manufactures amplification far above nominal, while the cluster bootstrap over whole workflows holds the false-positive rate near 5% on both the realistic and the strict-uniform flat null. Right: on truly amplifying workflows the recovered per-edge coefficient $\hat{g}$ tracks the true $\log \gamma$ across the grid. From `results/analysis.json` and `results/validity_results.json`.

Δ (Eq. (3)) is exactly the quantity that exposes it—we report Δ as an accounting line rather than assuming conservation.

- **The synthetic budget is baked in.** The synthetic sweep *sets* LLM-node injection variance largest, so "LLM nodes carry the majority share" on the simulator is parameter recovery, not a discovered finding. The transferable claim is the *identifiability* result (the estimator recovers whatever budget is injected). The only non-baked evidence that LLM nodes actually dominate is the single live pipeline.
- **Depth law: positive-gain only, and empirical.** g is a fitted regularity with a CI and a gate, not a universal theorem. The monotone depth–contribution relation holds for a *homogeneous positive-gain chain*; with signed or heterogeneous path gains the coefficient can invert ($g < 0$ under attenuation) or cancel, so depth is *not* a universal amplifier. "Depth" is longest-path depth and the law assumes roughly comparable path lengths (we divide out multiplicity but not heterogeneous path-length mixtures).
- **Gate requires per-workflow blocks and a population.** The calibrated gate is a cluster bootstrap that resamples whole workflows; it needs the data *as* per-workflow blocks and a population of

realizations. A node-level permutation gate is mis-calibrated here because depth points cluster within workflows and are heteroscedastic in depth (we measure its inflated flat-null rate). A single small DAG does not yield a calibrated slope test.

- **Live experiment is thin and shallow.** It is one 8B model on 30 inputs; the JSON-format node is folded into the router readout, leaving 3 depth levels, so the live data corroborates the node-*type* budget but cannot test the depth law. The router's silence is partly an artifact of the easy input set's 100.0% validity rate.
- **Intervention accounting is simulator-only.** The pin/cache forecast is validated against the *same* simulator that generated the data; we make no claim of a validated real-pipeline causal pin/cache prediction and ran no live pinning experiment.
- **Readout map.** "Variance" of a text output is defined through an explicit readout ϕ (parsed value, branch indicator, or semantic-cluster code); a different ϕ yields a different decomposition. We use a literal scalar readout in the synthetic core and a parsed numeric readout on the live pipeline.
- **Scope.** We measure run-to-run *variance*, not correctness, failure, or task success; WID complements, and does not replace, failure-localization tools.

7 Conclusion

Run-to-run instability in multi-step LLM workflows is not a monolith. WID contributes a *method*—an exact per-node variance decomposition under stated assumptions, with the non-additivity it leaves reported as an explicit residual—and two things that make the method trustworthy: an *identifiability result* (the observational estimator recovers an injected per-node budget) and a *calibrated significance gate* for the depth coefficient (a cluster bootstrap over whole workflows whose flat-null false-positive rate stays near nominal where a node-level permutation test does not). We are explicit that the synthetic node-type budget is recovery of an injected modeling choice rather than a discovered fact; the empirical evidence that LLM nodes dominate is a single small live pipeline, which corroborates the node-type budget but is too shallow to test the depth law and on which intervention forecasting was not run. What we offer is therefore a decomposition *instrument* with honest calibration, not a strong empirical law—and we hope the per-node variance budget, read on real logs, becomes a routine diagnostic for agentic pipelines, the way a profiler is for code.

Reproducibility. All cores are stdlib Python with fixed, printed seeds; the validity suite, the analysis sweep, the macro generator, and the live-pipeline runner are released, and every number in this paper is regenerated from `results/*.json`.

References

[1] Harrison Chase et al. Langgraph: Building stateful multi-actor applications with llms. 2024. https://github.com/langchain-ai/langgraph.

[2] Bradley Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979. doi: 10.1214/aos/1176344552.

[3] Amine El Hattami, Nicolas Chapados, and Christopher Pal. Skill.nb: Selective formalization and gated execution for durable agent workflows. 2026. arXiv:2606.08049.

[4] Faisal Fareed. Cost-aware speculative execution for llm-agent workflows: An integrated five-dimension method. 2026. arXiv:2606.07846.

[5] Aaron R. Flouro and Shawn P. Chadwick. Hallucinations live in variance. 2026. arXiv:2601.07058.

[6] Raja Gond, Aditya K Kamath, Ramachandran Ramjee, and Ashish Panwar. Llm-42: Enabling determinism in llm inference with verified speculation. 2026. arXiv:2601.17768.

[7] Dongxin Guo, Jikun Wu, and Siu Ming Yiu. Agenteval: Dag-structured step-level evaluation for agentic workflows with error propagation tracking. 2026. arXiv:2604.23581.

[8] Aayush Gupta. Reliabilitybench: Evaluating llm agent reliability under production-like stress conditions. 2026. arXiv:2601.06112.

[9] Jennifer Haase, Jana Gonnermann-Müller, Paul H. P. Hanel, Nicolas Leins, and Thomas Kosch. Within-model vs between-prompt variability in large language models for creative tasks. 2026. arXiv:2601.21339.

[10] Aaditya Khanal, Yangyang Tao, and Junxiu Zhou. Beyond pass@1: A reliability science framework for long-horizon llm agents. 2026. arXiv:2603.29231.

[11] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. 2024. arXiv:2310.03714.

[12] Shizhe Lin and Ladan Tahvildari. Fase: Fast adaptive semantic entropy for code quality. 2026. arXiv:2606.09800.

[13] Xuyan Ma, Xiaofei Xie, Yawen Wang, Junjie Wang, and Boyu Wu. Demystifying the lifecycle of failures in platform-orchestrated agentic workflows. 2025. arXiv:2509.23735.

[14] Md Hafizur Rahman, Zafaryab Haider, Tanzim Mahfuz, and Prabuddha Chakraborty. Harp: Measuring harm amplification in multi-agent llm systems. 2026. arXiv:2605.27489.

[15] Angelika Romanou, Mark Ibrahim, Candace Ross, Chantal Shaib, and Kerem Oktar. Brittlebench: Quantifying llm robustness via prompt sensitivity. 2026. arXiv:2603.13285.

[16] Shayle R. Searle, George Casella, and Charles E. McCulloch. *Variance Components*. John Wiley & Sons, 2006.

[17] Yawen Wang, Wenjie Wu, Junjie Wang, and Qing Wang. From flat logs to causal graphs: Hierarchical failure attribution for llm-based multi-agent systems. 2026. arXiv:2602.23701.

[18] Yizhe Xie, Congcong Zhu, Xinyue Zhang, Tianqing Zhu, and Dayong Ye. From spark to fire: Modeling and mitigating error cascades in llm-based multi-agent collaboration. 2026. arXiv:2603.04474.

[19] Matei Zaharia, Omar Khattab, Lingjiao Chen, et al. The shift from models to compound ai systems. 2024. https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/.