HATCHRCA: SELF-INCUBATED SMALL MODELS FROM LLMS FOR ROOT CAUSE ANALYSIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Root cause analysis (RCA) of a microservice failure must report when the fault began, which component carried it and what caused it. A large language model (LLM) interprets heterogeneous telemetry well, but it compares numbers unreliably, and the evidence for one incident spans thousands of feature rows. Existing RCA systems therefore either rely on the LLM alone or add machine learning pipelines that an expert rebuilds for every new system, and whose rules are admitted without measuring their firing rate or precision. We present HATCHRCA, a framework in which the LLM builds the small models it later consults, designing task-specific models for each new system and writing and running their training programs. Because the LLM cannot reliably judge what it generates, HATCHRCA admits a generated model, rule or experience only when its performance measured on training cases meets the admission criteria, and each admitted model attaches its measured accuracy to the candidates it proposes. On the OPENRCA benchmark, with the same backbone, scoring program and 87 test tasks as the official RCA-AGENT, HATCHRCA raises the full-score rate from 29.9% to 52.9% and the mean score from 0.397 to 0.673, and the admission criteria rejected 4 of the 8 training programs.

1 INTRODUCTION

Microservice systems in banking, telecommunication and online retail degrade when a fault in one component propagates along service calls to every request on the affected path. Root cause analysis (RCA) must therefore report, quickly and precisely, the onset time, the component and the cause, and on-call engineers still spend much of an outage searching metrics, traces and logs by hand, which has motivated agents that automate RCA with large language models (LLMs) (Xu et al., 2025; Clark et al., 2026; Gong et al., 2026). The task is hard because it combines semantic interpretation, such as grouping a latency spike with an error burst, with numeric comparison over 1904 to 4368 feature rows per system to find the component with the earliest sustained anomaly. An LLM performs the first well but the second unreliably, so on the OPENRCA benchmark its official agent solves 11.34 percent of tasks in the original evaluation and 11.4 to 29.4 percent across datasets when re-run with a frontier backbone by Gopal & Krishnan (2026).

The first line of existing work keeps the whole analysis inside LLM agents. The official RCA-AGENT pairs a controller that plans each step with an executor that writes Python over the telemetry (Xu et al., 2025). This line of work changes how the LLM plans and gathers evidence, but the numeric comparison over thousands of rows, the step that failure analyses identify as the main source of error (Gopal & Krishnan, 2026; Riddell et al., 2026; Kim et al., 2026), still takes place inside the LLM.

The second line adds learned components around the LLM, which can perform the numeric comparison, but an expert selects their features and training pipeline by hand for every new system. Self-improving RCA agents remove part of this manual work by rewriting their knowledge or their analysis (Huang et al., 2026; Wu et al., 2025; Zan et al., 2026). What they generate is text, a workflow or a graph rather than a trained model, and a generated object is kept when a measured system-level score improves, so the measured quantity is a score over the whole agent rather than

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.

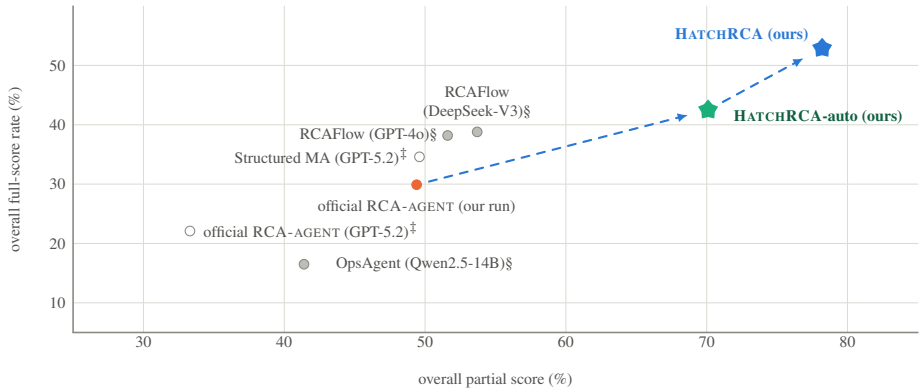


Figure 1: Overall results on OPENRCA. Each point is one system, with the overall partial score on the x axis and the overall full-score rate on the y axis. Stars mark our two configurations and the orange dot our run of the official RCA-AGENT, all with the same backbone on the 87-task split, and the dashed arrows link our three runs. Grey markers mark published systems on the full 335-case benchmark with other backbones. Filled grey markers use the at-least-one-element partial rate reported by their sources, open grey markers quote the printed Partial column of Gopal & Krishnan (2026), whose values match the mean partial-credit score rather than that rate, and OpsAgent uses its own 133-case test split. Published positions orient the reader rather than rank the systems.

the firing rate or the precision of one generated rule. No existing system therefore lets the LLM build the model that performs the numeric comparison, and none carries a measured accuracy of that model into diagnosis.

An LLM that writes code can take over that manual work by inspecting the data of a new system, designing models and writing the programs that train them, so the numeric comparison moves into small models that the LLM consults during diagnosis. HATCHRCA therefore lets measurements on training cases, not the judgment of the LLM, determine what the LLM may consult.

Figure 1 shows the outcome. Over all 87 test tasks, HATCHRCA raises the full-score rate from 29.9% to 52.9% and the partial rate, the fraction of tasks with at least one correct field, from 49.4% to 78.2%, both against the official RCA-AGENT with the same backbone, and it also exceeds every published system on the overall full-score rate. HATCHRCA-auto, which removes authored RCA knowledge, still improves on the official agent on both metrics. This paper makes three contributions.

- **Framework.** To our knowledge, HATCHRCA is the first root cause analysis framework in which an LLM generates its own task-specific small models (Figure 2). No person writes or edits the training program of any system, and one unchanged protocol onboards the four evaluation units of the three OPENRCA systems.

- **Evidence-based admission.** LLM-generated models, rules and experience may be wrong, and the LLM cannot reliably judge them. HATCHRCA therefore admits each one only if its performance measured on training cases meets the admission criteria, and attaches each admitted model's measured accuracy to its candidates. The LLM thus reasons only over evidence of measured reliability, which replaces the review a person would otherwise perform.

- **Performance.** Compared with the official RCA-AGENT under the same backbone, scoring program and task set, HATCHRCA raises the full-score rate on every evaluation unit. Over all 87 tasks, the full-score rate rises from 29.9% to 52.9% (+77%) and the mean score from 0.397 to 0.673 (+70%).

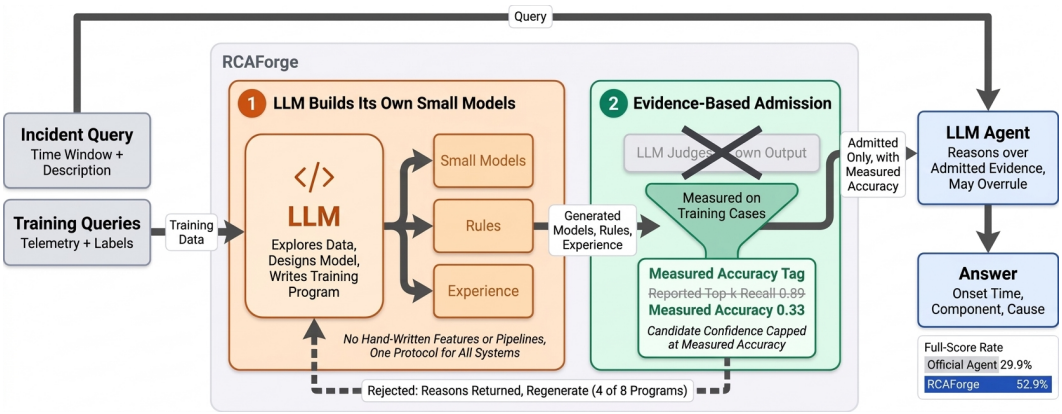


Figure 2: Overview of HATCHRCA. In Part 1 the LLM explores the data of each system, designs small models and writes their training programs, and mines rules and builds experience from training cases. Part 2 admits these generated objects only by measurements on training cases and returns the failure reasons of rejected programs for regeneration. During diagnosis, the LLM agent reasons over the admitted evidence and returns the onset time, the component and the cause. Each admitted model carries its measured accuracy, which caps the confidence of its candidates, as in the Telecom latency model, whose top-k recall of 0.889 exceeds its measured accuracy of 0.333. The inset gives the full-score rate against the official RCA-AGENT with the same backbone.

2 RELATED WORK

LLM agents for root cause analysis. Planning-based agents follow a controller and executor loop (Xu et al., 2025), standard operating procedures (Pei et al., 2025) or hierarchical workflows (Gao et al., 2026), and multi-agent designs test hypotheses recursively (Zhang et al., 2026a) or vote on candidates (Zhang et al., 2024). Other agents add multimodal tools (Zhang et al., 2025) or causal structure (Liang et al., 2026), or train the backbone itself for fault localization or incident management (Zhang et al., 2026b; Luo et al., 2025). In all of them the numeric comparison is performed either by the LLM or by a component that a person designed, and failure studies attribute most errors to reasoning over the evidence (Gopal & Krishnan, 2026; Riddell et al., 2026; Kim et al., 2026). HATCHRCA instead lets the LLM write the training program of the small model that performs this comparison.

Self-improving agents and generated artefacts. Self-improving RCA agents rewrite their knowledge, workflows and skills (Huang et al., 2026), improve alarm-based analysis (Wu et al., 2025), or evolve causal graphs (Zan et al., 2026). They keep a generated object when a measured system-level score improves, for two of them on the same incidents that drove the change (Wu et al., 2025; Zan et al., 2026) and for the third additionally on a held-out testbed (Huang et al., 2026). HATCHRCA generates trained models, executable rules and experience, and admits each only through measurements on training cases, with the measured accuracy of each model carried into diagnosis.

Benchmarks and evaluation. OPENRCA requires the onset time, the component and the cause jointly for every task (Xu et al., 2025), while its successor and other benchmarks evaluate the reasoning process (Fang et al., 2026a; Wang et al., 2026), live incidents (Clark et al., 2026; Gong et al., 2026) or coarser attribution (Pham et al., 2025). Because pooled scores can hide which method wins on a subsystem (Hu et al., 2026) and benchmark accuracy can overstate readiness (Fang et al., 2026b), we report every result per dataset and against the official agent with the same backbone.

Table 1: The three kinds of generated object in HATCHRCA. The LLM produces every object, and the host admits it only through measurements on training cases.

Object	Produced by	Admitted when
Small models	a training program that the LLM writes and the host runs in a sandbox	each model reaches a bar measured on a reference model set, overfits within a bound, uses no test query and reloads from its saved feature function, Eq. (2)
Rules	the LLM, from clusters of failed training cases	they fire on enough training cases, are precise or strictly improving where they fire, never lower the training score, and pass again on held-out training cases, Eq. (4)
Experience	runs of HATCHRCA on training queries	it comes from a training query and its measured correctness passes the retrieval filter

3 METHOD

HATCHRCA rests on two design decisions, one per methodological contribution. The first moves numeric comparison out of the LLM and into small models that the LLM itself generates, so no person designs a model for a new system (Section 3.2). The second separates generation from evaluation, so that everything the LLM generates reaches diagnosis only through measurements computed on training cases (Section 3.3). Section 3.1 fixes the notation, and Section 3.4 describes how diagnosis consumes the admitted objects.

3.1 PROBLEM SETTING

A system comes with a set of training queries $\mathcal{Q}_{\mathrm{tr}}$ with labels and a disjoint set of test queries $\mathcal{Q}_{\mathrm{te}}$. A query q consists of a time window and a natural-language description of an incident, and its answer is a set of faults $y_q = \{(t_i, c_i, r_i)\}_i$, where t_i is the onset time, c_i the component and r_i the cause. Every cause belongs to one fault type in $\mathcal{F} = \{\text{latency}, \text{packet loss}, \text{resource}\}$. The evidence of a query is summarized in two feature tables, a fault-type table X^{ft} with one row per query and a component table X^{c} with one row per pair of query and candidate component $c \in \mathcal{C}_q$. The benchmark scores a prediction $\hat{y}_q$ with a function $S(\hat{y}_q, y_q) \in [0, 1]$ that equals one only when every field of every fault is correct. The goal is a high mean of S over $\mathcal{Q}_{\mathrm{te}}$, obtained without per-system design by a person and without any use of $\mathcal{Q}_{\mathrm{te}}$ before diagnosis.

We call a model, a rule or an experience entry produced by the LLM a *generated object*. The host is the fixed program that runs the LLM, executes its code and computes every admission measurement. Table 1 summarizes the three kinds of generated object, how each is produced and how each is admitted.

3.2 LLM-GENERATED SMALL MODELS

Numeric comparison over thousands of feature rows is the step at which the LLM is least reliable, while writing code is a step at which it is reliable. HATCHRCA therefore asks the LLM to write the program that performs this comparison, instead of asking it to perform the comparison for every task. For each system, the LLM generates a model set

$$\mathcal{M} = \big(\phi,\ g,\ \{h_f\}_{f \in \mathcal{F}},\ m_{\mathrm{cause}},\ m_{\mathrm{time}}\big), \tag{1}$$

where ϕ is a feature function that maps the columns of X^{ft} and X^{c} to model inputs, g predicts the fault type of a query, h_f scores every candidate component for fault type f so that the candidates can be ranked, m_{cause} predicts the cause and m_{time} predicts the onset time. The LLM generates $\mathcal{M}$ in three steps, which Algorithm 1 summarizes.

Exploration. The first LLM call writes read-only analysis code over the training rows of both tables. The host runs this code and returns its printed summary, which reports the class balance, the number of queries per fault type, sparse and duplicated columns and possible leakage. This summary, not an inspection by the authors, is the context of the next step. The model design can therefore follow the data, whose size ranges from 34 to 102 training queries across systems.

Algorithm 1 Onboarding of one system

Require: training tables $X^{\mathrm{ft}}, X^{\mathrm{c}}$ restricted to $\mathcal{Q}_{\mathrm{tr}}$, reference model set $\mathcal{M}^{\mathrm{ref}}$, round limit K
$b \leftarrow$ host re-scores $\mathcal{M}^{\mathrm{ref}}$ on the query-level five-fold split ▷ measured bars
$\sigma \leftarrow$ run the LLM-written exploration code ▷ data summary
$R \leftarrow \emptyset$
for $k = 1, \ldots, K$ **do**
  $P \leftarrow$ LLM writes a training program from σ, the contract and the failure reasons R
  $(\mathcal{M}, \hat{\mu}, \mathcal{V}) \leftarrow$ host runs P in the sandbox
  $R \leftarrow$ host evaluates the admission predicate of Eq. (2) with bars b
  if $R = \emptyset$ **then**
    $a_f \leftarrow$ host measures the accuracy of h_f for every $f \in \mathcal{F}$ by Eq. (3)
    return admitted $\mathcal{M}$ with $\{a_f\}$
  end if
end for
return no admitted model set

Program generation. The second LLM call writes a complete training program under a fixed contract. The contract prohibits four things, namely a reference implementation, a feature list, hyperparameter values and any access to $\mathcal{Q}_{\mathrm{te}}$. It requires the program to save every element of $\mathcal{M}$ under fixed file names and to report its training and validation metrics $\hat{\mu}$, together with its validation query identifiers $\mathcal{V}$, through one structured call. Everything else is left to the LLM, including the learner, the class weighting, the validation scheme and the construction of features inside ϕ. The same contract is used unchanged for every system.

Sandboxed execution and regeneration. The host executes the program in a sandbox that exposes only a saving function and a reporting function. The saving function accepts only the fixed file names, the sandbox blocks the files that define the test split, and saved models are later loaded with a restricted loader that cannot execute code. The host then applies the admission criteria of Section 3.3. If they fail, the host returns the list of failure reasons to the LLM, which writes a new program. This loop runs for at most $K = 6$ rounds per system, and no person edits a program at any point.

3.3 Evidence-based admission

Once the LLM writes the programs, rules and experience that diagnosis consumes, their correctness can no longer be assumed. Asking the same LLM to assess its own output does not resolve this problem, because the assessment inherits the errors of the generation. HatchRCA therefore separates the two roles. The LLM generates, the host measures on training cases, and a generated object reaches diagnosis only when the measurements satisfy explicit criteria. The same principle applies to all three kinds of generated object, and the criteria differ only in what is measured.

Admitting a model set. Let $\mathrm{acc}(g)$ be the validation accuracy of the fault-type model, $\mathrm{rec}_k(h_f)$ the validation top-k recall of the component model for fault type f, and $\Delta(m)$ the gap between training and validation accuracy of any model m in $\mathcal{M}$. A model set is admitted if and only if

$$\begin{aligned} &\mathrm{acc}(g) \geq b_g - \delta_g, \qquad \mathrm{rec}_k(h_f) \geq b_f - \delta_c \;\; \forall f \in \mathcal{F}, \qquad \Delta(m) \leq \gamma \;\; \forall m \in \mathcal{M}, \\ &\mathcal{V} \cap \mathcal{Q}_{\mathrm{te}} = \emptyset, \qquad \mathrm{Load}(\mathcal{M}) = 1, \qquad \phi(X) = X_{\mathcal{M}}. \end{aligned} \tag{2}$$

Here b_g and b_f are the fault-type accuracy and the top-k recall of the reference model set $\mathcal{M}^{\mathrm{ref}}$ (Appendix C), re-scored by the host on the same query-level five-fold split, and $\delta_g = 0.02$, $\delta_c = 0.05$ and $\gamma = 0.30$ are margins fixed once for all systems. The condition $\mathrm{Load}(\mathcal{M}) = 1$ requires that every model loads, uses only existing columns and does not score zero recall on every fault type. The last condition requires that ϕ, reloaded by the host from its saved source, reproduces exactly the input columns $X_{\mathcal{M}}$ on which the models were fitted, so the models can be applied at diagnosis without the program. Because b_g and b_f are measured on the system itself, the bars adapt to each system without a hand-set threshold. A reported metric that names the test set also fails the predicate. The cause model and the time model are admitted separately, the cause model when its validation accuracy reaches 0.40 and the time model when the program does not mark it as low confidence. A

model that is not admitted is never shown to the LLM agent, and the corresponding field falls back to the reasoning of the agent.

Measured accuracy. Admission alone is binary, while the LLM agent needs a measure of how far each candidate can be trusted. The confidence that a model assigns to its own candidate does not answer this question, and the top-k recall that the program reports answers a different one. The host therefore measures, for every admitted component model, how often its top-ranked candidate is correct. For fault type f, let h_f^{hit} be the rank-1 hit rate of h_f when the host refits a copy of the model on four of five query-level folds of $\mathcal{Q}_{\mathrm{tr}}$ and ranks the candidates of the fifth. The measured accuracy a_f and the confidence $\tilde{s}$ shown to the LLM agent are

$$a_f = \min\big(\max(h_f^{\mathrm{hit}},\, \ell),\, u\big), \qquad \tilde{s} = \min(s,\, a_f), \tag{3}$$

where s is the confidence of h_f for a candidate, and $\ell = 0.30$ and $u = 0.95$ keep a_f away from the extremes. A candidate therefore never appears more reliable than its model was measured to be. The capped confidence also selects one of three instruction tiers in the prompt of the LLM agent, which range from verifying the top candidate first to relying mainly on the analysis of the telemetry, so a model with a low a_f cannot present its candidates as high-confidence evidence. If h_f^{hit} and the rate that the program reported differ by more than 0.15, which indicates an inconsistent environment, the host discards a_f and leaves the candidates unchanged.

Admitting rules. A rule is a pair $\rho = (\pi, \kappa)$ of an executable predicate π over the evidence available at diagnosis and a correction κ that overrides some fields of the current answer. After each training cycle, the host clusters the failed training cases by the pair of predicted and true fault types, and the LLM writes one rule per cluster. The prompt shows representative cases with their wrong fields but masks the exact ground-truth component, cause and time, so a rule cannot memorize answers. Let $F_\rho \subseteq \mathcal{Q}_{\mathrm{tr}}$ be the training cases on which ρ fires and changes the answer. On each fired case the host scores the answer before and after the correction and counts the cases improved, harmed and unchanged, n^+, n^- and n^0. A rule is admitted if

$$|F_\rho| \geq n_{\min}, \qquad \sum_{q \in \mathcal{Q}_{\mathrm{tr}}} S(\hat{y}_q^\rho, y_q) \geq \sum_{q \in \mathcal{Q}_{\mathrm{tr}}} S(\hat{y}_q, y_q), \qquad \frac{n^{\mathrm{full}}}{|F_\rho|} \geq \pi_{\min} \ \text{ or } \ \big(n^- = 0 \wedge n^+ \geq 1\big), \tag{4}$$

where $\hat{y}_q^\rho$ is the answer after the rule, n^{full} is the number of fired cases that become fully correct, $n_{\min} = 2$ and $\pi_{\min} = 0.8$. The last condition gives two admission channels. A precise rule must be fully correct on most cases where it fires, and an incremental rule must improve at least one case while harming none. Because these counts are computed on the same cases from which the rule was mined, the host repeats the check on a held-out slice of the training cases under an 80/20 split. A rule that does not fire on the held-out slice is admitted only if it improves at least 3 training cases without harming any. A rejected rule returns its failure reasons, including its counts n^+, n^- and n^0, to the LLM for a bounded number of revisions.

Admitting experience. An experience entry records one training query with the answer of HATCHRCA, the ground truth and the correctness of each field, so its admission also rests on a measurement. Entries are created only from training queries, test query identifiers are excluded at retrieval, and at diagnosis an entry whose component and cause were both wrong is excluded, except for the two minority fault types, where such entries are shown as negative examples.

3.4 DIAGNOSIS WITH ADMITTED EVIDENCE

For a test query, the admitted fault-type model and component models produce the three highest-ranked candidates per fault type, each with its capped confidence $\tilde{s}$ from Eq. (3). The LLM agent, which uses the controller and executor of the official RCA-AGENT, receives these candidates together with the most relevant experience entries and the telemetry access of the official agent. The agent checks the candidates against the telemetry and may overrule any of them, and the admitted rules then correct the fields on which they fire. The division of labor is therefore explicit. The small models perform the numeric ranking, the admission criteria and the measured accuracy determine how much of this ranking reaches the agent, and the LLM contributes the semantic reasoning over the telemetry and the incident description.

Table 2: Full-score rate (%) and mean score against the official RCA-AGENT with the same backbone and scoring program. HATCHRCA-auto keeps only generated and admitted components. Bold marks the best value per row.

	Official RCA-AGENT		HATCHRCA		HATCHRCA-auto	
Dataset	Rate	Mean	Rate	Mean	Rate	Mean
Telecom	29.4	0.441	**52.9**	**0.696**	35.3	0.510
Bank	32.4	0.399	**61.8**	**0.709**	52.9	0.602
cloudbed-1	17.6	0.265	29.4	**0.525**	**35.3**	0.436
cloudbed-2	36.8	0.469	**57.9**	**0.719**	36.8	0.548
All	29.9	0.397	**52.9**	**0.673**	42.5	0.540

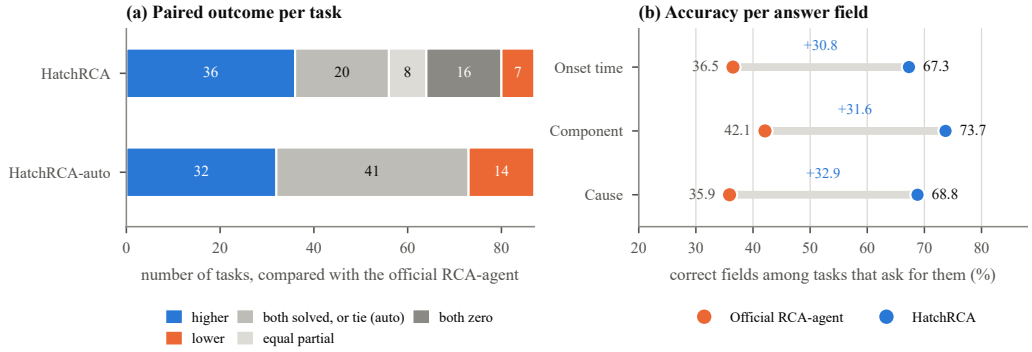


Figure 3: Per-task and per-field comparison with the official RCA-AGENT on the same tasks. (a) Paired outcome of every task as higher, equal or lower, with the equal tasks of HATCHRCA split by tie type. (b) Accuracy per answer field. Exact values are in Appendix F.

4 EXPERIMENTS

The experiments evaluate the three contributions. Section 4.2 measures the performance gain, Section 4.3 examines the small models that the LLM generated, and Section 4.4 reports what the admission criteria accepted and rejected.

4.1 SETUP

OPENRCA gives four evaluation units over the 87 test tasks, and both configurations use the same backbone and the same public scoring program as the official RCA-AGENT. Appendix B states the unit definitions, the scoring rule, the re-run that completed baseline coverage, and the authored material each configuration keeps.

4.2 MAIN RESULTS

Table 2 gives the main results, the full-score rate and the mean score of every system on every evaluation unit. HATCHRCA solves 46 of the 87 tasks against 26 for the official RCA-AGENT, which raises the full-score rate from 29.9% to 52.9% and the mean score from 0.397 to 0.673. The gain appears on every evaluation unit and on both metrics. Per unit, HATCHRCA solves between 1.6 and 1.9 times as many tasks as the official agent, and the absolute gain is largest on Bank, where the full-score rate rises from 32.4% to 61.8%. cloudbed-1 has the lowest mean score of the four units for all three systems. The counts behind each rate, in tasks solved at full credit out of the unit's test tasks, are 11, 21 and 18 of 34 on Bank, 5, 9 and 6 of 17 on Telecom, 3, 5 and 6 of 17 on cloudbed-1, and 7, 11 and 7 of 19 on cloudbed-2.

Because all systems answer the same 87 tasks, a paired comparison per task is possible, as Figure 3(a) shows. HATCHRCA scores higher than the official RCA-AGENT on 36 tasks, lower on 7

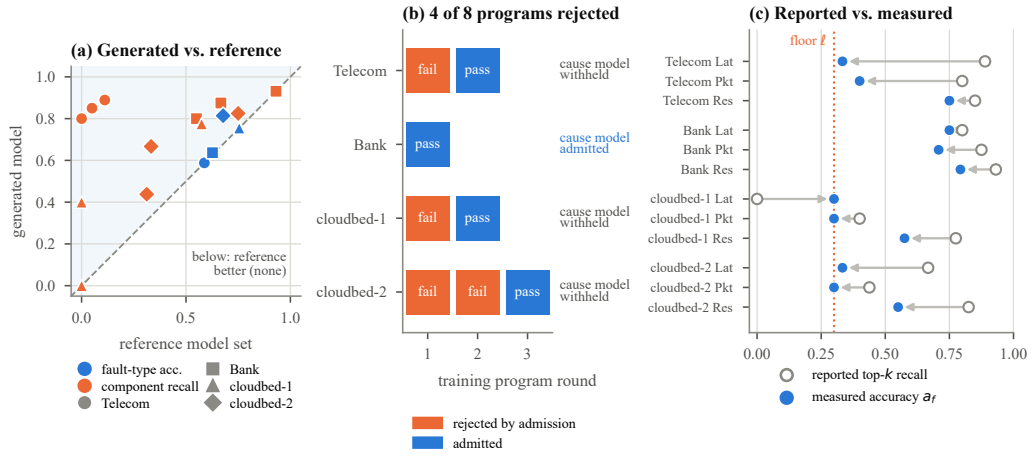


Figure 4: Generated small models and their admission. (a) Validation quality of the 16 generated models against the reference set. (b) Training programs per dataset until admission. (c) Reported top-k recall against the measured accuracy a_f of Eq. (3).

and equal on 44. An exact sign test on the tasks that differ gives $p = 9.0 \times 10^{-6}$. A bootstrap over tasks gives a 95 percent interval of 0.171 to 0.382 for the gain in mean score and of 11.4 to 34.5 percentage points for the gain in full-score rate, and both intervals exclude zero.

HATCHRCA-auto solves 37 tasks, 11 more than the official RCA-AGENT and 9 fewer than the main configuration. The authored material therefore accounts for 9 tasks, and the generated and admitted components alone still improve on the official agent. The paired comparison of HATCHRCA-auto with the official RCA-AGENT gives 32 tasks higher, 14 lower and 41 equal, with a sign-test $p = 1.1 \times 10^{-2}$ and a bootstrap interval of 0.032 to 0.254 for the gain in mean score. Appendix D compares these rates with published results on OPENRCA, which use other backbones and are therefore not directly comparable.

Figure 3(b) splits the result by answer field, and HATCHRCA improves each of the three fields by at least 30.8 percentage points. The largest gain is in cause identification and the smallest in onset time.

4.3 GENERATED SMALL MODELS

The generated training programs adapted to the data of each system. On Bank, the largest dataset, the program trained one class-balanced gradient-boosted model (Chen & Guestrin, 2016) per fault type and validated it with query-level five-fold cross-validation. On Telecom, the smallest dataset, it used leave-one-out validation. On cloudbed-2 the program trained a regularized logistic model for the fault type, and on cloudbed-1 it added conservative fallbacks for rare classes.

Figure 4(a) compares the generated models with the frozen reference model set, which was built by hand, and Table 6 in Appendix F lists the values. Across the 16 comparisons, 16 generated models reach their admission bar and 12 exceed the reference itself, with none falling below it. The largest differences appear on Telecom, where the generated latency model reaches a top-k recall of 0.889 against 0.111 for the reference. Component recall remains low where positive examples are scarce. On cloudbed-1 the latency model reaches a recall of zero with only 4 positive rows, where the majority class covers 97.6% of the rows, and the reference also scores zero on this cell. Appendix E gives the per-class statistics, which show that row-level accuracy stays close to the majority share on every cell and is therefore not informative for these models.

4.4 EVIDENCE-BASED ADMISSION

The second contribution requires that the admission criteria reject generated objects in practice and that the measured accuracy differ from what the generated program reports. Both conditions hold in our runs.

Models. The LLM generated 8 training programs across the four units, and the admission criteria rejected 4 of them, as Figure 4(b) shows. The rejections recorded 6 reasons, namely 4 top-k recall shortfalls, 1 fault-type accuracy shortfall and 1 program that did not run, and every rejected program was regenerated within the limit of 6 rounds. Two first rounds show how the checks act. On Telecom the first program reached a fault-type accuracy of 0.529 against a bar of 0.568 and was rejected. On cloudbed-2 the first program reached a fault-type accuracy of 0.797, above its bar of 0.658, but its latency model reached a top-k recall of 0.000 against a bar of 0.283. A program with a good aggregate score was therefore still rejected because of one weak component model. The criteria also withheld models inside admitted model sets. The cause model passed its floor only on Bank, and the time model passed on none of the four units, so for these fields the answer comes from the reasoning of the LLM agent instead of a model that was measured to be weak.

Measured accuracy. Figure 4(c) shows that the measured rank-1 accuracy is often far below the top-k recall that the programs report, and Table 7 in Appendix F lists both. Without the cap of Equation 3, candidates of a model with such a gap would enter the context of the LLM agent with a confidence close to their reported recall. On 3 of the 12 cells the measured rate lies at or below ℓ, so the cap there equals the floor rather than the measurement. The admission checks rely on metrics that the program reports, so we re-executed every stored program in a separate harness, which recomputed all 36 reported metrics from the saved models with every difference below 10^{-14}.

Clauses and experience. In HATCHRCA-auto the LLM mined clauses for the prompt checklist from failed training cases, and the clause library carries 12 admitted clauses, 9 ranking components and 3 naming a fault type. A component clause is admitted when it beats the majority-component baseline on the training cases and again on held-out cases, and a fault-type clause when its hit rate and false-fire rate clear fixed bars and the held-out cases confirm it. A second-seed re-verification exists for the 3 fault-type clauses only, so the 9 component clauses rest on one measurement. Experience entries are built from training queries only, and test query identifiers are excluded when entries are retrieved.

Link to the gain. Admission and the measured accuracy therefore bound what the LLM agent sees, but no experiment disables admission alone, so their shares in the gain are not isolated (Appendix G).

5 CONCLUSION

HATCHRCA moves the numeric comparison of root cause analysis into small models that the LLM generates and that the admission criteria accept, and each admitted model attaches its measured accuracy to its candidates. On OPENRCA, with the same backbone and scoring program as the official RCA-AGENT, it raises the full-score rate from 29.9% to 52.9% and improves every evaluation unit. Appendix G states the limitations of this study, and two directions follow from them, an ablation that switches admission off within the same pipeline and repeated onboarding runs that measure how much the generated models vary.

REPRODUCIBILITY STATEMENT

Every reported number is recomputed from stored per-task predictions with the unmodified public scoring program of OPENRCA, and every statistic in the text is a macro in a generated numbers file. The anonymized supplement contains the admission logs, the per-task predictions and the aggregation scripts that regenerate the tables.

USE OF LLMS

A large language model generated the training programs, rules and experience that this paper studies, and it was used to draft parts of the text. The authors verified all claims, figures and numbers.

REFERENCES

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016. doi: 10.1145/2939672.2939785.

Jackson Clark, Yiming Su, Saad Mohammad Rafid Pial, Yifang Tian, et al. SREGym: A live benchmark for AI SRE agents with high-fidelity failure scenarios, 2026. arXiv:2605.07161.

Aoyang Fang, Yifan Yang, Jin'ao Shang, Qisheng Lu, Junjielong Xu, Rui Wang, Songhan Zhang, Yuzhong Zhang, Boxi Yu, and Pinjia He. OpenRCA 2.0: From outcome labels to causal process supervision, 2026a. arXiv:2606.27154.

Aoyang Fang, Songhan Zhang, Yifan Yang, Haotong Wu, Junjielong Xu, Xuyang Wang, Rui Wang, Manyi Wang, Qisheng Lu, and Pinjia He. Rethinking the evaluation of microservice RCA with a fault propagation-aware benchmark. *Proceedings of the ACM on Software Engineering*, 3:1605–1627, 2026b. doi: 10.1145/3797100.

Yufei Gao, Zhengong Cai, and Bowei Yang. RCAFlow: A workflow-informed hierarchical planning multi-agent system for root cause analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 300–308, 2026. doi: 10.1609/aaai.v40i1.36991.

Albert Gong, Kyuseong Choi, Abhineet Agarwal, Jason Schechner, et al. ORCA-bench: How ready are language model agents for oncall?, 2026. arXiv:2607.28545.

Athira Gopal and Ashwanth Krishnan. How far can root cause analysis go on real-world telemetry data?, 2026. arXiv:2607.13548.

Lining Hu, Ting Liu, and Yuzhuo Fu. Pooled leaderboards hide system-specific winners: A reporting-protocol audit of offline root-cause analysis benchmarks, 2026. arXiv:2606.29159.

Haiyu Huang, Jiewei Lyu, Zhihan Jiang, Jinyang Liu, Xiao He, et al. From general agents to RCA experts: A self-evolving harness for root cause analysis, 2026. arXiv:2608.25661.

Taeyoon Kim, Woohyeok Park, Hoyeong Yun, and Kyungyong Lee. Why do AI agents systematically fail at cloud root cause analysis?, 2026. arXiv:2602.09937.

Shuai Liang, Pengfei Chen, Bozhe Tian, Gou Tan, et al. MetaRCA: A generalizable root cause analysis framework for cloud-native systems powered by meta causal knowledge. *Proceedings of the ACM on Software Engineering*, 3(FSE):138–159, 2026. doi: 10.1145/3797069.

Yu Luo, Jiamin Jiang, Jingfei Feng, Lei Tao, Qingliang Zhang, et al. OpsAgent: An evolving multi-agent system for incident management in microservices, 2025. arXiv:2510.24145.

Changhua Pei, Zexin Wang, Fengrui Liu, Zeyan Li, Yang Liu, et al. Flow-of-Action: SOP enhanced LLM-based multi-agent system for root cause analysis. In *Companion Proceedings of the ACM on Web Conference 2025*, pp. 422–431, 2025. doi: 10.1145/3701716.3715225.

Luan Pham, Hongyu Zhang, Huong Ha, Flora Salim, and Xiuzhen Zhang. RCAEval: A benchmark for root cause analysis of microservice systems with telemetry data. In *Companion Proceedings of the ACM on Web Conference 2025*, pp. 777–780, 2025. doi: 10.1145/3701716.3715290.

Evelien Riddell, James Riddell, Gengyi Sun, Michał Antkiewicz, and Krzysztof Czarnecki. Stalled, biased, and confused: Uncovering reasoning failures in LLMs for cloud-based root cause analysis. In *Proceedings of the 2026 IEEE/ACM Third International Conference on AI Foundation Models and Software Engineering (FORGE '26)*, pp. 172–183, 2026. doi: 10.1145/3793655.3793732.

Yilun Wang, Guangba Yu, Haiyu Huang, Yujie Huang, et al. Cloud-OpsBench: A reproducible benchmark for agentic root cause analysis in cloud systems, 2026. arXiv:2603.00468.

Keyu Wu, Qianjin Yu, Manlin Mei, Ruiting Liu, et al. TN-AutoRCA: Benchmark construction and agentic framework for self-improving alarm-based root cause analysis in telecommunication networks, 2025. arXiv:2507.18190.

Junjielong Xu, Qinan Zhang, Zhiqing Zhong, Shilin He, Chaoyun Zhang, Qingwei Lin, Dan Pei, Pinjia He, Dongmei Zhang, and Qi Zhang. OpenRCA: Can large language models locate the root cause of software failures? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=M4qNIzQYpd.

Lei Zan, Keli Zhang, Shifeng Xie, Jiale Zheng, et al. EvoCause: LLM-guided evolution of causal graphs for root cause analysis, 2026. arXiv:2607.27290.

Lingzhe Zhang, Tong Jia, Kangjin Wang, Chiming Duan, Minghua He, et al. Toward in-depth root cause localization for microservices with multi-agent recursion-of-thought. *IEEE Transactions on Dependable and Secure Computing*, 23(5):10702–10719, 2026a. doi: 10.1109/TDSC.2026.3706572.

Lingzhe Zhang, Yunpeng Zhai, Tong Jia, Chiming Duan, et al. ThinkFL: Self-refining failure localization for microservice systems via reinforcement fine-tuning. *ACM Transactions on Software Engineering and Methodology*, 35(9):1–44, 2026b. doi: 10.1145/3789262.

Wei Zhang, Hongcheng Guo, Jian Yang, Zhoujin Tian, Yi Zhang, et al. mABC: Multi-agent blockchain-inspired collaboration for root cause analysis in micro-services architecture. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 4017–4033, 2024. doi: 10.18653/v1/2024.findings-emnlp.232.

Xiao Zhang, Qi Wang, Mingyi Li, Yuan Yuan, Mengbai Xiao, et al. TAMO: Fine-grained root cause analysis via tool-assisted LLM agent with multi-modality observation data in cloud-native systems. *IEEE Transactions on Services Computing*, 18(6):4221–4233, 2025. doi: 10.1109/TSC.2025.3629066.

A DATASETS

Table 3 lists the test tasks, training queries, test faults and distinct answer components of each evaluation unit. The units differ by a factor of three in training queries, from 34 on Telecom to 102 on Bank, which is the range over which one unchanged onboarding protocol must work. cloudbed-1 is the unit with the lowest mean score for every system in Table 2.

Table 3: Datasets and task counts. Test tasks are the benchmark split, and training queries are the only data available to onboarding.

Dataset	Test tasks	Training queries	Test faults	Components in answers
Telecom	17	34	17	6
Bank	34	102	39	10
cloudbed-1	17	53	24	9
cloudbed-2	19	59	29	10
Total	87	n/a	109	n/a

B EXPERIMENTAL SETUP

We evaluate on the three systems of OPENRCA (Xu et al., 2025), a telecommunication system, a banking system and an online market whose two clusters are reported separately because their fault distributions differ, giving four evaluation units in all, as Table 3 in Appendix A lists. A task counts as solved only when the onset time lies within one minute of the ground truth and the component and the cause are correct for every fault in the task. We report the full-score rate, which is the fraction of solved tasks, and the mean score of the public scoring program, which also gives partial credit. Both systems use the same backbone, deepseek-v4-flash. The official RCA-AGENT runs unmodified except for the API endpoint. Its first run left 20 of the 87 tasks without an answer, and our re-runs answered all of them, so both systems are scored on the same tasks by the same program. Each task is scored once, with no selection across runs.

The main configuration of HATCHRCA keeps authored prompt text from development and 9 rules ported by hand, in addition to what the LLM generated. Among that text are a per-dataset decision sheet, a per-dataset signature checklist, on Telecom a fault-signature scan of derived medians and ratios, and on the output side a hand-written vocabulary of allowed causes, falling back to the first entry where that vocabulary holds one. We therefore also report an autonomous configuration, HATCHRCA-auto, which removes authored RCA knowledge from the prompts and the outputs, so that its rule scan no longer carries the rules ported by hand and its prompt checklist carries only the mined clauses that passed admission. HATCHRCA-auto closes the 17 sources of authored RCA judgment the switch enumerates, on the prompt side and on the output side, and the control arm, where such material is found in 31 of its rows, shows that the check detects it when it is present. The 4 surfaces outside that claim are feature tables the dataset provides, benchmark answer vocabulary and the task/data interface text, tool-interface safety text written for the executor and human-distilled hint cards used as an input to the rule miner. Both configurations use the same generated small models.

C ADMISSION CONSTANTS

Table 4 lists every constant that the admission criteria apply and states whether it is measured or configured. The frozen reference model set is an earlier model set for each system, built by hand before this work and never changed afterwards. The host re-scores it on the same query-level five-fold split that it uses for generated models, so a generated program must reach a quality that another model set already reached on the same folds. The feature tables are produced from raw telemetry by one extraction step shared by all systems and are not generated by the LLM, while the feature function that maps table columns to model inputs is written by the LLM for each system.

Table 4: Admission constants as implemented. Measured values are computed by the host from training data at onboarding, and configured values are fixed by the authors for all systems.

Constant	Value	Role	Origin
Fault-type bar	reference − 0.02	minimum fault-type accuracy	measured
Component bar	reference − 0.05	minimum top-k recall per fault type	measured
Overfit bound	0.30	maximum gap of training over validation accuracy	configured
Cause floor	0.40	minimum accuracy for using the cause model	configured
Round limit	6	training programs per system	configured
Measured accuracy a_f	per fault type	cap on candidate confidence, Equation 3	measured
Cap bounds ℓ, u	0.30, 0.95	range of a_f	configured
Deviation guard	0.15	discards a_f when it exceeds the reported metric by more than this	configured
Rule support	2	minimum number of fired training cases	configured
Rule precision $\pi_{\min}$	0.8	minimum full correctness on fired cases, precision channel	configured
No-fire alternative	3	improved training cases required when a rule never fires on the held-out slice	configured
Held-out split	80/20	re-verification of an admitted rule	configured

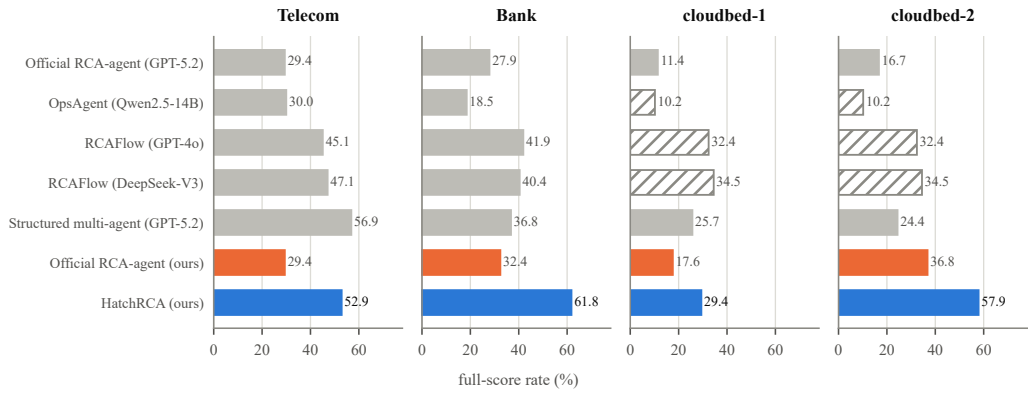


Figure 5: Full-score rates on OpenRCA, one panel per dataset. Grey bars are published rates quoted from their sources, hatched bars are published rates that merge the two Market clusters and therefore appear in both Market panels, and colored bars are our same-backbone runs of the official RCA-agent (orange) and HatchRCA (blue).

D Published results on OpenRCA

Figure 5 places HatchRCA next to the published full-score rates on OpenRCA. The published systems use other backbones, from Qwen2.5-14B to GPT-5.2, and three of them report one merged rate for the two Market clusters, so the comparison orients the reader rather than ranks the systems. HatchRCA exceeds every published rate on Bank, where it reaches 61.8% against at most 41.9%, and every published per-cluster rate on the two Market clusters. On Telecom it falls below only the structured multi-agent system, which uses GPT-5.2 together with hand-written domain knowledge and reaches 56.9% against 52.9% for HatchRCA. Our own run of the official RCA-agent with deepseek-v4-flash stays close to the third-party GPT-5.2 re-measurement on Telecom and Bank, which indicates that the gain of HatchRCA does not come from a weak baseline run.

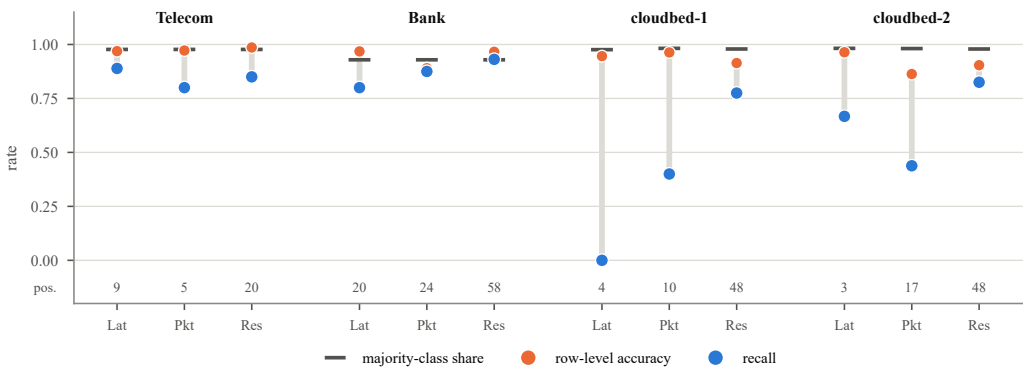


Figure 6: Per-class statistics of the 12 generated component models on training queries. For each fault type (Lat, Pkt, Res) the bar marks the majority-class share, the orange dot the row-level accuracy and the blue dot the recall. The numbers above the axis give the positive rows, which are root-cause candidates.

E PER-CLASS STATISTICS OF THE COMPONENT MODELS

Figure 6 shows, for every component model, the share of the majority class, the row-level accuracy and the recall on training queries. The majority class, which consists of candidates that are not the root cause, covers at least 92.9% of the rows in every cell, and the row-level accuracy stays close to this share everywhere. Row-level accuracy therefore says little about these models, while recall separates them clearly, from 0.931 for Bank resource faults to zero for cloudbed-1 latency faults, which have only 4 positive rows. This contrast is the reason why the admission criteria use top-k recall and why the LLM agent receives the measured accuracy of Eq. (3) rather than the accuracy that a program reports.

F DETAILED RESULTS

This appendix lists the exact values behind Figures 3 and 4, one table per panel group.

Table 5 gives the counts behind Figure 3(b). A task asks for one, two or all three fields, so each field has its own denominator, which is the number of tasks that ask for it. HATCHRCA answers 44 of the 64 cause fields correctly against 23 for the official agent, the largest gain of the three fields.

Table 5: Correct answers per field over the tasks that ask for that field. A field is correct when it matches the ground truth under the OPENRCA criterion.

Field	Tasks asking for it	Official RCA-AGENT	HATCHRCA
Onset time	52	19 (36.5%)	**35 (67.3%)**
Component	57	24 (42.1%)	**42 (73.7%)**
Cause	64	23 (35.9%)	**44 (68.8%)**

Table 6 gives the values behind Figure 4(a). Each cell pairs the validation result of a generated model with that of the frozen reference model set on the same query-level folds. The generated model is equal or better in every cell, and it is strictly better in 12 of the 16 cells, with the largest margins on the Telecom component models.

Table 6: Validation quality of the small models generated by the LLM, with the frozen reference model set in parentheses. Fault-type is the accuracy of the fault-type model. Lat, Pkt and Res are the top-k recall of the component models for latency, packet loss and resource faults. All values are measured on training queries.

Dataset	Fault-type	Lat	Pkt	Res
Telecom	0.588 (0.588)	0.889 (0.111)	0.800 (0.000)†	0.850 (0.050)
Bank	0.637 (0.627)	0.800 (0.550)	0.875 (0.667)	0.931 (0.931)
cloudbed-1	0.755 (0.755)	0.000 (0.000)†	0.400 (0.000)†	0.775 (0.575)
cloudbed-2	0.814 (0.678)	0.667 (0.333)	0.438 (0.312)	0.825 (0.750)

3 cells of Table 6 are marked † because the frozen reference scores zero on that class, namely Telecom packet loss, cloudbed-1 latency and cloudbed-1 packet loss, so a generated model that merely equals the reference there has not been tested against anything.

Table 7 gives the values behind Figures 4(b) and (c). Only Bank passed on the first training program, and only Bank admitted its cause model, because the other three cause models stay below the floor of 0.40. The last three columns list the measured accuracy a_f that caps the confidence of each component model at diagnosis.

Table 7: Admission outcomes per dataset. Rounds counts training programs until admission, first round states whether the first program passed, and cause accuracy is the validation accuracy of the cause model against the floor of 0.40. Measured accuracy is a_f of Equation 3.

		First	Cause	Cause model	Measured accuracy a_f		
Dataset	Rounds	round	accuracy	admitted	Lat	Pkt	Res
Telecom	2	rejected	0.353	no	0.333	0.400	0.750
Bank	1	passed	0.529	yes	0.750	0.708	0.793
cloudbed-1	2	rejected	0.057	no	0.300	0.300	0.575
cloudbed-2	3	rejected	0.220	no	0.333	0.300	0.550

G LIMITATIONS

Five limits bound the claims of this paper. First, the main configuration still carries authored prompt text and 9 rules ported by hand, and the autonomous configuration without them solves 9 fewer tasks, so part of the reported gain depends on authored material. Second, no experiment disables admission alone or replaces the generated models with the reference model set inside an otherwise identical pipeline, so the individual shares of the small models and of admission in the gain are not isolated. Third, the admission checks read the metrics that each program reports, and the guarantee that these metrics are genuine comes from the separate re-execution in Section 4.4 rather than from the checks themselves. Fourth, the model bars are relative to a reference, and 3 of the 16 comparisons used a reference that scores zero on its class, so on those cells the bar could not reject a model. The cloudbed-1 latency model passed in this way with zero recall, and its measured accuracy was therefore capped at the floor ℓ. Fifth, each task was scored once on 87 tasks, so the reported intervals capture variation over tasks but not over repeated onboarding runs of a stochastic generator.