LOCATING THE MODEL–HARNESS BOUNDARY IN INTERACTIVE DIAGNOSIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Interactive diagnosis requires an agent to recognize diseases, maintain evidence state, decide next steps, and determine when evidence is sufficient for a diagnosis. Existing LLM-based medical agents typically place this entire sequential control loop inside the model. We study a fundamental architectural question: which responsibilities should remain inside the LLM, and which should be executed by the surrounding harness? We decompose interactive diagnosis into evidence-state maintenance, search strategy, and terminal control, progressively externalizing them to a rule-bound runtime that executes machine-readable diagnostic programs compiled from clinical knowledge. At the fully externalized endpoint, the full harness directly takes over these responsibilities, restricting the LLM primarily to language–symbol grounding and residual reasoning for cases not resolved by the programs. Across three interactive medical benchmarks, the full harness improves diagnostic correctness by 24.7–49.7 percentage points over prompting with the same programs, exposes explicit execution provenance, and narrows performance variation across model scales. These results highlight a critical distinction between merely accessing procedural knowledge and reliably executing it. The effectiveness of an interactive agent depends not just on the underlying model's capability, but on how sequential control is strategically divided between the model and its execution harness during diagnosis.[1]

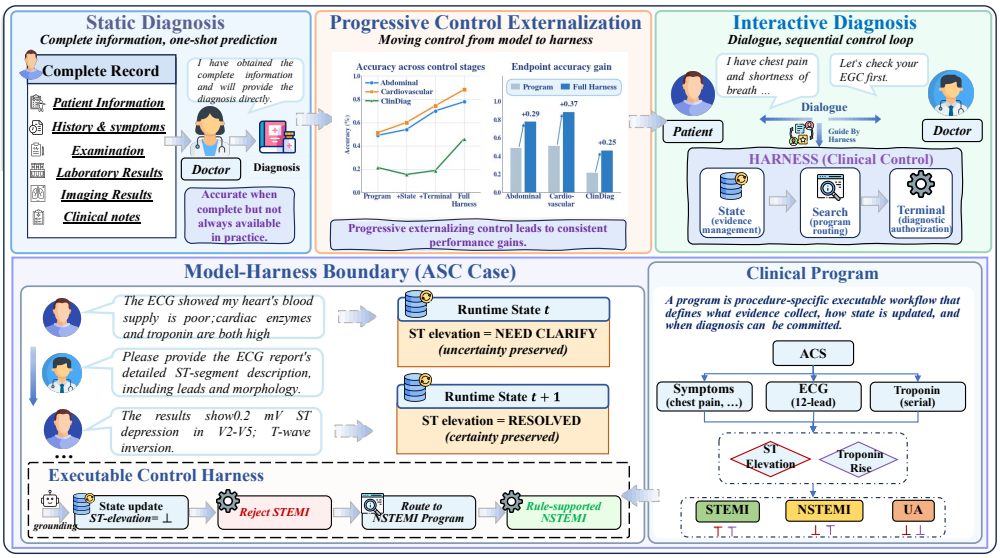

Figure 1: **Progressive Control Externalization in Interactive Diagnosis.** Progressively offloading state, search, and terminal control from the LLM to a rule-bound execution harness yields consistent accuracy improvements. *Note: "Full Harness" in the figure denotes proposed **Full Harness (Hybrid)**.*

[1]Code and data are available here.

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.

1 INTRODUCTION

As large language models (LLMs) scale, their performance on medical QA and case-based diagnosis has become increasingly powerful (Singhal et al., 2023; 2025; Nori et al., 2023; McDuff et al., 2025). However, the vast majority of existing evaluations are predicated on static records, where symptoms, lab results, and medical histories have already been fully collected and summarized (Jin et al., 2021; Singhal et al., 2023). Real-world clinical consultation is not the static interpretation of established facts; it is a sequential decision-making process under severe partial observation. In this interactive process, physicians must iteratively gather new patient evidence, dynamically update their epistemic state, and rigorously follow standardized clinical guidelines to determine the next optimal step. Existing agentic systems attempt to solve this by placing the entire sequential control loop inside a single LLM (Tu et al., 2025; Tang et al., 2024; Kim et al., 2024).

Recent evaluations illustrate this phenomenon: although current models achieve high accuracy on complete static records, their performance degrades drastically when placed in interactive settings that require eliciting clinical findings (Hager et al., 2024; Johri et al., 2025; Li et al., 2024; Chen et al., 2026; Schmidgall et al., 2024). This demonstrates that a model may know how to recognize a diagnosis given static features, yet fail completely to acquire and preserve the evidence interactively. Figure 1 previews both the accuracy gap and a matched-content case in which the same programs produce opposite diagnostic commitments. Interactive diagnosis is therefore not static diagnosis with a longer prompt. It adds a sequential control problem: the system must decide what to ask, preserve information across turns, clarify ambiguous observations, and revise the differential as new evidence arrives. While scaling language models can partially improve this issue, achieving rigorous tracking of clinical states requires us to re-examine the system architecture. In modern agent systems, a model becomes an executable agent through a harness: the runtime environment that manages context, state, tool execution, and control flow (Yao et al., 2023; Sumers et al., 2024; Yang et al., 2024). This raises a practical systems engineering question: in complex interactive tasks, which reasoning burdens must reside inside the LLM, and which should be strictly offloaded to the underlying harness? Clarifying this division of labor is critical in medicine, as generic orchestration layers fall short of clinical rigor.

Existing medical agents have not yet fully addressed this allocation of control. Early approaches largely relied on autonomous, open-ended interactions or simulated multi-agent collaboration through role-playing (Tu et al., 2025; Schmidgall et al., 2024; Tang et al., 2024; Kim et al., 2024). To mitigate the divergence of unconstrained LLMs, recent studies have begun introducing clinical guidelines or decision trees as state controls to constrain the model's diagnostic trajectory (Oniani et al., 2024; Sun et al., 2026). However, while these explorations have demonstrated the value of structured clinical protocols, they still leave the core clinical control loop entirely to the LLM's autonomous processing. This includes critical operations such as determining which evidence has been established, which hypotheses should be discarded, and whether the current state is sufficient to make a final diagnosis. Consequently, the entire state-update and decision-making process remains a black box hidden within the model's latent space. Fundamentally, merely prompting an LLM to read text-based clinical workflows is not equivalent to ensuring that the agent can strictly and reliably execute these protocols during dynamic, sequential decision-making.

Motivated by this gap, we explore how to optimally allocate responsibilities between the LLM and an external harness framework. Rather than relying on the model's latent space to implicitly manage the clinical workflow, we decompose the interactive diagnostic process into three core responsibilities: evidence-state maintenance, search strategy, and terminal control. We then progressively externalize these responsibilities from the LLM, transferring them to a rule-bound harness that strictly executes standardized clinical guidelines. This design turns the model–harness boundary itself into an experimental variable: the same procedural knowledge can either be interpreted by the model or executed by the surrounding runtime. At the fully externalized endpoint, the full harness maintains structured patient evidence, selects admissible evidence-acquisition actions and diagnostic programs, and authorizes terminal decisions through executable clinical criteria. The LLM remains responsible primarily for mapping natural language into declared evidence variables, verbalizing runtime-selected questions, and handling residual cases that cannot be resolved by the executable programs.

We conduct comprehensive evaluations across multiple LLM scales on three real-world interactive medical benchmarks. The empirical results demonstrate that shifting the control loop from the LLM to a guideline-based harness yields substantial performance improvements and smaller models can

match or even surpass the performance of larger models. Across model scales, full-harness control also substantially reduces the performance variation observed under model-controlled interaction. Together, these results suggest that agent reliability depends not only on what the model knows, but also on which parts of sequential decision-making it is asked to control. In summary, the main contributions of this work are as follows:

- **Model–Harness Decomposition.** We decompose interactive diagnosis into three core control responsibilities—evidence-state maintenance, search strategy, and terminal control—and make their allocation between the LLM and its execution harness an explicit architectural variable.
- **Executable Harness Design.** We develop a modular, rule-bound harness that maintains structured evidence under partial observation, selects evidence-acquisition and diagnostic-program transitions, and gates diagnostic commitments through machine-executable criteria.
- **Controlled Boundary Study.** Holding the underlying diagnostic programs fixed, we progressively externalize these responsibilities from the model to the runtime and evaluate the resulting systems across three interactive medical benchmarks and multiple model scales. We further analyze abstention, decision provenance, auditable failure states, and interaction cost to characterize both the benefits and limits of executable control.

2 Related Work

Interactive medical agents. Conversational systems such as AMIE optimize history taking and diagnostic dialogue through self-play and inference-time reasoning (Tu et al., 2025). AgentClinic and AI Hospital convert static cases into simulated interactive environments (Schmidgall et al., 2024; Fan et al., 2025); DoctorAgent-RL and MedExAgent learn inquiry and examination policies under multi-turn or noisy partial observation, while PLPF injects physician process preferences into the model (Feng et al., 2026; Gao et al., 2026; Dou et al., 2024). MDAgents, DocCHA, MEDDxAgent, and MeDxAgent organize multiple agents, confidence modules, or consultation stages (Kim et al., 2024; Liu et al., 2025; Rose et al., 2025; Sanghvi et al., 2026). These works establish the importance of sequential interaction. Our focus is complementary: rather than asking only how to improve a learned controller, we hold procedural content fixed and ask which control operations should cease to be learned decisions at inference time.

Executable knowledge and agent harnesses. Computer-interpretable guideline systems, including PROforma and GLIF-style formalisms, established explicit clinical tasks, decision criteria, and machine-executable workflows long before the advent of LLMs (Fox et al., 1998; Peleg et al., 2003; Boxwala et al., 2011; de Clercq et al., 2008). Executable guidelines are now widely used in LLM-based medical tasks (Cao et al., 2026; Zhang et al., 2026; Deng et al., 2026). For example, AIDA-SEAT retrieves a state–evaluation–action tree and directly uses an LLM for state assessment, evaluation, and action execution (Sun et al., 2026). However, these existing approaches still rely on the LLM to interpret and execute the program requirements entirely within its own context window. In contrast, we treat this execution allocation as an explicit experimental variable: the exact same program is either interpreted internally by the LLM or executed externally by a strict runtime environment, while diagnostic commitment is separately controlled. By externalizing the execution, our approach aligns with broader trends in tool-using and program-aided LMs, which explicitly separate language reasoning from external actions or computation (Yao et al., 2023; Gao et al., 2023; Sumers et al., 2024; Yang et al., 2024). By utilizing an agent harness to enforce the boundary between open-ended reasoning and strict execution (Guo et al., 2026), we contribute an empirical boundary study for safety-sensitive clinical settings that embeds evidence provenance and selective prediction directly into the execution trace.

3 Interactive Diagnosis as Executable Control

To operationalize the externalization of control, we formalize the interactive diagnostic process and decompose it into three distinct responsibilities over a shared diagnostic program. We then describe how these responsibilities are instantiated under model-controlled and runtime-controlled execution, and how they are composed at the fully externalized endpoint.

3.1 Problem Formulation

In an interactive clinical consultation, a diagnosis is not given upfront but must be dynamically derived under partial observation. At any conversational turn t, the diagnostic agent's information is restricted to the observed interaction history $h_t = h_0 \oplus (q_1, a_1, \ldots, q_t, a_t)$, where h_0 contains the initial patient presentation, q_i is an inquiry or test request, a_i is the corresponding patient or record response, and $\oplus$ denotes concatenation.

Based on this evolving history h_t, the agent must assess potential diseases, iteratively acquire new evidence, and ultimately decide when to terminate the consultation and issue a diagnosis. To successfully execute this sequential decision-making process, the core controller must manage three distinct responsibilities: *patient evidence state* $\mathcal{S}$ (what is currently known), *search policy* π (what to ask or examine next), and *terminal control* Γ (whether the accumulated evidence licenses a diagnosis). In our formulation, each of these three responsibilities is assigned to either the LLM or a rule-bound runtime harness (L or R), though not fully independently.

3.2 Executable Diagnostic Programs

To externalize the responsibilities $(\mathcal{S}_{\mathrm{R}}, \pi_{\mathrm{R}}, \Gamma_{\mathrm{R}})$ from the LLM, the agent requires procedural clinical knowledge that is both human-readable and machine-executable. We represent standard clinical knowledge as executable rule programs

$$P_d = (E_d, R_d), \tag{1}$$

where the index d ranges over a library $\mathcal{D}$ of target diagnoses. E_d is the set of evidence slots and R_d the set of terminal rules. Each evidence slot $e \in E_d$ names a clinical finding . A rule $r = (\phi_r, k_r, \omega_r) \in R_d$ has a clinical criterion ϕ_r, a action $k_r \in \{\textsc{Complete}, \textsc{Reject}\}$, and priority ω_r.

Crucially, the exact same program P_d can be supplied as text in the prompt for LLM execution, or compiled as code for harness execution. The detailed construction method of library $\mathcal{P} = \{P_d \mid d \in \mathcal{D}\}$ can be found in Appendix B.1, which is built and fixed before evaluation.

3.3 Patient Evidence State

The evidence state $\mathcal{S}$ refers to valid content extracted from patient information throughout the dialogue. The LLM-based state $\mathcal{S}_{\mathrm{L}}$ at turn t is an implicit view $\widehat{M}_t = LLM(h_t)$ of the patient hidden within its latent space, and is prone to hallucination and tracking drift in long multi-turn conversations (Ji et al., 2023; Liu et al., 2024; Laban et al., 2026). In contrast, explicitly externalizing and structuring the patient record prevents the model from guessing directly from raw text. For a given program P_d, the explicit runtime record at turn t is defined as the set of statuses for each evidence slot $e \in E_d$:

$$M_t = \{\sigma_t(P_d, e) \mid e \in E_d\} \tag{2}$$

where $\sigma_t(P_d, e)$ denotes the current status of the evidence. The complete runtime state then aggregates these records over all turns: $\mathcal{S}_{\mathrm{R}} = \bigcup_{t=0}^{T} M_t$. (details in Section C.2).

3.4 Search Policy

To navigate the diagnostic space, the agent's search policy operates at two levels: *inquiry* for new evidence within the active program P_d (denoted π_c^{ask}), or *routing* to evaluate a different program (denoted π_c^{route}), where $c \in \{\mathrm{L}, \mathrm{R}\}$. While the unconstrained LLM policy π_{L} selects actions freely via internal reasoning over the current state, the runtime policy π_{R} enforces a deterministic search. For an active program P_d with a candidate set of unresolved slots $C_t(P_d)$, π_{R} executes an *inquiry* if $C_t(P_d) \neq \emptyset$. Otherwise (upon rule rejection or empty C_t), it *routes* to a new program in eligible queue Q_t. The selections maximize their respective objectives:

$$\pi_{\mathrm{R}}^{\mathrm{ask}} = \arg\max_{e \in C_t(P_d)} U_t(e), \qquad \pi_{\mathrm{R}}^{\mathrm{route}} = \arg\max_{P_{d'} \in Q_t} s_{\mathrm{route},t}(P_{d'}), \tag{3}$$

where utility $U_t(e)$ balances diagnostic progress against acquisition costs, and $s_{\mathrm{route},t}$ ranks candidate programs. Concrete details are deferred to Section 3.6 and Appendix C.3.

3.5 TERMINAL CONTROL AND RESIDUAL HANDLING

The terminal control Γ determines when the agent should halt inquiries and issue a diagnosis. For Γ_{L}, the decision to stop, diagnose, or abstain relies entirely on subjective internal reasoning. By contrast, Γ_{R} operates as a deterministic logic gate. Given the active program P_d, it evaluates the clinical criterion ϕ_r for each rule $r \in R_d$ using three-valued logic (Kleene, 1952) $[\![\phi_r]\!]_{M_t} \in \{\top, \bot, ?\}$ (true, false, unknown). This yields the subset of currently satisfied rules:

$$R_d^+(M_t) = \{r \in R_d \mid [\![\phi_r]\!]_{M_t} = \top\}. \tag{4}$$

When multiple rules are satisfied, the gate selects the rule r^* with the highest priority ω_r and computes the next control state based on its action $k_{r^*} \in \{\textsc{Complete}, \textsc{Reject}\}$:

$$\Gamma_{\mathrm{R}}(M_t, P_d) = \begin{cases} \textsc{Complete}(d, r^*), & \text{if } R_d^+(M_t) \neq \varnothing \text{ and } k_{r^*} = \textsc{Complete} \\ \textsc{Reject}(P_d), & \text{if } R_d^+(M_t) \neq \varnothing \text{ and } k_{r^*} = \textsc{Reject} \\ \textsc{Continue}, & \text{if } R_d^+(M_t) = \varnothing. \end{cases} \tag{5}$$

These outputs dictate the diagnostic workflow: COMPLETE halts the search and outputs diagnosis d supported by rule r^*; REJECT eliminates P_d and invokes the routing policy $\pi_{\mathrm{R}}^{\mathrm{route}}$ to transition to a new program; and CONTINUE delegates back to the inquiry policy $\pi_{\mathrm{R}}^{\mathrm{ask}}$ to gather more evidence.

Residual handling. If the interaction exhausts the search budget or runs out of eligible candidate programs without triggering a COMPLETE rule, the system must terminate gracefully. We govern this via a configurable residual policy $\eta \in \{\mathrm{Rule}, \mathrm{LLM}\}$. Under the Rule configuration, if no eligible programs remain, the system strictly returns an ABSTAIN result rather than guessing (Geifman & El-Yaniv, 2017; Wen et al., 2025), thereby preventing any unsupported diagnostic claims. Conversely, the LLM configuration invokes a fallback mechanism, prompting the model to provide a best-effort diagnostic guess based on the accumulated dialogue history. This design keeps every rule-mode answer traceable to a satisfied rule while allowing empirical exploration.

3.6 THE FULL HARNESS

As Table 1 shows, our variants represent a progressive externalization rather than an exhaustive permutation. First, the explicit runtime state ($\mathcal{S}_{\mathrm{R}}$) is a fundamental prerequisite, as neither programmatic terminal control nor search can operate without it. Second, combining deterministic search with an LLM terminal gate ($\mathcal{S}_{\mathrm{R}}, \pi_{\mathrm{R}}, \Gamma_{\mathrm{L}}$) is structurally redundant. The deterministic search π_{R} inherently encapsulates termination logic by determining when dependencies are fulfilled or the search space is exhausted. Returning the stopping decision to the LLM would only add inconsistency, so externalizing search requires programmatic terminal control Γ_{R}. The variants thus measure what each responsibility adds given the preceding ones, not independent effects.

Following this progressive dependency, the fully externalized configuration assigns all three control responsibilities to the runtime:

$$\mathcal{H}_\eta = (\mathcal{S}_{\mathrm{R}}, \pi_{\mathrm{R}}, \Gamma_{\mathrm{R}}; \eta), \qquad \eta \in \{\mathrm{Rule}, \mathrm{LLM}\}. \tag{6}$$

We refer to this endpoint as the full harness. The Rule ($\eta = \mathrm{Rule}$) and Hybrid ($\eta = \mathrm{LLM}$) variants share this architecture and differ only in residual handling. The complete algorithmic details of this harness framework are provided in Appendix C.

Table 1: **Progressive allocation of control between the model and runtime.** Program Prompt variants cumulatively add explicit state, programmatic terminal gate and search.

Variant	Evidence $\mathcal{S}$	Terminal Γ	Search π	Residual η
Program Prompt	LLM	LLM	LLM	LLM
+ Runtime State	Runtime	LLM	LLM	LLM
+ Runtime Terminal (Rule)	Runtime	Runtime	LLM	Rules
+ Runtime Terminal (Hybrid)	Runtime	Runtime	LLM	LLM
Full Harness (Rule)	Runtime	Runtime	Runtime	Rules
Full Harness (Hybrid)	Runtime	Runtime	Runtime	LLM

4 EXPERIMENTS

Our experiments investigate whether reliable interactive diagnosis requires externalizing sequential control from the LLM. Specifically, we address the following research questions: *how does the progressive offloading of state, search, and terminal control contribute to diagnostic accuracy, and what are the system trade-offs when all three responsibilities are executed by the harness?*

4.1 EXPERIMENTAL SETUP

We evaluate our approach on three benchmarks: **Abdominal** (60 cases, 4 programs; from MIMIC-CDM, Hager et al., 2024, built on MIMIC-IV, Johnson et al., 2023), **Cardiovascular** (60 cases, 8 programs; de-identified inpatient records from a cardiovascular hospital), and **ClinDiag** (100 cases, 100 programs; Chen et al., 2026). Our primary diagnostic backbones are Qwen3-8B (Yang et al., 2025) and the frontier model DeepSeek-V4-Flash (Xu et al., 2026). We evaluate progressive baselines mentioned in Table 1. Reported metrics include correctness (**Corr.**), wrong-answer rate (**Wrong**), and abstention rate (**Abst.**). Detailed definitions of baselines, metrics, and the full evaluation protocol are provided in Appendix D. Subsequent analyses mainly use Qwen3-8B; the corresponding DeepSeek-V4-Flash results are reported in Appendix E.

Table 2: **Interactive diagnosis under partial observation.** All results are the average of three replicated runs, and the sample standard deviation is shown in gray type. †ClinDiag programs are elicited from model knowledge without external guideline provenance.

Method	Abdominal			Cardiovascular			ClinDiag†		
	Corr.↑	Wrong↓	Abst.	Corr.↑	Wrong↓	Abst.	Corr.↑	Wrong↓	Abst.
DeepSeek-V4-Flash									
Direct LLM	49.4 ±5.1	30.0 ±3.3	20.6 ±1.9	71.1 ±1.0	28.3 ±1.7	0.6 ±1.0	38.3 ±0.6	57.7 ±3.2	4.0 ±3.5
LLM + Candidate Set	75.6 ±3.5	21.7 ±4.4	2.8 ±1.0	88.3 ±0.0	11.1 ±1.0	0.6 ±1.0	52.3 ±10.0	46.0 ±7.9	1.7 ±2.1
Program Prompt	55.0 ±3.3	4.4 ±3.5	40.6 ±1.0	53.3 ±5.0	0.6 ±1.0	46.1 ±5.9	18.0 ±7.0	4.0 ±2.6	78.0 ±8.7
+ Runtime State	56.1 ±1.0	6.1 ±1.0	37.8 ±1.0	48.3 ±5.0	1.1 ±1.0	50.6 ±4.2	18.7 ±5.8	6.0 ±2.6	75.3 ±7.8
+ Runtime Terminal (Rule)	56.7 ±1.7	4.4 ±1.0	38.9 ±2.5	33.9 ±3.5	1.7 ±1.7	64.4 ±1.9	6.3 ±2.1	1.7 ±0.6	92.0 ±2.6
+ Runtime Terminal (Hybrid)	69.4 ±2.5	5.0 ±1.7	25.6 ±1.9	88.9 ±1.0	3.9 ±1.0	7.2 ±1.0	43.0 ±11.4	12.7 ±5.5	44.3 ±16.7
Full Harness (Rule)	61.1 ±1.9	1.1 ±1.0	37.8 ±1.0	30.6 ±2.5	0.6 ±1.0	68.9 ±3.5	17.7 ±1.5	0.7 ±0.6	81.7 ±2.1
Full Harness (Hybrid)	**83.3** ±3.3	1.1 ±1.0	15.6 ±2.5	**92.2** ±2.5	1.7 ±0.0	6.1 ±2.5	**67.7** ±4.2	7.7 ±1.5	24.7 ±2.9
Qwen3-8B									
Direct LLM	25.6 ±1.9	32.2 ±6.7	42.2 ±6.7	40.0 ±1.7	44.4 ±2.5	15.6 ±1.9	11.3 ±2.5	46.3 ±0.6	42.3 ±3.1
LLM + Candidate Set	52.2 ±3.5	21.7 ±0.0	26.1 ±3.5	48.9 ±4.2	32.8 ±8.4	18.3 ±4.4	12.3 ±3.1	56.3 ±2.5	31.3 ±2.1
Program Prompt	48.9 ±1.0	16.7 ±5.0	34.4 ±5.1	51.1 ±1.0	26.7 ±2.9	22.2 ±3.5	21.3 ±1.5	11.7 ±1.5	67.0 ±3.0
+ Runtime State	53.9 ±2.5	26.1 ±2.5	20.0 ±5.0	60.0 ±6.7	11.7 ±1.7	28.3 ±6.0	15.3 ±2.5	7.0 ±1.7	77.7 ±4.2
+ Runtime Terminal (Rule)	50.0 ±1.7	5.0 ±1.7	45.0 ±2.9	35.6 ±3.5	2.8 ±1.0	61.7 ±4.4	4.0 ±3.0	1.0 ±1.0	95.0 ±2.6
+ Runtime Terminal (Hybrid)	70.0 ±7.6	6.7 ±3.3	23.3 ±9.3	74.4 ±12.5	12.2 ±1.9	13.3 ±11.7	18.7 ±4.9	14.7 ±2.5	66.7 ±7.4
Full Harness (Rule)	68.9 ±5.4	7.8 ±5.4	23.3 ±0.0	40.0 ±4.4	**0.0** ±0.0	60.0 ±4.4	17.7 ±2.1	1.3 ±1.5	81.0 ±3.6
Full Harness (Hybrid)	**77.8** ±3.5	8.3 ±5.8	13.9 ±2.5	**88.3** ±2.9	3.3 ±0.0	8.3 ±2.9	**46.0** ±5.6	12.0 ±2.0	42.0 ±6.6

4.2 WHAT CHANGES WHEN CONTROL IS EXTERNALIZED?

Same procedures, different execution. As shown in Table 2, Full Harness (Hybrid) outperforms Program Prompt despite both operating over identical diagnostic programs. For instance, the wrong-answer rate for Qwen3-8B on Cardiovascular drops from 26.7% to 3.3%, and DeepSeek-V4-Flash correctness on ClinDiag surges from 18.0% to 67.7%. Furthermore, to prevent varying abstention rates from confounding the full-cohort accuracy, Figure 2 compares the error rates pairwise. For each pair of methods, the comparison is conducted strictly on their jointly answered samples—the specific subset where both methods successfully provide a diagnosis. This controlled comparison confirms that progressively adding harness components steadily decreases errors on these joint cases, allowing the complete harness to achieve the lowest relative error rate.

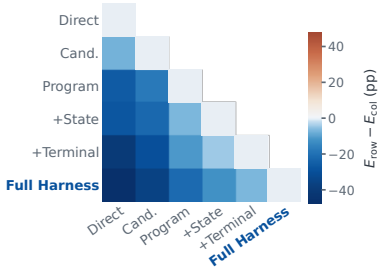

Figure 2: **Pairwise comparison of error rates on jointly answered samples.** Heatmap values represent the error difference ($E_{\text{row}} - E_{\text{col}}$).

Explicit state is a substrate for control, not control itself. Moving only evidence-state maintenance into the runtime has no uniform effect on correctness across the six primary settings. Relative to Program Prompt, + Runtime State changes correctness by -6.0 to $+8.9$ points. For example, Qwen3-8B improves from 51.1% to 60.0% on Cardiovascular, but drops from 21.3% to 15.3% on ClinDiag. Structured state therefore makes the epistemic status of evidence explicit, but does not by itself determine what evidence to acquire next or when a diagnosis should be released.

Terminal control governs authorization rather than acquisition. Incorporating executable terminal rules establishes strict criteria for finalizing a diagnosis. While this effectively suppresses unsubstantiated results, it incurs a severe performance penalty by forcing the agent to abstain whenever evidence is incomplete. For instance, when upgrading Qwen3-8B from + Runtime State to + Runtime Terminal (Rule) on ClinDiag, the wrong-answer rate drops from 7.0% to 1.0%, yet the abstention rate surges to 95.0%. This phenomenon reveals that terminal control acts merely as a rigid gatekeeper: it evaluates *whether the current state justifies a diagnosis*, but lacks the agency to proactively acquire the missing evidence needed to satisfy those criteria.

Runtime search closes the control loop. Building upon explicit state and terminal control, externalizing search (Full Harness (Hybrid)) consistently improves correctness over + Runtime Terminal (Hybrid). Specifically, on ClinDiag, DeepSeek-V4-Flash and Qwen3-8B achieve striking gains of $+24.7$ and $+27.3$ points, respectively. This indicates that in scenarios characterized by vast diagnostic search spaces, externalizing search effectively empowers the agent to proactively acquire missing evidence and evaluate competing clinical hypotheses.

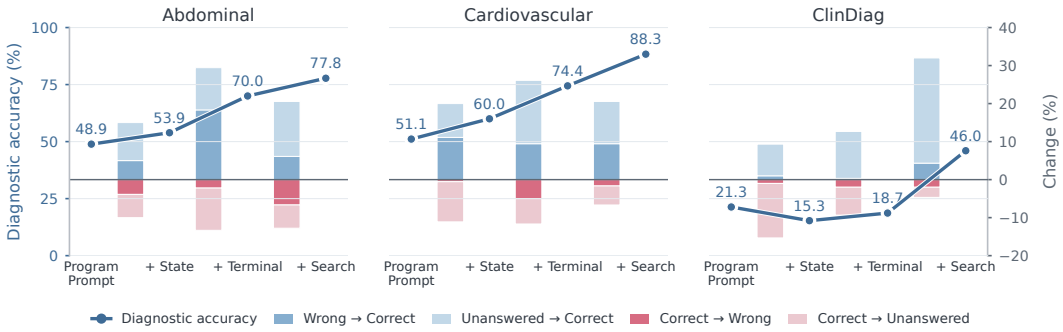

Figure 3: **Progressive externalization of interactive control.** Lines show correctness as evidence state, terminal control, and search are successively transferred from the model to the runtime. Bars report case-level outcome transitions between adjacent configurations.

External control compensates for internal capacity. As Figures 3 and 10 illustrate, a model's reliance on external control depends heavily on its internal knowledge reserves. Faced with the 100 candidate diseases in ClinDiag, DeepSeek-V4-Flash maintains robustness due to its broad medical knowledge. In contrast, the smaller Qwen3-8B lacks this domain familiarity, causing its performance to plummet under these rigid constraints. By actively offloading this cognitive burden to the runtime, Full Harness (Hybrid) elevates Qwen3-8B's correctness to 46.0%. This demonstrates that an external harness is indispensable to overcome internal capacity limits, especially for small models.

A functional division of labor. Taken together, these progressive variants reveal that runtime control modules are not simply interchangeable features. Rather, they represent a strict division of labor: state specifies *what is known*, terminal control dictates *when to commit*, and search determines *how to navigate ambiguity*. The full Full Harness (Hybrid) harness performs best when these responsibilities operate as a closed, deterministic loop; as each variant builds on the previous ones, these are incremental rather than independent effects.

4.3 Where Does Executable Control Stop?

Residual reasoning defines the other side of the boundary. Full externalization does not imply that clinical diagnosis can be entirely reduced to pure rule execution. While the Full Harness (Rule) and Full Harness (Hybrid) variants share the same runtime mechanisms for state tracking and search, they diverge significantly when no rule is perfectly satisfied. As demonstrated across the six primary settings, Full Harness (Hybrid) outperforms Full Harness (Rule) by a massive margin of 8.9–61.6

points in correctness. This stark contrast highlights the inherent brittleness of rigid rules in medical contexts. Therefore, executable control does not eliminate the LLM's reasoning capacity; rather, it acts as a structural filter. It handles deterministic navigation and delegates to the LLM's "residual" reasoning only when strict programmatic logic hits a dead end, fundamentally changing *where* and *why* the model is invoked.

The limits of residual reasoning under missing evidence. While residual reasoning effectively overcomes the brittleness of strict rules, this resilience introduces a new trade-off when critical evidence is genuinely absent. To explore this boundary, we conduct an ablation by actively masking rule-supporting imaging findings in patient cases. Under these conditions, as shown in Table 3, the failure modes diverge sharply: Full Harness (Rule) fails conservatively by halting, driving the abstention rate (ΔA) up by 55.0 points on Abdominal. In contrast, Full Harness (Hybrid) relies on residual reasoning to bridge the gap, trading strict safety for aggressive guesswork and raising the wrong-answer rate (ΔW) by 11.1 points on Cardiovascular. Ultimately, this highlights the core trade-off: the hybrid fallback successfully prevents procedural stalling, but risks unsafe clinical guesswork when pushed beyond the boundaries of available evidence.

Table 3: **Outcome changes after removing rule-supporting imaging evidence.** Paired Δpp vs. the full-record cohort. $\Delta C/\Delta W/\Delta A$ are correctness, wrong, and abstention; ΔP is answered-case correctness.

	ΔC	ΔW$\downarrow$	ΔA	ΔP$\uparrow$
Full Harness (Rule)				
Abd.	-47.2 $\pm$3.4	-7.8 $\pm$4.8	+55.0 $\pm$1.7	+6.5 $\pm$4.7
Cardio.	-24.4 $\pm$5.4	-0.6 $\pm$2.0	+25.0 $\pm$4.4	-0.7 $\pm$8.2
Full Harness (Hybrid)				
Abd.	-36.1 $\pm$6.3	-6.6 $\pm$4.4	+42.8 $\pm$7.0	+4.4 $\pm$4.1
Cardio.	-23.3 $\pm$7.6	+11.1 $\pm$4.2	+12.2 $\pm$3.9	-13.9 $\pm$5.5
Program Prompt				
Abd.	-18.9 $\pm$7.5	-2.8 $\pm$5.4	+21.6 $\pm$10.1	-7.6 $\pm$7.3
Cardio.	-21.7 $\pm$1.7	-2.8 $\pm$1.0	+24.4 $\pm$1.0	-8.2 $\pm$1.8

Why do executable programs fail to close? To quantify just how pervasive this issue of missing evidence is, we audit the execution traces to pinpoint exactly where strict logic breaks down. As shown in Table 4, across all case-runs, only 35.4% successfully terminate via a reference rule. More crucially, 49.8% successfully route to the correct reference program but fail to trigger a terminal release, dwarfing the 14.8% that never reach the target program at all. Within these stalled trajectories, the overwhelming bottleneck is indeed "unresolved evidence". By manually examining specific patient cases, we found that patient medical record information may be incomplete or ambiguous, creating gaps with the program library. As a result, executable control logic fails when confronted with ambiguous clinical scenarios. Although Full Harness (Hybrid) leverages the LLM to compensate for missing evidence, this approach is essentially a compromise. Finding a balance between safety and accuracy remains a challenge.

Table 4: **Auditable execution states in interactive diagnosis.** Distribution of execution outcomes pooled across Qwen3-8B and DeepSeek-V4-Flash.

Execution state	Diagnostic implication	Abd.	Cardio.	ClinDiag	All
Closed by reference rule	The program fired and gave the final answer.	65.0%	35.3%	17.7%	35.4%
Reached, not closed	The program was entered but never fired.	34.2%	49.2%	59.7%	49.8%
• Evidence unresolved	A required finding was never elicited.	27.8%	36.1%	49.3%	39.8%
• Criteria not met	Findings failed the targeted rule's criteria.	4.2%	13.1%	9.3%	8.9%
• Criteria overridden	A higher-priority rule outranked the match.	2.2%	0.0%	1.0%	1.1%
Program not reached	Interaction never entered the program.	0.8%	15.6%	22.7%	14.8%

4.4 Does the Boundary Persist Across Model Scale?

Harness control compresses the performance gap across scales. Across the Qwen3 family, baseline performance under LLM + Candidate Set scales steeply with model size. In contrast, the Full Harness (Hybrid) framework significantly flattens this trajectory, particularly on ClinDiag, as shown in Figure 4. By delegating state maintenance, search, and terminal authorization to the runtime, smaller models recover much of the performance typically achieved only through scaling. This suggests that part of the advantage of larger models comes from their capacity to internalize procedural control, not only from superior medical reasoning. As further detailed in Figure 4, while rule-supported decisions achieve high precision, their coverage remains incomplete and setting-dependent. Therefore, the

harness does not obsolete the model; instead, it establishes a reliable executable pathway for cases that can be closed programmatically. For all other scenarios, the model's diagnostic capabilities remain indispensable. This explains how the harness narrows the performance gap across scales, while simultaneously amplifying the model's intrinsic capabilities through structural support.

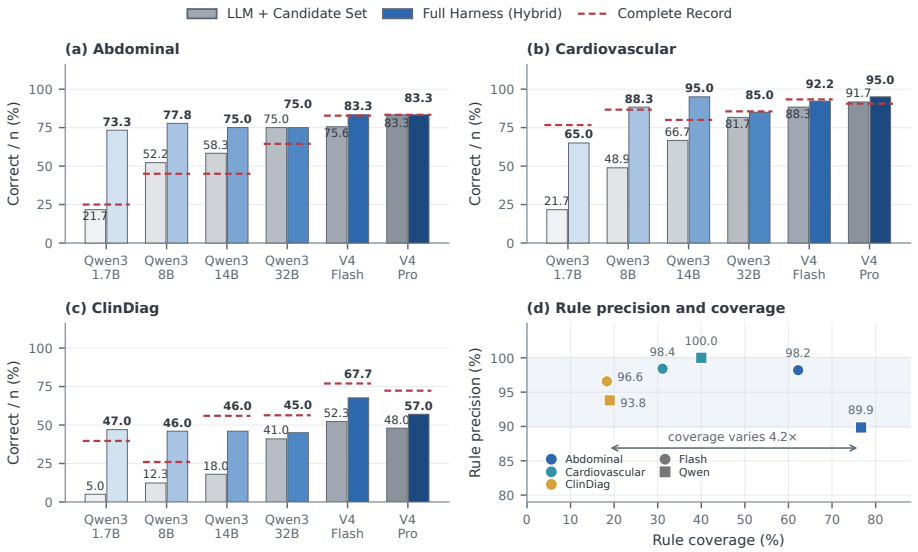

Figure 4: **Backbone sensitivity and decision provenance.** **(a–c)** Paired bars compare LLM + Candidate Set (gray) with Full Harness (Hybrid) (blue). Red dashes mark Complete Record on the same backbone. **(d)** Rule precision against coverage.

Externalization is not free. As shown in Figure 5, the full harness does not uniformly minimize execution cost. In some complex settings it achieves higher correctness with fewer model calls than program-based prompting variants, whereas on smaller program libraries Program Prompt can terminate with fewer calls. We therefore treat interaction cost as a design trade-off rather than a primary advantage of externalization; full accounting of model calls, patient turns, and injected program context is reported in Table 16.

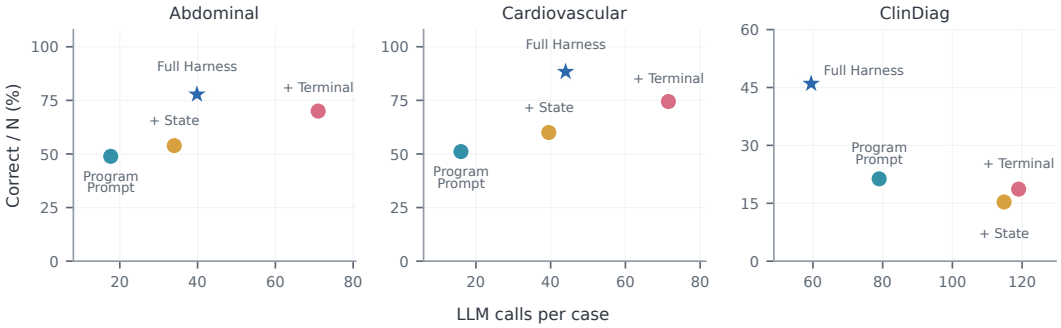

Figure 5: **Accuracy against interaction cost.** Different variants use Qwen3-8B as the diagnostic backbone to compare correctness and patient interaction costs across three benchmarks.

5 Conclusion

Interactive diagnosis reveals a broader lesson for language agents: procedural knowledge is not procedural control. Reliability improves when state, search, and commitment are treated as executable runtime semantics rather than latent model responsibilities, while the LLM is reserved for semantic grounding and cases that formal programs cannot close. This suggests a different path to stronger agents: not only improving what the model can do, but explicitly deciding what it should no longer be asked to control. The model–harness boundary should therefore be treated as a first-class architectural choice.

Ethics Statement

This work studies an LLM agent and its harness in simulated diagnostic interaction; it is not a clinical device. The reported experiments do not involve prospective care or treatment recommendations. Any release of case data, traces, or program artifacts must follow the licenses, de-identification requirements, and review procedures of the underlying sources. We explicitly avoid claims of readiness for autonomous clinical deployment.

Reproducibility Statement

We release the code, the diagnostic programs, and the presentation graphs; data access for each benchmark is described in Section D. The appendix also contains detailed experimental setup and descriptions, including library construction, model and evaluation details, and prompt templates.

AI Use Statement

We used Generative AI to polish the draft of our paper, but all experimental results and core viewpoints were independently completed by us.

References

Aziz A. Boxwala, Beatriz H. Rocha, Saverio M. Maviglia, Vipul Kashyap, Seth Meltzer, Jihoon Kim, Ruslana Tsurikova, Adam Wright, Marilyn D. Paterno, Amanda Fairbanks, and Blackford Middleton. A multi-layered framework for disseminating knowledge for computer-based decision support. *J. Am. Medical Informatics Assoc.*, 18(Supplement):132–139, 2011.

Lang Cao, Yuhao Shen, Tianyang Luo, Simo Du, Hao Peng, and Yue Guo. Guideskill: Evolving executable llm agent skills for guideline-grounded clinical reasoning. *arXiv preprint arXiv:2607.26160*, 2026.

Xi Chen, Hanyu Zhou, Huahui Yi, Mingke You, WeiZhi Liu, Li Wang, Ziyuan Qin, Hairui Li, Xue Zhang, Yingman Guo, et al. Grounding large language models in clinical diagnostics. *Nature Communications*, 2026.

Paul A. de Clercq, Katharina Kaiser, and Arie Hasman. Computer-interpretable guideline formalisms. 139:22–43, 2008.

Ruiqi Deng, Geoffrey Martin, Tony Wang, Gongbo Zhang, Yi Liu, Chunhua Weng, Yanshan Wang, Justin F Rousseau, and Yifan Peng. Cpgprompt: Translating clinical guidelines into llm-executable decision support. *arXiv preprint arXiv:2601.03475*, 2026.

Chengfeng Dou, Ying Zhang, Zhi Jin, Wenpin Jiao, Haiyan Zhao, Yongqiang Zhao, and Zhengwei Tao. Integrating physician diagnostic logic into large language models: Preference learning from process feedback. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 2453–2473, 2024.

Zhihao Fan, Lai Wei, Jialong Tang, Wei Chen, Wang Siyuan, Zhongyu Wei, and Fei Huang. Ai hospital: Benchmarking large language models in a multi-agent medical interaction simulator. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 10183–10213, 2025.

Yichun Feng, Jiawei Wang, Lu Zhou, Zhen Lei, and Yixue Li. Doctoragent-rl: A multi-agent collaborative reinforcement learning system for multi-turn clinical dialogue. pp. 16952–16956, 2026.

John Fox, Nicky Johns, and Ali Rahmanzadeh. Disseminating medical knowledge: the proforma approach. *Artif. Intell. Medicine*, 14(1-2):157–182, 1998.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International conference on machine learning*, pp. 10764–10799. PMLR, 2023.

Yicheng Gao, Xiaolin Zhou, Yahan Li, Yue Zhao, and Ruishan Liu. MedExAgent: Training LLM agents to ask, examine, and diagnose in noisy clinical environments. *arXiv preprint arXiv:2605.07058*, 2026.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Jianyuan Guo, Zhiwei Hao, Chengcheng Wang, Cheng Fan, Tingzhang Luo, Hongguang Li, Ying Gao, Hefei Mei, Jiankun Peng, Rongjian Xu, et al. From question answering to task completion: A survey on agent system and harness design. *arXiv preprint arXiv:2606.20683*, 2026.

Paul Hager, Friederike Jungmann, Robbie Holland, Kunal Bhagat, Inga Hubrecht, Manuel Knauer, Jakob Vielhauer, Marcus Makowski, Rickmer Braren, Georgios Kaissis, et al. Evaluation and mitigation of the limitations of large language models in clinical decision-making. *Nature medicine*, 30(9):2613–2622, 2024.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM computing surveys*, 55(12):1–38, 2023.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.

Alistair EW Johnson, Lucas Bulgarelli, Lu Shen, Alvin Gayles, Ayad Shammout, Steven Horng, Tom J Pollard, Sicheng Hao, Benjamin Moody, Brian Gow, et al. Mimic-iv, a freely accessible electronic health record dataset. *Scientific data*, 10(1):1, 2023.

Shreya Johri, Jaehwan Jeong, Benjamin A Tran, Daniel I Schlessinger, Shannon Wongvibulsin, Leandra A Barnes, Hong-Yu Zhou, Zhuo Ran Cai, Eliezer M Van Allen, David Kim, et al. An evaluation framework for clinical use of large language models in patient interaction tasks. *Nature medicine*, 31(1):77–86, 2025.

Yubin Kim, Chanwoo Park, Hyewon Jeong, Yik Siu Chan, Xuhai Xu, Daniel McDuff, Hyeonhoon Lee, Marzyeh Ghassemi, Cynthia Breazeal, and Hae Won Park. MDAgents: An adaptive collaboration of LLMs for medical decision-making. In *Advances in Neural Information Processing Systems*, volume 37, pp. 79410–79452, 2024.

Stephen Cole Kleene. *Introduction to Metamathematics*. North-Holland, Amsterdam, 1952.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. Llms get lost in multi-turn conversation. In *International Conference on Learning Representations*, volume 2026, pp. 54738–54778, 2026.

Shuyue S Li, Vidhisha Balachandran, Shangbin Feng, Jonathan S Ilgen, Emma Pierson, Pang W Koh, and Yulia Tsvetkov. Mediq: Question-asking llms and a benchmark for reliable interactive clinical reasoning. In *Advances in Neural Information Processing Systems*, 2024.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

Xinyi Liu, Dachun Sun, Yi Fung, Dilek Hakkani-Tur, and Tarek F Abdelzaher. Doccha: Towards llm-augmented interactive online diagnosis system. In *Proceedings of the 26th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pp. 609–619, 2025.

Daniel McDuff, Mike Schaekermann, Tao Tu, Anil Palepu, Amy Wang, Jake Garrison, Karan Singhal, Yash Sharma, Shekoofeh Azizi, Kavita Kulkarni, et al. Towards accurate differential diagnosis with large language models. *Nature*, 642(8067):451–457, 2025.

Harsha Nori, Nicholas King, Scott Mayer McKinney, Dean Carignan, and Eric Horvitz. Capabilities of gpt-4 on medical challenge problems. *arXiv preprint arXiv:2303.13375*, 2023.

David Oniani, Xizhi Wu, Shyam Visweswaran, Sumit Kapoor, Shravan Kooragayalu, Katelyn Polanska, and Yanshan Wang. Enhancing large language models for clinical decision support by incorporating clinical practice guidelines. In *2024 IEEE 12th International Conference on Healthcare Informatics (ICHI)*, pp. 694–702. IEEE, 2024.

Mor Peleg, Samson W. Tu, Jonathan Bury, Paolo Ciccarese, John Fox, Robert A. Greenes, Richard W. Hall, Peter D. Johnson, Neill Jones, Anand Kumar, Silvia Miksch, Silvana Quaglini, Andreas Seyfang, Edward H. Shortliffe, and Mario Stefanelli. Research paper: Comparing computer-interpretable guideline models: A case-study approach. *J. Am. Medical Informatics Assoc.*, 10(1): 52–68, 2003.

Daniel Philip Rose, Chia-Chien Hung, Marco Lepri, Israa Alqassem, Kiril Gashteovski, and Carolin Lawrence. Meddxagent: A unified modular agent framework for explainable automatic differential diagnosis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13803–13826, 2025.

Akshat Sanghvi, Naren Akash, Raza Imam, Amit Sharma, and Mohit Jain. MeDxAgent: Multi-agent consultation for interactive medical diagnosis. *arXiv preprint arXiv:2606.03416*, 2026.

Samuel Schmidgall, Rojin Ziaei, Carl Harris, Eduardo Pontes Reis, Jeffrey K. Jopling, and Michael Moor. Agentclinic: a multimodal agent benchmark to evaluate AI in simulated clinical environments. *CoRR*, abs/2405.07960, 2024. doi: 10.48550/ARXIV.2405.07960. URL https://doi.org/10.48550/arXiv.2405.07960.

Karan Singhal, Shekoofeh Azizi, Tao Tu, S Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. Large language models encode clinical knowledge. *Nature*, 620(7972):172–180, 2023.

Karan Singhal, Tao Tu, Juraj Gottweis, Rory Sayres, Ellery Wulczyn, Mohamed Amin, Le Hou, Kevin Clark, Stephen R Pfohl, Heather Cole-Lewis, et al. Toward expert-level medical question answering with large language models. *Nature medicine*, 31(3):943–950, 2025.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024.

Lianxin Sun, Xiaoying Ying, Guangya Yu, Weiyan Zhang, Chenhao Guan, Hao He, Mingxi Shang, Jianhua Li, ChunMing Wang, and Tong Ruan. Aida-seat: Towards reliable ai doctor assistant via state-evaluation-action tree enhanced llms in online hospital. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL 2026)*, pp. 704–715, 2026.

Xiangru Tang, Anni Zou, Zhuosheng Zhang, Ziming Li, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. Medagents: Large language models as collaborators for zero-shot medical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 599–621, 2024.

Tao Tu, Mike Schaekermann, Anil Palepu, Khaled Saab, Jan Freyberg, Ryutaro Tanno, Amy Wang, Brenna Li, Mohamed Amin, Yong Cheng, et al. Towards conversational diagnostic artificial intelligence. *Nature*, 642(8067):442–450, 2025.

Bingbing Wen, Jihan Yao, Shangbin Feng, Chenjun Xu, Yulia Tsvetkov, Bill Howe, and Lucy Lu Wang. Know your limits: A survey of abstention in large language models. *Transactions of the Association for Computational Linguistics*, 13:529–556, 2025.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023.

Yichi Zhang, Nabeel Seedat, Yinpeng Dong, Peng Cui, Jun Zhu, and Mihaela van de Schaar. Guideline-grounded evidence accumulation for high-stakes agent verification. *arXiv preprint arXiv:2603.02798*, 2026.

Contents

A Diagnostic Programs **15**
A.1 Evidence Slots . . . 15
A.2 Terminal Rules and Three-Valued Evaluation . . . 15
A.3 What Missing Evidence Can and Cannot Change . . . 16
A.4 Worked Example: The NSTEMI Program . . . 16

B Building the Program Library and the Presentation Graph **17**
B.1 Program Library . . . 17
B.2 Presentation Graph . . . 18

C How the Full Harness Runs a Consultation **21**
C.1 Which Responsibilities Move to the Runtime . . . 21
C.2 Updating the Evidence State ($\mathcal{S}_{\mathrm{R}}$) . . . 21
C.3 Choosing the Next Question ($\pi_{\mathrm{R}}^{\mathrm{ask}}$) . . . 22
C.4 Choosing the Next Program ($\pi_{\mathrm{R}}^{\mathrm{route}}$) . . . 22
C.5 Terminal Gate and Residual Handling (Γ_{R}, η) . . . 22
C.6 Consultation Algorithm . . . 23

D Experimental Setup **23**
D.1 Benchmarks . . . 23
D.2 Simulator, Budgets, and Scoring . . . 24
D.3 Baselines . . . 24
D.4 Metrics . . . 24

E Additional Results **26**
E.1 Additional Baselines . . . 26
E.2 Robustness Checks . . . 28
E.3 Inside the Full Harness . . . 32
E.4 Interaction Cost . . . 34
E.5 Case Study: One NSTEMI Consultation . . . 35

F Prompt Templates **36**
F.1 Program Library Prompts . . . 36
F.2 Presentation Graph Prompts . . . 42
F.3 Consultation Prompts . . . 44
F.4 Evaluation Prompts . . . 46

A DIAGNOSTIC PROGRAMS

This section specifies the diagnostic program $P_d = (E_d, R_d)$ introduced in Section 3, including its evidence declarations, terminal rules, and evaluation under partial information.

A.1 EVIDENCE SLOTS

In standard LLM consultations, the model can freely generate unbounded clarifying questions. To explicitly manage the patient evidence state $\mathcal{S}_{\mathrm{R}}$, the program strictly bounds the required information space via E_d. Each evidence slot $e \in E_d$ defines a specific clinical condition or piece of information required to formulate the diagnosis. It is formally parameterized as:

$$e = \langle \text{type}, \text{kind}, \text{ask}, \text{judgment_note} \rangle \tag{7}$$

Here, type specifies the algebraic format (Boolean, numeric, categorical) and kind categorizes the clinical source (e.g., patient history, laboratory, imaging).

To operationalize data acquisition and evaluation systematically, each evidence slot relies on preset configurations rather than open-ended model generation. Specifically, the **`ask`** field provides a pre-configured question presented to the patient. It serves as a targeted, suggested inquiry designed to elicit a precise and accurate response regarding the required evidence. Once the patient provides this information, the **`judgment_note`** field supplies the explicit evaluation criteria.

The evaluation of the patient's response is handled mechanically depending on the data format: numerical thresholds and rule-based criteria are evaluated programmatically by the runtime engine, whereas the LLM is invoked solely to perform semantic assessment on unstructured natural language text against the judgment note. This hybrid evaluation determines whether the required condition is satisfied and assigns the appropriate value to the slot.

By structuring the state boundaries this way, the inquiry policy π_{R} effectively uses the `ask` to gather data, while the LLM acts merely as a constrained perception module rather than a free-reasoning engine. This ensures standardized evaluation and prevents the system from inventing ad-hoc diagnostic criteria mid-conversation.

A.2 TERMINAL RULES AND THREE-VALUED EVALUATION

The terminal rules R_d instantiate the terminal control gate Γ_{R}. To formalize this control, we represent clinical guidelines as executable **logical predicates**. This allows the engine to systematically evaluate the clinical criterion ϕ_r of any given rule. However, medical dialogues are inherently partially observable—in clinical practice, the absence of evidence is not evidence of absence. To safely evaluate these predicates under uncertainty, the execution engine strictly operates over a **three-valued logic system**: True ($\top$), False ($\bot$), and Unknown (?).

Let c be a valid constant and C be a set of categorical values. The atomic predicates p and composite criteria ϕ follow the grammar:

$$p ::= e \bowtie c \mid e \in C \mid s(E') \bowtie c, \tag{8}$$

$$\phi ::= p \mid \mathrm{all}(\phi_1, \ldots, \phi_k) \mid \mathrm{any}(\phi_1, \ldots, \phi_k) \mid \mathrm{not}(\phi) \mid \mathrm{atleast}(k; \phi_1, \ldots, \phi_n). \tag{9}$$

Under this logical system, standard value comparisons read only resolved findings; an unresolved slot yields ? rather than an assumed negative. Consequently, logical operators strictly propagate this uncertainty. For example, standard conjunction (all) evaluates to $\bot$ immediately if any condition is $\bot$, evaluates to $\top$ only when all are $\top$, and yields ? if pending conditions prevent a definitive conclusion.

Beyond simple boolean logic, medical decision-making frequently requires evaluating complex judgments that aggregate multiple conditions, such as **clinical scoring mechanisms** and **risk stratification tools**. To support these complex evaluations, we introduce an additive clinical score representation, defined as $s = \sum_j w_j b_j$.

Crucially, when evaluating these scores, unresolved components are never naively imputed as zero, as doing so could prematurely dismiss a high-risk patient. Instead, the runtime computes the set of all

mathematically reachable totals given the currently resolved state M_t:

$$\mathcal{V}_t(s) = \left\{ \sum_j w_j b_j : b_j \in \begin{cases} \{1\}, & [\![\phi_j]\!]_{M_t} = \top, \\ \{0\}, & [\![\phi_j]\!]_{M_t} = \bot, \\ \{0,1\}, & [\![\phi_j]\!]_{M_t} = ? \end{cases} \right\}. \tag{10}$$

A score predicate $s(E') \bowtie c$ evaluates to $\top$ only if *every* value in $\mathcal{V}_t(s)$ satisfies the condition. This rigorously guarantees that the clinical threshold is met regardless of how the pending variables are ultimately resolved by the patient.

A.3 What Missing Evidence Can and Cannot Change

The three-valued logic and range-based scoring mechanisms introduced above serve a critical structural purpose: missing information cannot turn a false criterion into a true one. We formalize this safety guarantee as *partial-information monotonicity*.

In a standard diagnostic process, if the LLM fails to extract a piece of evidence, a resolved clinical finding is downgraded to an unknown state (?). Under our logical framework, replacing a known True ($\top$) or False ($\bot$) with an Unknown (?) can only make the final composite criterion become ?. It is mathematically impossible for missing information to flip a False criterion into a True one, or vice versa.

As a result, incomplete information leaves a criterion unresolved and prompts further questions instead of a diagnosis. The guarantee covers missing evidence, not misread evidence: a slot grounded to the wrong value can still fire a rule, which is where rule-wrong answers come from.

A.4 Worked Example: The NSTEMI Program

We illustrate the representation with the `nstemi_acs` program for non-ST-elevation myocardial infarction and acute coronary syndrome. The excerpt below shows its serialized evidence declarations and terminal rules.

Program example: serialized evidence and terminal rules

```
{
  "schema": "clinical_rulepack_v1", "id": "nstemi_acs", "version": "1.0.0",
  "eligibility": null, "scores": [],
  "evidence": [{"id": "ecg_stemi_pattern", "type": "bool", "kind": "imaging",
     "ask": "Quote the ECG report: STEMI pattern, or test not done?",
     "judgment_note": "An unassessed finding is unavailable, not negative."}
    // Other slot declarations and descriptive fields omitted.
  ],
  "rules": [{"id": "r_stemi_exclude", "priority": 1000,
     "when": {"e": "ecg_stemi_pattern", "op": "eq", "value": true},
     "then": {"disposition": "reject"}},
    {"id": "r_ecg_unavailable", "priority": 950,
     "when": {"e": "ecg_stemi_pattern", "op": "unavailable"},
     "then": {"disposition": "reject"}},
    {"id": "c_nstemi", "priority": 900,
     "when": {"all": [
       {"e": "ecg_stemi_pattern", "op": "eq", "value": false},
       {"e": "myocardial_injury_marker_elevated",
        "op": "eq", "value": true},
       {"any": [
         {"e": "ecg_ischemic_st_t_changes", "op": "eq", "value": true},
         {"e": "regional_wall_motion_abnormality",
          "op": "eq", "value": true},
         {"e": "coronary_obstructive_lesion", "op": "eq", "value": true}
       ]}
     ]},
     "then": {"disposition": "complete"}}
    // Eight lower-priority rules omitted.
  ],
  "match_policy": {"strategy": "priority_first", "on_tie": "error"}
}
```

How the gate responds to the ECG slot. With the static program defined, we now illustrate how the execution engine dynamically evaluates these rules to drive the diagnostic conversation. To see

this in action, assume a scenario where the patient's injury-marker and ischemic-ST-T slots have been successfully resolved as TRUE ($\top$) in $\mathcal{S}_{\mathrm{R}}$, while all other slots remain unresolved (?). The following table illustrates how varying the state of the `ecg_stemi_pattern` slot strictly governs the terminal gate Γ_{R} and the subsequent action of the search policy π_{R}:

STEMI-pattern state	$[\![\phi_{\texttt{c_nstemi}}]\!]$	**Terminal Gate (Γ_{R}) & Search Policy (π_{R})**
UNKNOWN	?	CONTINUE: Γ_{R} yields $\varnothing$; $\pi_{\mathrm{R}}^{\mathrm{ask}}$ queries the missing ECG.
RESOLVED, false	$\top$	COMPLETE: `c_nstemi` matches; diagnosis authorized.
RESOLVED, true	$\bot$	REJECT: `r_stemi_exclude` matches; $\pi_{\mathrm{R}}^{\mathrm{route}}$ switches program.
UNAVAILABLE	?	REJECT: `r_ecg_unavailable` explicitly intercepts missing data.

Notice the critical role of the `r_ecg_unavailable` rule. If the ECG was simply never performed, the ordinary clinical evaluation (`c_nstemi`) gracefully yields ?. However, rather than indefinitely looping to ask for a test the patient cannot provide, the higher-priority explicit availability rule (`r_ecg_unavailable`) intercepts the execution and safely rejects the program. This demonstrates how programmatic rules translate complex clinical guardrails into rigorous, executable control logic.

B BUILDING THE PROGRAM LIBRARY AND THE PRESENTATION GRAPH

Both artifacts are built offline and frozen before evaluation. Neither step sees evaluation patient records.

B.1 PROGRAM LIBRARY

The library $\mathcal{P} = \{P_d : d \in \mathcal{D}\}$ is constructed offline from disease specifications, source documents where available, and a staged multi-agent pipeline. The prompts and their exact context bindings appear in Section F.1. The presentation graph built from this library is described in Section B.2; its prompts appear in Section F.2. Figure 6 summarizes the construction pipeline.

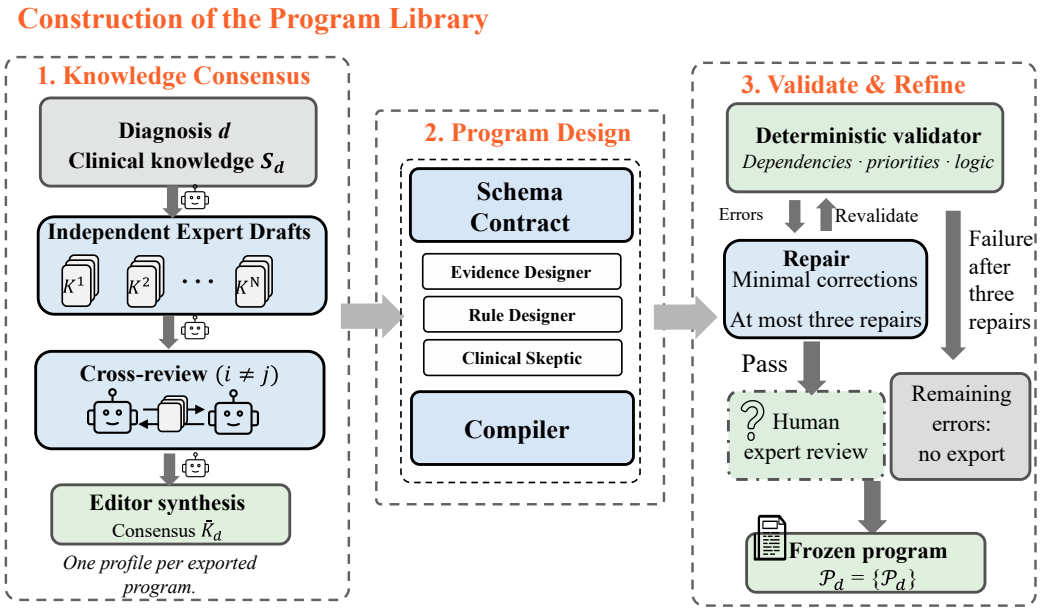

Figure 6: Offline construction of the diagnostic program library. Independent drafts and directed cross-reviews feed an editor consensus, followed by evidence and rule design, clinical critique, and schema-constrained compilation. Structural validation permits at most three repair attempts; artifacts with remaining errors are not exported. The Abdominal and Cardiovascular programs were reviewed by a clinician before the library was frozen; the ClinDiag programs were not reviewed (Section B.1).

1. Drafting and consensus. For a target diagnosis d, supplied references form the source block S_d. Each of N configured expert models independently drafts diagnostic knowledge using the same evidence-based instructions:

$$K_d^{(i)} = \text{Draft}_{\theta_i}(S_d, d), \qquad i = 1, \ldots, N. \tag{11}$$

Each other model then reviews that draft against the references, identifying contradictions, missing criteria, and threshold/unit errors:

$$C_d^{(j \to i)} = \text{Critique}_{\theta_j}(S_d, K_d^{(i)}), \qquad i \neq j. \tag{12}$$

The reviewer receives the author's draft and sources, not the other drafts or its own previous draft. The editor receives all drafts and directed reviews:

$$\bar{K}_d = \text{Synthesize}_{\theta_{\text{editor}}}(S_d, \{K_d^{(i)}\}, \{C_d^{(j \to i)}\}_{i \neq j}). \tag{13}$$

It preserves source tags, corrects identified errors, and records unresolved disagreements explicitly. Source-backed assertions take precedence over model knowledge. A specification without references is explicitly marked as knowledge-only; consensus does not itself establish source fidelity.

2. Evidence and rule design. The Evidence Designer converts $\bar{K}_d$ into typed slots with patient-answerable `ask` text and report-level `judgment_note` criteria. The Rule Designer receives both the consensus and this slot table to specify scores, priorities, and `complete`/`reject` exits. A Clinical Skeptic then receives both designs and the consensus, testing for permissive completion rules, unanswerable questions, missingness errors, threshold mismatches, and coverage gaps. Finally, the Compiler receives all these artifacts, the field contract, and a fictitious structural example, and emits one JSON program constrained to the schema in Section F.1.1. These are distinct role-specific calls, even when they use the same underlying model.

3. Validation and repair. A deterministic validator checks the compiled artifact for structural errors, including invalid dependencies, duplicate priorities, and selected dead or tautological conditions. If errors remain, a Repairer receives the current JSON, errors, warnings, and contract; it makes minimal corrections and the validator runs again. The default limit is three repairs. Programs with remaining errors are not exported to the top-level output directory. An optional runtime-parser load check provides a separate compatibility check. The pipeline caches all intermediate artifacts for inspection and resumption; validation establishes executable integrity, not clinical correctness.

Expert review. The 12 Abdominal and Cardiovascular programs (96 terminal rules, 126 evidence slots) were reviewed by one clinician before the library was frozen and before any evaluation run. The reviewer checked each program against its source guideline, covering evidence slots, thresholds, exclusion conditions, and terminal rules, and edited the programs directly. The 100 ClinDiag programs were compiled from the model's own knowledge without source guidelines and were not individually reviewed. ClinDiag therefore tests execution and routing over a large program library and does not support claims about the clinical accuracy of the programs.

B.2 Presentation Graph

Relying exclusively on semantic similarity to route between diseases is clinically insufficient; the textual distance between a chief complaint and a disease name does not capture the diagnostic workflows of real physicians. For instance, a doctor might pivot from one hypothesis to another not because the names sound similar, but to rule out a life-threatening condition with overlapping symptoms.

To operationalize this, we construct a weighted *Presentation Graph* $G = (\mathcal{D}, W)$ that explicitly maps the clinical correlations among the diagnostic programs. This graph serves as a structural search prior. When an active program is rejected, the routing policy (Section C.4) performs random walks (Personalized PageRank) over G to yield a clinically logical sequence of next hypotheses. The offline construction of this graph proceeds in four automated steps, summarized in Figure 7.

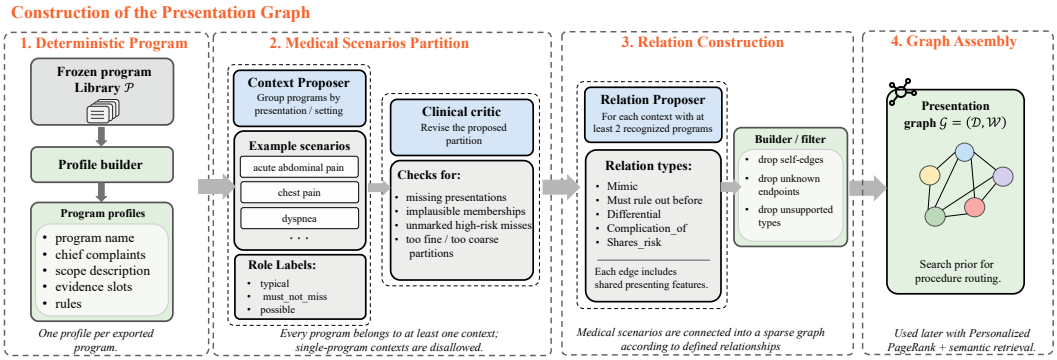

Figure 7: Construction of the presentation graph. Each exported program is reduced to a profile without a model call. A proposer then groups programs into medical scenarios and a critic revises the partition. Within each medical scenario that contains at least two programs, a relation proposer nominates typed edges. The builder drops invalid edges and assembles the weighted graph $G = (\mathcal{D}, W)$ used later as a search prior for program routing.

1. Program profiles. The builder reads each compiled program and writes one profile: program identifier and name, chief complaints and scope description, evidence-slot labels grouped into history, exam, lab, and imaging, and the labels of rules whose action is COMPLETE. This step makes no model call.

2. Medical scenarios. Next, an LLM-based medical scenario proposer groups these isolated profiles into broad clinical presentations (e.g., *acute abdominal pain*). Programs are clustered by how they present in a clinical setting rather than by strict anatomical taxonomy. Each membership is stratified by clinical urgency (e.g., `typical`, `must_not_miss`, `possible`). To ensure medical plausibility, a subsequent LLM critic reviews this partition to detect missing presentations, correct implausible groupings, and ensure high-risk conditions are properly flagged.

3. Relation edges. Within each validated clinical medical scenario, a relation proposer identifies pairwise connections between programs. Rather than generating a dense, fully connected graph, it specifically nominates directed edges that reflect actionable clinical transitions. Each proposed edge is assigned one of five specific clinical relation types (detailed in Table 5), alongside the shared presenting features that justify the link. Self-edges, unrecognized endpoints, and unsupported relation types are deterministically pruned.

4. Edge weights and assembly. To synthesize the final graph, the pipeline merges all nominations for any given program pair across all medical scenarios. If multiple semantic relations are proposed for the same pair, the system selects the relation with the highest base weight (b_{type}), resolving ties based on the number of shared clinical features (n_{shared}). The final directed edge weight w_{ij} is computed as:

$$w_{ij} = \text{clip}_{[\epsilon,\, 1-\epsilon]} \left(b_{\text{type}} + \alpha \min(n_{\text{shared}}, k) + \beta(n_{\text{scenario}} - 1) \right), \tag{14}$$

where n_{scenario} denotes the number of distinct clinical scenario that independently nominated the pair. Here, α and β are scaling factors that explicitly reward clinical feature overlap and diagnostic consensus, respectively, while k caps the overlap reward to prevent weight saturation. The bounds $[\epsilon, 1-\epsilon]$ guarantee strict transition validity for the downstream PageRank algorithm. (The exact empirical values for these hyperparameters are configured during offline tuning and are provided in our open-source implementation).

Finally, an automated structural check validates the assembled graph, rejecting duplicate edges or invalid weights. Isolated programs trigger a system warning rather than being silently dropped. Figure 8 shows the resulting graph for ClinDiag.

Table 5: **Relation types and base weights on the presentation graph.** Each nominated edge represents a specific clinical transition between two diagnostic programs. The base weight (b_{type}) acts as the foundational routing prior before dynamic adjustments for shared features and overlapping scenario.

Relation Type	Directionality	Clinical Rationale	Illustrative Scenario	b_{type}
`mimic`	Bidirectional	Presentations are highly similar; the two conditions are readily mistaken for each other.	*Acute chest pain:* Non-ST-elevation ACS and aortic dissection both present with sudden, severe pain.	0.80
`must-rule-out`	Source → Target	The target condition must be explicitly excluded before the source is diagnosed due to high severity.	*Acute abdominal pain:* Appendicitis must be ruled out before settling on gastroenteritis to avoid perforation.	0.75
`differential`	Bidirectional	Standard differential diagnosis pair within the same overarching presentation.	*Acute abdominal pain:* Appendicitis and cholecystitis (distinguished later by pain site and imaging).	0.60
`complication-of`	Source → Target	The source condition arises as a direct clinical complication along the causal chain of the target.	*Pancreatitis:* Can lead to infected necrosis or persistent organ failure as a complication.	0.45
`shares-risk`	Bidirectional	Conditions share a risk factor or demographic, even if the direct presentation slightly differs.	*Gallstones:* Serves as a shared risk factor for both cholecystitis and acute pancreatitis.	0.30

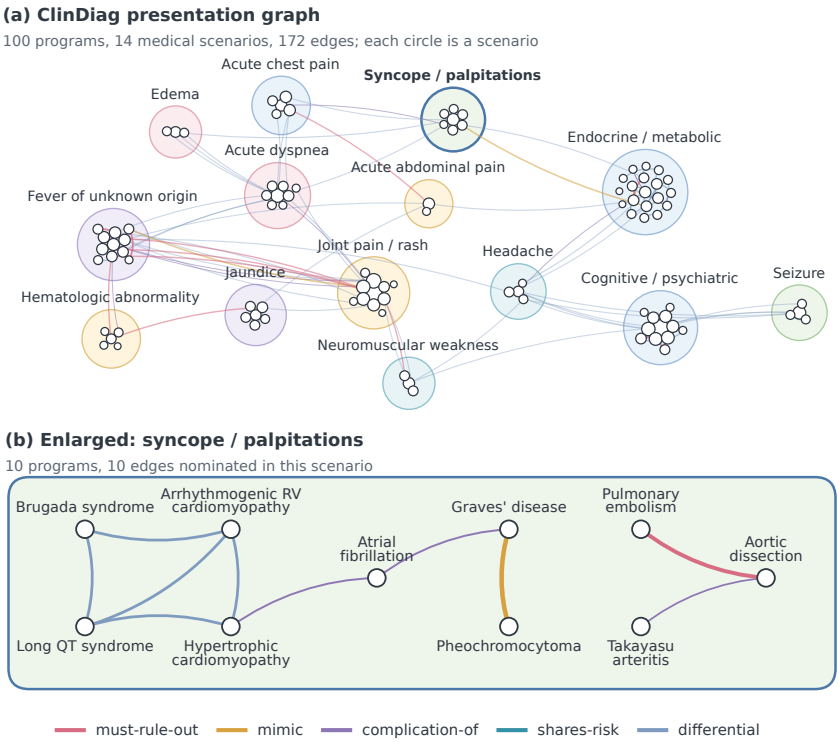

Figure 8: The presentation graph built for ClinDiag. (a) Each circle is a medical scenario. Every edge is nominated within one scenario, but a program in several scenarios is drawn only once, in the scenario where most of its edges lie, so a line between two circles is an edge from a scenario the two programs share. Node size grows with the number of edges. (b) The *syncope / palpitations* scenario enlarged, with all its programs and the edges nominated in it. Edge color gives the relation type (Table 5), and line width grows with the weight w_{ij} of Equation (14).

C How the Full Harness Runs a Consultation

This section details the dynamic execution architecture of the harness introduced in Section 3.6. With the diagnostic library $\mathcal{P} = \{P_d : d \in \mathcal{D}\}$ rigorously constructed and frozen offline, the runtime of the full harness focuses entirely on orchestrating the live consultation. Figure 9 summarizes the consultation loop and its rule-supported and residual exits.

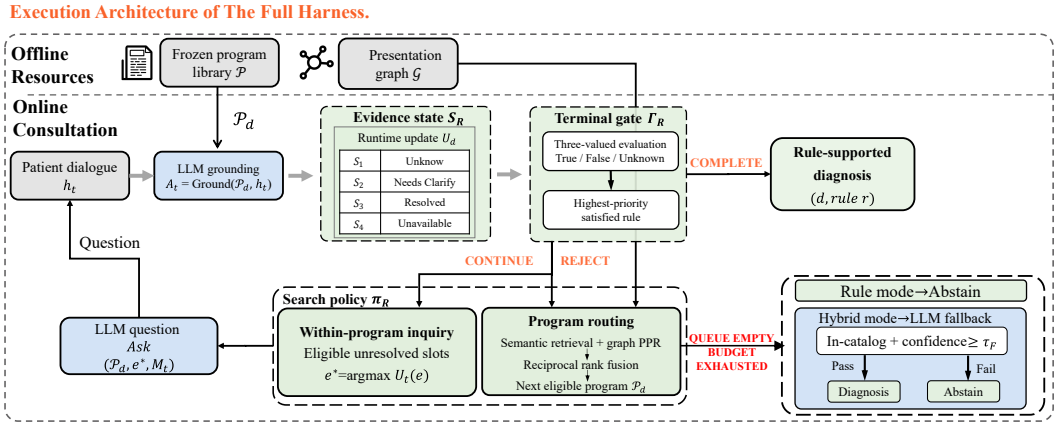

Figure 9: Execution architecture of the full harness. The LLM grounds dialogue into proposed evidence and verbalizes runtime-selected questions; the runtime owns evidence updates ($\mathcal{S}_{\mathrm{R}}$), inquiry and routing (π_{R}), and terminal control (Γ_{R}). COMPLETE returns a rule-supported diagnosis, while CONTINUE selects an eligible inquiry when budget remains. Rejection or exhausted evidence slots trigger program routing and re-grounding, indicated by the matching "Next program" ports. An empty queue or exhausted budget invokes the residual policy: Rule mode abstains; Hybrid mode returns a learned diagnosis only when its catalog and confidence checks pass.

C.1 Which Responsibilities Move to the Runtime

Crucially, our framework formally decouples the diagnostic process into three responsibilities: patient evidence state ($\mathcal{S}$), search policy (π), and terminal control (Γ). Because these modules interact through strictly defined state boundaries rather than entangled model weights, they are highly modular. Each responsibility can be independently assigned to either the LLM's internal reasoning (denoted by subscript L) or the deterministic runtime (denoted by subscript R).

By progressively substituting LLM-driven components with programmatic ones, this modularity allows us to explicitly construct the spectrum of system variants presented in Table 1. For instance, replacing only the state tracking yields the + *Runtime State* variant ($\mathcal{S}_{\mathrm{R}}, \pi_{\mathrm{L}}, \Gamma_{\mathrm{L}}$); further externalizing the terminal gate produces the + Runtime Terminal (Rule) variant ($\mathcal{S}_{\mathrm{R}}, \pi_{\mathrm{L}}, \Gamma_{\mathrm{R}}$); and finally, externalizing the search policy yields the fully externalized endpoint, the full harness ($\mathcal{S}_{\mathrm{R}}, \pi_{\mathrm{R}}, \Gamma_{\mathrm{R}}$). The remainder of this section details the mechanics of this fully externalized endpoint, as synthesized in Algorithm 1.

C.2 Updating the Evidence State ($\mathcal{S}_{\mathrm{R}}$)

During an active consultation, the runtime explicitly maintains the state of the active program P_d. Let $M_t = \{\sigma_t(P_d, e) \mid e \in E_d\}$ be the runtime record at turn t. Each slot resides in one of four states: UNKNOWN (unasked), NEEDSCLARIFY (ambiguous), RESOLVED (clinically supported), or UNAVAILABLE (unobtainable).

When the patient responds to an inquiry, the LLM processes the new dialogue history h_t to propose semantic assertions. The runtime then mechanically updates the slot statuses:

$$A_t^d = \mathrm{Ground}_\theta(P_d, h_t), \qquad M_t \leftarrow \mathcal{U}_d(M_t, A_t^d). \tag{15}$$

Here, the operator $\mathcal{U}_d$ acts as a strict programmatic filter. It verifies that the LLM's proposed values (A_t^d) comply with the predefined types, unit conventions, and logical consistencies of E_d. A supported

response resolves the slot ($\top$ or $\bot$), vague assertions trigger a NEEDSCLARIFY state, and explicitly unobtainable evidence is flagged as UNAVAILABLE. The LLM therefore cannot write the state directly; it only proposes observations, and a proposal that is well-formed but misreads the reply is the remaining failure mode.

C.3 CHOOSING THE NEXT QUESTION ($\pi_{\mathrm{R}}^{\mathrm{ask}}$)

The search policy π_{R} makes two kinds of choice: which slot to ask about within the active program (this subsection), and which program to try next (Section C.4).

If the active program is neither complete nor rejected, the engine must decide what to ask next. Let $C_t(P_d)$ denote the set of eligible, unresolved evidence slots required by pending rules. The runtime selects the optimal slot $e_t^* = \arg\max_{e \in C_t(P_d)} U_t(e)$, where the utility function is formalized as:

$$U_t(e) = w_1 n_t(e) + w_2 n_t^{\mathrm{complete}}(e) - w_3 c(e) - w_4 \alpha_t(e) - w_5 \mathbf{1}[\sigma_t(e) = \textsc{NeedsClarify}] - \epsilon \cdot p(e). \tag{16}$$

This formula mathematically balances diagnostic progress against acquisition cost. Here, $n_t(e)$ and $n_t^{\mathrm{complete}}(e)$ measure the structural yield (how many pending paths or completion rules rely on this slot). Conversely, $c(e)$ is the inherent clinical cost of the test (e.g., imaging carries a higher cost than history-taking), $\alpha_t(e)$ penalizes repeated ask attempts, and $p(e)$ breaks ties using the authored slot priority. By default, empirical weights are set to $(w_1 = 100, w_2 = 30, w_3 = 8, w_4 = 15, w_5 = 8)$, effectively mimicking a clinician's preference for high-yield, low-burden inquiries.

C.4 CHOOSING THE NEXT PROGRAM ($\pi_{\mathrm{R}}^{\mathrm{route}}$)

When an active program is explicitly rejected, the agent must seamlessly pivot. Routing uses the weighted presentation graph $G = (\mathcal{D}, W)$ built offline in Section B.2. Its edges are the clinical relations in Table 5, such as `must-rule-out` and `mimic`, with weights assigned in code.

When routing is invoked, the engine fuses standard semantic retrieval with graph propagation. Using Personalized PageRank (PPR) seeded by a distribution v_t over the current and visited programs, the engine traverses G to yield a graph-based distribution $p_t = \rho v_t + (1-\rho) T^\top p_t$. Finally, the routing score $s_{\mathrm{route},t}(P_d)$ integrates semantic similarity and graph centrality via Reciprocal Rank Fusion (RRF):

$$s_{\mathrm{route},t}(P_d) = \frac{\lambda_{\mathrm{sem}}}{\tau + \mathrm{rank}_{\mathrm{sem}}(d)} + \frac{\lambda_{\mathrm{PPR}}}{\tau + \mathrm{rank}_{\mathrm{PPR}}(d)}. \tag{17}$$

Using empirical fusion weights ($\lambda_{\mathrm{sem}} = 0.65, \lambda_{\mathrm{PPR}} = 0.35, \tau = 10$), this mechanism surfaces the most logical next hypotheses. Crucially, this routing strictly dictates the sequence of hypotheses tested; it cannot arbitrarily authorize a diagnosis.

C.5 TERMINAL GATE AND RESIDUAL HANDLING (Γ_{R}, η)

After every state update, the runtime evaluates the active program using the three-valued logic gate $\Gamma_{\mathrm{R}}(M_t, P_d)$ defined in Section 3. This gate securely maps the current evidence state to one of three deterministic control actions:

$$\Gamma_{\mathrm{R}}(M_t, P_d) = \begin{cases} \textsc{Complete}(d, r^*), & \text{highest priority rule } r^* \text{ satisfied, action is Complete,} \\ \textsc{Reject}(P_d), & \text{highest priority rule } r^* \text{ satisfied, action is Reject,} \\ \textsc{Continue}, & \text{no terminal rule is decisively satisfied.} \end{cases} \tag{18}$$

Residual handling (η). In clinical practice, a consultation may exhaust available tests without reaching a definitive rule-supported conclusion. Our framework manages this edge case via a residual policy η. Under the conservative **Rule mode** ($\eta = \mathrm{Rule}$), the system guarantees strict traceability by returning an ABSTAIN result, preventing unsupported guessing.

Conversely, the **Hybrid mode** ($\eta = \mathrm{LLM}$) provides a best-effort fallback. The LLM performs residual reasoning F_θ over the explicit clinical catalog $\mathcal{K}_t$ gathered during the interaction:

$$(\hat{d}, \hat{c}) = F_\theta(h_t, M_t, \mathcal{K}_t) \tag{19}$$

If the LLM's diagnostic confidence $\hat{c}$ surpasses a threshold τ_F, it returns $\hat{d}$ with a strictly marked *learned provenance*, explicitly decoupling transparent, rule-backed diagnoses from black-box inferences.

C.6 Consultation Algorithm

Algorithm 1 orchestrates the modules above into a cohesive loop. It demonstrates how LLM operations (Grounding and Asking) are firmly encapsulated within the rigid control structures of the deterministic harness (State Update, Terminal Gating, and Routing).

Algorithm 1 The full harness: Integrating State, Search, and Terminal Control

Require: Frozen library $\mathcal{P}$ and Graph G; patient interface $\mathcal{I}$; initial history h_0; search budget; residual policy η.
Ensure: Diagnosis y, provenance ζ, supporting rule r (if applicable).

$h \leftarrow h_0$; initialize explicit state M with all slot statuses as UNKNOWN
$Q \leftarrow \mathrm{Retrieve}(h_0, \mathcal{P})$ ▷ Initial semantic retrieval
while $Q \neq \varnothing$ and search budget remains **do**
  $P_d \leftarrow \arg\max_{P \in Q} s_{\mathrm{route}}(P)$ ▷ Route to the most likely program
  loop
    $A \leftarrow \mathrm{Ground}_\theta(P_d, h)$ ▷ LLM perceives new evidence
    $M \leftarrow \mathcal{U}_d(M, A)$ ▷ Runtime rigorously updates state
    $g \leftarrow \Gamma_{\mathrm{R}}(M, P_d)$ ▷ Evaluate three-valued logical rules
    if $g = \textsc{Complete}(d, r)$ **then**
      return (d, rule, r)
    end if
    $C(P_d) \leftarrow$ eligible unresolved evidence slots in P_d
    if $g = \textsc{Reject}(P_d) \ \vee \ C(P_d) = \varnothing$ **then**
      Exclude P_d and refresh eligible queue Q using Graph G
      break ▷ Pivot to a new diagnostic program
    end if
    if no budget remains for another inquiry **then**
      $Q \leftarrow \varnothing$; **break**
    end if
    $e^* \leftarrow \arg\max_{e \in C(P_d)} U_t(e)$ ▷ Select optimal clinical question
    $q \leftarrow \mathrm{Ask}_\theta(P_d, e^*, M)$; $a \leftarrow \mathcal{I}(q)$
    $h \leftarrow h \oplus (q, a)$; update remaining budget
  end loop
end while
if $\eta = \mathrm{LLM}$ **then**
  $\mathcal{K} \leftarrow$ candidate catalog with observed program outcomes
  $(\hat{d}, \hat{c}) \leftarrow F_\theta(h, M, \mathcal{K})$ ▷ LLM residual reasoning
  if $\hat{d} \in \mathcal{D} \ \wedge \ \hat{c} \geq \tau_F$ **then**
    return $(\hat{d}, \mathrm{learned}, \varnothing)$
  end if
end if
return $(\varnothing, \mathrm{abstain}, \varnothing)$

D Experimental Setup

D.1 Benchmarks

To comprehensively evaluate our proposed framework, we utilize three distinct clinical datasets covering different medical domains. For every diagnostic category within these benchmarks, we constructed a corresponding executable diagnostic program to populate the frozen library $\mathcal{P}$.

All baseline methods and our system variants are evaluated on the exact same patient cases within each benchmark. The datasets are detailed as follows:

- **Abdominal:** This benchmark focuses on acute abdominal pain. We use 60 cases, which are mapped to 4 distinct diagnostic categories (programs). Cases are drawn from MIMIC-CDM (Hager et al., 2024), which is built on MIMIC-IV (Johnson et al., 2023); the four categories are appendicitis, cholecystitis, diverticulitis, and pancreatitis.

- **Cardiovascular:** This benchmark covers acute cardiovascular conditions. We evaluate 60 cases, mapped to 8 specific diagnostic categories (programs). Cases are de-identified inpatient records from a cardiovascular hospital.
- **ClinDiag:** Serving as a library-scale test, this benchmark comprises 100 clinical cases, one per disease, drawn from the ClinDiag benchmark (Chen et al., 2026), which is compiled from published clinical case reports.

The Abdominal and Cardiovascular programs were compiled from published clinical guidelines and reviewed by a clinician before the library was frozen. The ClinDiag programs were compiled from model knowledge without clinical guidelines (Section B.1).

D.2 Simulator, Budgets, and Scoring

Patient simulator. To simulate realistic clinical interactions, a patient simulator is initialized with the complete, hidden patient record. Crucially, it reveals information *only* in response to the diagnostic agent's specific questions. Across all main experiments, DeepSeek-V4-Flash is fixed as the patient simulator engine. The diagnostic agent and the patient alternate turns until the agent authorizes a diagnosis, explicitly abstains, or exhausts the interaction budget.

Budgets. To ensure fair comparison and simulate realistic time constraints, all baseline methods and harness variants operate under a strict interaction budget. The system allows a maximum of 80 shared dialogue messages per case. For the full harness, the search space is bounded to at most 5 program visits (hypothesis switches) per consultation.

Retrieval and execution. At the start of the consultation, the initial diagnostic queue is populated with the top 5 candidate programs via semantic retrieval based on the patient's opening complaint. From there, the routing, terminal control, and fallback mechanisms strictly follow the logic detailed in Appendix C. The presentation graph serves purely as a search prior to guide the routing policy.

Scoring. For baseline LLM methods, their free-text diagnostic outputs are evaluated against the ground-truth reference using a fixed GPT-4o-mini judge. Full Harness (Rule) and Full Harness (Hybrid) output a registered program, so they are scored by program identifier; Section E.2.3 shows that scoring them with the same judge changes little. Table 6 collects these fixed settings.

Table 6: **Interactive evaluation parameters.**

Parameter	Value
Patient simulator	DeepSeek-V4-Flash
Dialogue budget	80 messages
Program visits	5
Initial queue	5
Free-text judge	GPT-4o-mini

D.3 Baselines

We separate the information supplied to each method from its control allocation in Table 7. The last four columns use exactly the responsibilities and ordering of Table 1. Comparison results that do not appear in the main text are reported later: Table 8 places LLM + Guideline Retrieval beside LLM + Candidate Set, Program Prompt, and Full Harness (Hybrid) under partial observation on Abdominal and Cardiovascular; Table 9 reports the static Complete Record and Complete Record + Candidate Set references and MDAgents against Full Harness (Hybrid) across diagnostic backbones.

D.4 Metrics

Outcome accounting. Each case receives exactly one terminal state: rule-correct (RC), rule-wrong (RW), fallback-correct (FC), fallback-wrong (FW), abstained (A), with

$$RC + RW + FC + FW + A = N. \tag{20}$$

Table 7: **Information access and control across all evaluated methods.** The four control columns follow the main-text allocation table. Blue cells mark runtime responsibilities. Full-record methods are static references; the other methods acquire patient information through dialogue.

Method	Information supplied			Control allocation			
	Patient record	Disease list	Procedural knowledge	Evidence State $\mathcal{S}$	Terminal Γ	Search Policy π	Residual Handling η
Complete-record references							
Complete Record	Full	No	–	–	–	–	–
Complete Record + Candidate Set	Full	All	–	–	–	–	–
MDAgents (Kim et al., 2024)	Full	All	–	–	–	–	–
Interactive language-model baselines							
Direct LLM	Dialogue	No	–	LLM	LLM	LLM	LLM
LLM + Candidate Set	Dialogue	All	–	LLM	LLM	LLM	LLM
LLM + Guideline Retrieval	Dialogue	Implicit	Guideline text	LLM	LLM	LLM	LLM
Same-program control variants							
Program Prompt	Dialogue	Active	Program text	LLM	LLM	LLM	LLM
+ Runtime State	Dialogue	Active	Program text	Runtime	LLM	LLM	LLM
+ Runtime Terminal (Rule)	Dialogue	Active	Program text	Runtime	Runtime	LLM	Rules
+ Runtime Terminal (Hybrid)	Dialogue	Active	Program text	Runtime	Runtime	LLM	LLM
Full Harness (Rule)	Dialogue	Active	Executable program	Runtime	Runtime	Runtime	Rules
Full Harness (Hybrid)	Dialogue	Active	Executable program	Runtime	Runtime	Runtime	LLM

Reading the table. "–" denotes absent procedural knowledge or inapplicable interactive control; static outputs are LLM decisions (multi-agent for MDAgents). *Disease list* records which candidate disease names can enter an LLM context. No: none. All: the complete benchmark list at once. Implicit: names occur inside retrieved guideline passages, which come from one source document per disease. Active: the name and scope of the program currently loaded, plus the names of programs already left; every program-based method sees this.

For hybrid, $C = RC + FC$ and $W = RW + FW$; for rules-only, $C = RC$ and $W = RW$. The main-text correctness, wrong-answer, and abstention rates are

$$\text{Corr.} = \frac{C}{N}, \qquad \text{Wrong} = \frac{W}{N}, \qquad \text{Abst.} = \frac{A}{N}. \tag{21}$$

Supplementary analyses use answered-case coverage $\text{Coverage} = (C + W)/N$ and risk $\text{Risk} = W/(C + W)$. Retry rows are de-duplicated by patient identifier, retaining the final record, so every denominator equals the test-set size. Infrastructure errors remain in the denominator.

Uncertainty. Beyond the run-level standard deviations in the tables, we estimate patient-level 95% confidence intervals by paired bootstrap, resampling patients with their three runs, and test each difference with a paired permutation test. Under these intervals, the full harness outperforms Program Prompt in all six settings, with every interval well above zero. Among the individual control components, adding explicit state alone has no significant effect in any setting; terminal control is significant in five of six; runtime search is significant in four of six, with its largest gains on ClinDiag.

Jointly answered cases. For methods a and b in run r, let $A_{a,r}$ and $A_{b,r}$ be their answered patient sets. Their paired cohort is $J_{ab,r} = A_{a,r} \cap A_{b,r}$. Both methods' error rates use this same denominator:

$$E_{a|ab,r} = \frac{\sum_{i \in J_{ab,r}} \mathbf{1}[\hat{d}_{a,r,i} \text{ is wrong}]}{|J_{ab,r}|}, \qquad E_{b|ab,r} = \frac{\sum_{i \in J_{ab,r}} \mathbf{1}[\hat{d}_{b,r,i} \text{ is wrong}]}{|J_{ab,r}|}. \tag{22}$$

Here $\hat{d}_{a,r,i}$ is method a's diagnosis for patient i. Each pair uses identical cases within a run. Intersections can differ across pairs.

For the display in Figures 2 and 11, each lower-triangle cell is the unweighted mean of $\Delta_{ab,d} = \bar{E}_{a,d} - \bar{E}_{b,d}$ over Abdominal, Cardiovascular, and ClinDiag, where $\bar{E}_{a,d}$ and $\bar{E}_{b,d}$ are the run-mean error rates on that pair's jointly answered patients in benchmark d. Blue marks a negative mean (the row method has lower error). These conditional error rates differ from the full-cohort wrong-answer rates in Table 2 because they use only jointly answered cases.

E Additional Results

These results add to the main text in five ways: further baselines (Section E.1), robustness checks (Section E.2), how the full harness reaches its answers (Section E.3), interaction cost (Section E.4), and one consultation followed step by step (Section E.5).

E.1 Additional Baselines

These comparisons use baselines defined in Section D.3 but omitted from the main result table, and extend the LLM + Candidate Set versus Full Harness (Hybrid) comparison to all six diagnostic models.

E.1.1 Guideline Retrieval

As shown in Table 8, across two benchmarks containing guideline corpora, we compared the direct retrieval of guideline text (LLM + Guideline Retrieval) with program-constrained approaches (Program Prompt and Full Harness (Hybrid)). Furthermore, because the corpus is structured by mapping each disease to a single original guideline, the retrieved text snippets typically list candidate diseases; therefore, we also included LLM + Candidate Set in our comparison.

LLM + Guideline Retrieval performs much like LLM + Candidate Set (within 3.3 points in three of four cells), suggesting that its benefit comes mainly from the disease names in the retrieved text. Compared to the programmatic Program Prompt, the text-based LLM + Guideline Retrieval and LLM + Candidate Set perform better on DeepSeek-V4-Flash, while the comparison is mixed on Qwen3-8B. Furthermore, we observed that under explicit programmatic constraints, the model's refusal rate is much higher (e.g., 40.6% with Flash on Abdomen , compared to only 1.1% for LLM + Guideline Retrieval). Although this characteristic effectively reduces the error rate, it simultaneously lowers overall accuracy. This aligns with our expectations: when subjected to standardized constraints, the model tends to prioritize information gathering and behaves more cautiously. Ultimately, the Full Harness (Hybrid) approach provides better harness control and achieves the highest correctness in all four cells.

Table 8: **Performance comparison of guideline retrieval and compiled programs.**

Method	Abdominal			Cardiovascular		
	Corr.↑	Wrong↓	Abst.	Corr.↑	Wrong↓	Abst.
DeepSeek-V4-Flash						
LLM + Candidate Set	75.6 ±3.5	21.7 ±4.4	2.8 ±1.0	88.3 ±0.0	11.1 ±1.0	0.6 ±1.0
LLM + Guideline Retrieval	73.3 ±5.8	25.6 ±3.8	1.1 ±1.9	88.9 ±3.5	10.6 ±4.2	0.6 ±1.0
Program Prompt	55.0 ±3.3	4.4 ±3.5	40.6 ±1.0	53.3 ±5.0	0.6 ±1.0	46.1 ±5.9
Full Harness (Hybrid)	83.3 ±3.3	1.1 ±1.0	15.6 ±2.5	92.2 ±2.5	1.7 ±0.0	6.1 ±2.5
Qwen3-8B						
LLM + Candidate Set	52.2 ±3.5	21.7 ±0.0	26.1 ±3.5	48.9 ±4.2	32.8 ±8.4	18.3 ±4.4
LLM + Guideline Retrieval	48.9 ±2.5	31.7 ±5.0	19.4 ±3.8	60.0 ±2.9	25.6 ±3.5	14.4 ±1.0
Program Prompt	48.9 ±1.0	16.7 ±5.0	34.4 ±5.1	51.1 ±1.0	26.7 ±2.9	22.2 ±3.5
Full Harness (Hybrid)	77.8 ±3.5	8.3 ±5.8	13.9 ±2.5	88.3 ±2.9	3.3 ±0.0	8.3 ±2.9

E.1.2 Complete-Record References and MDAgents

Table 9 asks how much it costs to ask instead of read. Full Harness (Hybrid) must ask for every finding, while three static references read the complete record at once: Complete Record, Complete Record + Candidate Set, and MDAgents (which also receives the candidate list). With the candidate list, these references set a ceiling.

On Abdominal and Cardiovascular, Full Harness (Hybrid) recovers at least 87% of that ceiling for every backbone from Qwen3-8B up; on ClinDiag it recovers 62–81%. With the candidate list, the static references are usually more accurate than Full Harness (Hybrid), most clearly on ClinDiag, where Complete Record + Candidate Set leads by 11 to 33 points for every backbone except Qwen3-1.7B. Full Harness (Hybrid) usually makes fewer wrong diagnoses instead: its wrong-answer rate is at or below that of Complete Record + Candidate Set in 14 of 18 backbone–benchmark cells, because it abstains when evidence is missing. For the smallest model, Qwen3-1.7B, Full Harness (Hybrid) is the most accurate method on Abdominal and ClinDiag, but its wrong-answer rate stays high (25–49%).

Table 9: **Impact of interactive harness vs. static complete records.** Comparison between idealized static diagnosis with full information access and sequential interactive diagnosis under partial observation.

Method	Abdominal			Cardiovascular			ClinDiag		
	Corr.	Wrong	Abst.	Corr.	Wrong	Abst.	Corr.	Wrong	Abst.
Qwen3-1.7B									
Complete Record	25.0 ±0.0	61.1 ±1.0	13.9 ±1.0	76.7 ±0.0	23.3 ±0.0	0.0 ±0.0	39.7 ±0.6	30.3 ±0.6	30.0 ±0.0
+ Candidate Set	63.3 ±0.0	36.7 ±0.0	0.0 ±0.0	53.3 ±0.0	46.7 ±0.0	0.0 ±0.0	25.3 ±0.6	29.7 ±0.6	45.0 ±0.0
MDAgents	58.3 ±1.7	41.7 ±1.7	0.0 ±0.0	61.7 ±2.9	38.3 ±2.9	0.0 ±0.0	35.0 ±1.0	62.0 ±1.7	3.0 ±1.0
Full Harness (Hybrid)	73.3 ±2.9	26.7 ±2.9	0.0 ±0.0	65.0 ±3.3	25.0 ±2.9	10.0 ±1.7	47.0 ±1.7	49.0 ±2.0	4.0 ±2.6
Qwen3-8B									
Complete Record	45.0 ±0.0	5.0 ±0.0	50.0 ±0.0	86.7 ±0.0	5.0 ±0.0	8.3 ±0.0	26.0 ±0.0	8.0 ±0.0	66.0 ±0.0
+ Candidate Set	85.0 ±0.0	13.3 ±0.0	1.7 ±0.0	96.1 ±1.0	3.9 ±1.0	0.0 ±0.0	61.7 ±0.6	34.0 ±1.0	4.3 ±0.6
MDAgents	83.3 ±1.7	13.3 ±1.7	0.0 ±0.0	95.0 ±1.7	5.0 ±1.7	0.0 ±0.0	61.0 ±1.0	38.0 ±0.0	1.0 ±1.0
Full Harness (Hybrid)	77.8 ±3.5	8.3 ±5.8	13.9 ±2.5	88.3 ±2.9	3.3 ±0.0	8.3 ±2.9	46.0 ±5.6	12.0 ±2.0	42.0 ±6.6
Qwen3-14B									
Complete Record	45.0 ±1.7	5.0 ±0.0	50.0 ±1.7	80.0 ±0.0	17.8 ±1.0	2.2 ±1.0	56.0 ±0.0	12.0 ±0.0	32.0 ±0.0
+ Candidate Set	78.3 ±0.0	3.3 ±0.0	18.3 ±0.0	93.3 ±0.0	6.7 ±0.0	0.0 ±0.0	57.0 ±0.0	14.0 ±0.0	29.0 ±0.0
MDAgents	81.7 ±0.0	8.3 ±2.9	10.0 ±2.9	91.7 ±1.7	5.0 ±1.7	3.3 ±1.7	69.0 ±1.0	20.0 ±1.0	11.0 ±1.7
Full Harness (Hybrid)	75.0 ±1.7	3.3 ±2.9	21.7 ±1.7	95.0 ±1.7	0.0 ±0.0	5.0 ±1.7	46.0 ±2.0	12.0 ±2.6	42.0 ±1.7
Qwen3-32B									
Complete Record	64.4 ±1.9	9.4 ±2.5	26.1 ±1.0	85.6 ±1.0	9.4 ±1.0	5.0 ±0.0	56.3 ±0.6	13.0 ±0.0	30.7 ±0.6
+ Candidate Set	85.6 ±1.0	3.3 ±0.0	11.1 ±1.0	97.8 ±1.0	2.2 ±1.0	0.0 ±0.0	73.0 ±0.0	7.7 ±0.6	19.3 ±0.6
MDAgents	88.3 ±0.0	5.0 ±1.7	6.7 ±1.7	96.7 ±1.7	3.3 ±1.7	0.0 ±0.0	80.0 ±1.0	18.0 ±1.0	2.0 ±0.0
Full Harness (Hybrid)	75.0 ±1.7	3.3 ±1.7	21.7 ±1.7	85.0 ±1.7	0.0 ±0.0	15.0 ±1.7	45.0 ±1.0	5.0 ±1.0	50.0 ±1.0
DeepSeek-V4-Flash									
Complete Record	82.8 ±1.0	14.4 ±1.0	2.8 ±1.0	93.3 ±4.4	6.1 ±3.5	0.6 ±1.0	77.0 ±1.0	16.7 ±0.6	6.3 ±0.6
+ Candidate Set	86.1 ±2.5	5.0 ±1.7	8.9 ±2.5	98.3 ±0.0	1.1 ±1.0	0.6 ±1.0	89.3 ±2.1	7.0 ±1.0	3.7 ±1.2
MDAgents	91.7 ±1.7	5.0 ±1.7	3.3 ±0.0	98.3 ±1.7	1.7 ±1.7	0.0 ±0.0	87.0 ±1.0	10.0 ±0.0	3.0 ±1.0
Full Harness (Hybrid)	83.3 ±3.3	1.1 ±1.0	15.6 ±2.5	92.2 ±2.5	1.7 ±0.0	6.1 ±2.5	67.7 ±4.2	7.7 ±1.5	24.7 ±2.9
DeepSeek-V4-Pro									
Complete Record	83.3 ±1.7	12.2 ±2.5	4.4 ±3.8	90.6 ±1.0	7.8 ±1.0	1.7 ±0.0	72.3 ±4.0	22.0 ±4.0	5.7 ±0.6
+ Candidate Set	88.3 ±2.9	3.3 ±1.7	8.3 ±1.7	98.3 ±0.0	0.6 ±1.0	1.1 ±1.0	89.7 ±1.2	5.7 ±0.6	4.7 ±0.6
MDAgents	91.7 ±1.7	5.0 ±1.7	3.3 ±1.7	96.7 ±1.7	3.3 ±1.7	0.0 ±0.0	88.0 ±1.7	8.0 ±1.0	4.0 ±1.0
Full Harness (Hybrid)	83.3 ±0.0	3.3 ±2.9	13.3 ±2.9	95.0 ±2.9	0.0 ±0.0	5.0 ±2.9	57.0 ±2.0	6.0 ±1.7	37.0 ±1.0

E.1.3 All Six Diagnostic Models

Table 10 gives the full measured matrix behind the six-backbone comparison in Figure 4a–c, for the Qwen3 family and DeepSeek-V4. The flattening is sharpest on ClinDiag: from Qwen3-1.7B to Qwen3-32B, Full Harness (Hybrid) stays at 45–47% while LLM + Candidate Set climbs from 5% to 41%. There the ceiling is set less by the model than by the library, since rule closure stays near 18% for both backbones we trace (Figure 13).

Table 10: **Full results across Qwen3 and DeepSeek diagnostic backbones.**

Diagnostic backbone	LLM + Candidate Set Corr.	Wrong	Abst.	Full Harness (Hybrid) Corr.	Wrong	Abst.
Abdominal						
Qwen3-1.7B	21.7	28.3	50.0	73.3	26.7	0.0
Qwen3-8B	52.2	21.7	26.1	77.8	8.3	13.9
Qwen3-14B	58.3	30.0	11.7	75.0	3.3	21.7
Qwen3-32B	75.0	18.3	6.7	75.0	3.3	21.7
V4-Flash	75.6	21.7	2.8	83.3	1.1	15.6
V4-Pro	83.3	16.7	0.0	83.3	3.3	13.3
Cardiovascular						
Qwen3-1.7B	21.7	36.7	41.7	65.0	25.0	10.0
Qwen3-8B	48.9	32.8	18.3	88.3	3.3	8.3
Qwen3-14B	66.7	26.7	6.7	95.0	0.0	5.0
Qwen3-32B	81.7	18.3	0.0	85.0	0.0	15.0
V4-Flash	88.3	11.1	0.6	92.2	1.7	6.1
V4-Pro	91.7	8.3	0.0	95.0	0.0	5.0
ClinDiag						
Qwen3-1.7B	5.0	18.0	77.0	47.0	49.0	4.0
Qwen3-8B	12.3	56.3	31.3	46.0	12.0	42.0
Qwen3-14B	18.0	56.0	26.0	46.0	12.0	42.0
Qwen3-32B	41.0	46.0	13.0	45.0	5.0	50.0
V4-Flash	52.3	46.0	1.7	67.7	7.7	24.7
V4-Pro	48.0	44.0	8.0	57.0	6.0	37.0

E.2 Robustness Checks

E.2.1 Replication with DeepSeek-V4-Flash

We repeat the analyses of Figures 2 and 3 with DeepSeek-V4-Flash as the diagnostic backbone; its interaction cost is in Section E.4. All quantities use the same three repetitions, patient cases, and outcome definitions as the Flash block in Table 2. + Runtime Terminal (Hybrid) and Full Harness (Hybrid) use Hybrid outputs, including learned fallback.

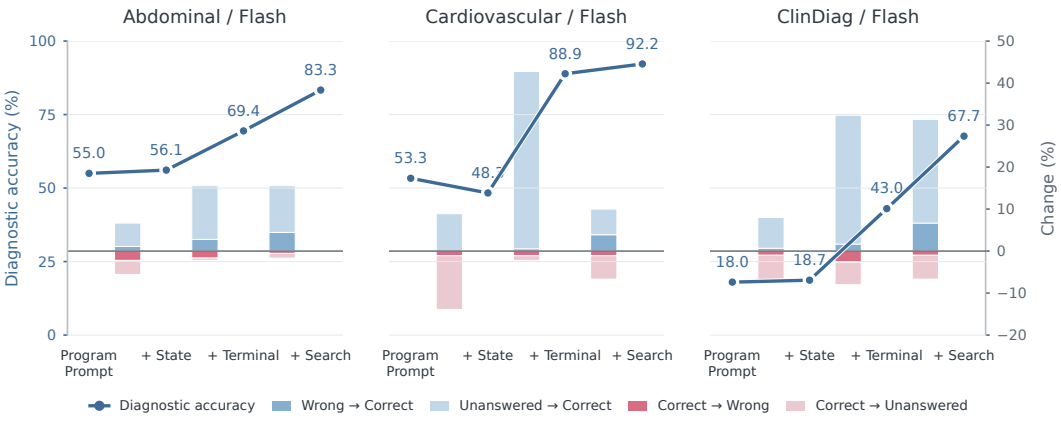

Figure 10: **Flash: progressive externalization of control.** Same layout as Figure 3. Lines (left axis) give correctness as evidence state, terminal control, and search move to the runtime; bars (right axis) give the share of patients whose outcome changes between adjacent configurations.

Stage-by-stage transitions. The + Runtime State variant changes correctness by $+1.1$, -5.0, and $+0.7$ points on Abdominal, Cardiovascular, and ClinDiag, respectively. Thus the inconsistent benefit of state as context is also present with Flash. Adding + Runtime Terminal (Hybrid) raises correctness substantially on Cardiovascular and ClinDiag; the full harness improves it further on all three benchmarks.

The gains involve different patient transitions. On Cardiovascular, cases moving from + Runtime State abstention to a correct + Runtime Terminal (Hybrid) answer account for 42.2% of the full cohort. Cases moving from a correct answer to a wrong answer or abstention account for 2.2% of the full cohort. On ClinDiag, moving from + Runtime Terminal (Hybrid) to the full harness gains 31.3% newly correct cases and loses 6.7%, for a net gain of 24.7 points. On ClinDiag, Flash rises from 18.0% under Program Prompt to 43.0% once terminal control is externalized and to 67.7% under the full harness, so the intermediate stages do not collapse. Qwen3-8B, by contrast, stays below 22% until search is externalized (Section 4.3) Flash can already cover more of this large-library search from its own medical knowledge. Qwen3-8B depends on the runtime to supply the procedural capacity it does not have.

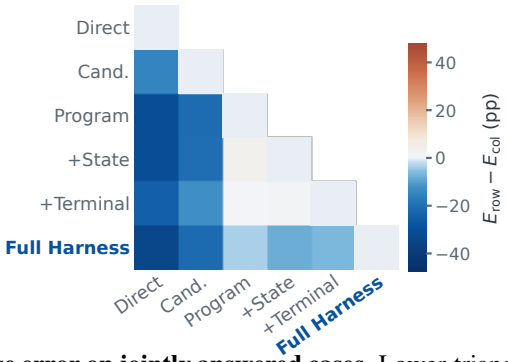

Figure 11: **Flash: pairwise error on jointly answered cases.** Lower-triangle mean $\Delta = E_{\text{row}} - E_{\text{col}}$ across Abdominal, Cardiovascular, and ClinDiag. Darker blue: the row method has lower error.

Errors on jointly answered cases. On Flash, the Full Harness (Hybrid) row remains blue against Direct LLM, Candidate Set, and Program Prompt after averaging the three benchmarks. The Cardiovascular Program Prompt pair is the close exception in the per-benchmark view (Program slightly lower error). Its results are similar to those of Qwen3-8B.

E.2.2 Is the Lower Error Explained by Answering Fewer Cases?

The paired analysis in Figure 2 compares identical patients within each pair. Here we ask a different question: if a baseline outputs only its most confident answers, can it match the harness's low answered risk?

Each panel of Figure 12 evaluates the six non-Rule methods at one common answer count k. We set k to the smallest number of answers produced by any of these methods in any of the three runs, and use the same k in all three runs; coverage is k/N. Baselines keep their k highest-confidence answers. + Runtime Terminal (Hybrid) is ranked by its recorded score (1.0 for a rule commitment, the fallback's self-reported confidence otherwise). Full Harness (Hybrid) has no diagnostic ranking score and is therefore subsampled uniformly, which leaves its expected risk equal to its native answered risk. Selection never uses correctness. Answer counts therefore match across methods, but the selected patients may differ, so this compares recorded selection policies rather than same-case performance. Rule-only rows are excluded; including them would lower k to their rule-closure counts.

For Qwen3-8B, confidence-based selection does not close the gap on Abdominal and Cardiovascular. At matched coverage, Full Harness (Hybrid) has expected answered risks of 9.6% and 3.6%, while confidence-filtered Program Prompt remains at 20.8% and 26.6%. + Runtime Terminal (Hybrid) is close to Full Harness (Hybrid) on Abdominal (9.0% versus 9.6%) but higher on Cardiovascular (6.3% versus 3.6%).

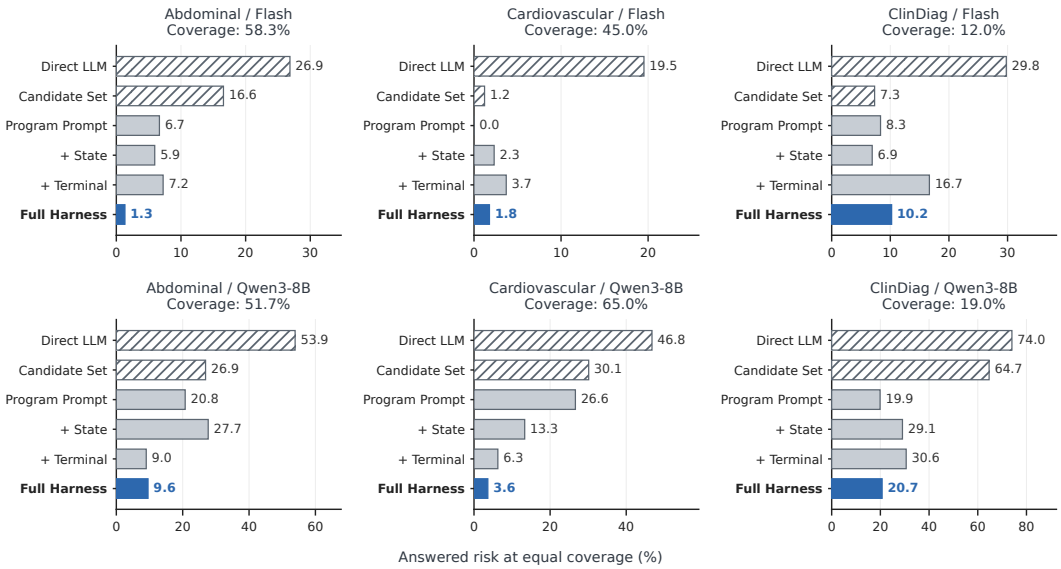

Figure 12: **Answered risk at equal coverage.** Lower is better.

A model's own confidence carries signal: filtering never raises the risk of the four ungated baselines. It does not carry what rule evidence carries. Where baselines catch up, as with Flash on ClinDiag, coverage has shrunk to 12% and each model answers only its most obvious cases.

E.2.3 Does the Scoring Path Favor the Full Harness?

Baselines are scored by the free-text judge, whereas Full Harness (Rule) and Full Harness (Hybrid) are scored by program identifier. To check whether this difference favors the harness, we score every harness run in Table 2 a second time through the baseline path: the output program is replaced by its disease name and passed to the same GPT-4o-mini judge, with the same prompt and the same reference diagnosis. Abstentions are not sent to the judge. Table 11 compares the two scorings.

Table 11: **Program-identifier versus judge scoring of the full harness.** Mean over three runs (%). *ID* is the scoring used in Table 2; *Judge* is the free-text judge used for baselines. *Agree* counts answered cases on which the two scorings give the same verdict, pooled over the three runs.

	Full Harness (Rule)				Full Harness (Hybrid)				
	Corr.		Wrong		Corr.		Wrong		
Benchmark	ID	Judge	ID	Judge	ID	Judge	ID	Judge	**Agree**
Qwen3-8B									
Abdominal	68.9	66.7	7.8	10.0	77.8	75.6	8.3	10.6	151/155
Cardiovascular	40.0	40.0	0.0	0.0	88.3	86.7	3.3	5.0	162/165
ClinDiag	17.7	17.7	1.3	1.3	46.0	46.0	12.0	12.0	174/174
DeepSeek-V4-Flash									
Abdominal	61.1	59.4	1.1	2.8	83.3	81.7	1.1	2.8	149/152
Cardiovascular	30.6	30.6	0.6	0.6	92.2	90.6	1.7	3.3	166/169
ClinDiag	17.7	17.7	0.7	0.7	67.7	67.7	7.7	7.7	226/226

The two scoring methods agree on 1,028 of 1,041 answered cases (98.8%), including every ClinDiag case. No correctness rate shifts by more than 2.2 points—a margin far smaller than the gaps over the free-text baselines shown in Table 2. Therefore, the discrepancy between the two scoring methods is negligible.

E.2.4 How Often Is Evidence Grounded Incorrectly?

While the runtime evaluates rules based on slot values, the values themselves are populated by the diagnostic model, which parses each patient reply and extracts the relevant findings (Ground_θ in Section C). Consequently, the runtime is completely oblivious to any misinterpretations made by the model. To address this, we evaluate the grounding accuracy directly. For every rule-supported diagnosis in Table 2, we identify the slots utilized by the triggered rule, and employ DeepSeek-V4-Pro to annotate the dialogue trajectories from the main experiment to assess the grounding correctness.

Table 12: **Evidence grounding on rule-supported diagnoses.** Three runs pooled. *Agree*: the recorded value matches the reference label. *Wrong fact*: the recorded value is contradicted by, or absent from, the patient reply. *Traced*: wrong rule-supported diagnoses that are no longer made once the misread slots are corrected.

Backbone	Benchmark	Slots	Agree (%)	Wrong fact (%)	Traced
Qwen3-8B	Abdominal	362	76.8	20.7	12/14
	Cardiovascular	336	63.7	32.4	–
	ClinDiag	172	75.0	23.3	3/4
DeepSeek-V4-Flash	Abdominal	263	85.9	11.8	0/2
	Cardiovascular	214	77.1	7.9	0/1
	ClinDiag	147	82.3	15.6	1/2

As shown in Table 12, grounding errors do occur and fall with model capability: 26% of slots carry a wrong fact under Qwen3-8B, compared to 11% under Flash. Because the harness executes clinical logic deterministically, an incorrect rule-supported diagnosis can be traced to the slots that produced it. As Table 12 shows, once the misread slots are manually corrected, 15 of the 18 incorrect rule-supported diagnoses under Qwen3-8B are resolved. This traceability isolates the impact of semantic processing from sequential control. When the LLM acts as the semantic interface of a deterministic harness, its grounding accuracy remains a key factor in the reliability of the whole system.

E.2.5 Sensitivity to the Patient Simulator

Prior to the experiments, we evaluated multiple models for the patient role-playing task. We screened the simulators based on their fidelity in reconstructing real and complete patient records. For each candidate model, we sampled 200 of its responses and utilized both GPT-6-luna and human annotators to verify whether there were any fabricated, contradicted, or omitted facts. The results in Table 13 show that, overall, DeepSeek-V4-Flash achieved the highest fidelity with the fewest errors of all types; therefore, we selected it as our patient simulator. Additionally, we recorded the diagnostic accuracy (Corr.) and wrong answer rate (Wrong) on Abdominal using Qwen3-8B as the diagnostic backbone across these patient models. We found that fidelity and diagnostic scores do not always change synchronously: for instance, the diagnostic accuracy associated with Qwen3-32B is significantly higher than that of Qwen3-8B, yet the fidelity of their responses is roughly equivalent. While inherent errors do exist under this simulated patient setup, human evaluation demonstrates that the performance of frontier models is generally sufficient for our current experimental needs.

Table 13: **Sensitivity to the patient simulator.** All entries are percentages. Rule coverage counts all rule-supported answers; rule precision is correctness among those answers. Faithful denotes the proportion of 200 responses sampled from a given simulator that are evaluated as free from fabricated, contradicted, or omitted facts.

Patient simulator	Corr.	Wrong	Rule coverage	Rule precision	Faithful	Faithful (human)
Qwen3-8B	60.0	6.7	58.3	88.6	60.5	79.0
	65.0	5.0	55.0	90.9	67.0	74.0
	63.3	5.0	56.7	91.2	63.5	76.5
Qwen3-32B	78.3	1.7	66.7	97.5	68.0	81.0
DeepSeek-V4-Pro	80.0	3.3	71.7	97.7	84.0	98.0
DeepSeek-V4-Flash	81.7	1.7	76.7	97.8	87.0	98.5

E.3 Inside the Full Harness

E.3.1 Rule-Supported versus Fallback Answers

Table 14 details the decision provenance for the Full Harness (Hybrid) approach, corroborating the analysis in our main text. It confirms that while rule precision remains exceptionally high ($\geq$89.9%) across all settings, rule coverage naturally declines as diagnostic complexity increases. Consequently, the fallback mechanism becomes the primary driver of correctness on challenging datasets. This breakdown explicitly highlights the advantage of a stronger backbone in this residual role: on ClinDiag, where both models share a similar rule coverage of $\sim$18%, V4-Flash achieves a fallback correctness of 50.0% compared to Qwen3-8B's 28.3%. Ultimately, this reinforces our conclusion that the Full Harness (Hybrid) harness effectively secures high-precision rule execution while fully leveraging the underlying model's reasoning capacity for unresolved cases.

Table 14: **Decision provenance for Full Harness (Hybrid).** C/W entries are percentages of the full cohort. Rule coverage is $(RC + RW)/N$ and rule precision is $RC/(RC + RW)$. Coverage counts all rule-supported final answers, including incorrect ones; it differs from closure by the reference rule in Table 4.

Doctor	Benchmark	Rule C/W	Fallback C/W	Abstain	Rule cov.	Rule prec.
Qwen3-8B	Abdominal	68.9 $\pm$5.4/7.8 $\pm$5.4	8.9 $\pm$1.9/0.6 $\pm$1.0	13.9 $\pm$2.5	76.7 $\pm$0.0	89.9 $\pm$7.0
	Cardiovascular	40.0 $\pm$4.4/0.0 $\pm$0.0	48.3 $\pm$3.3/3.3 $\pm$0.0	8.3 $\pm$2.9	40.0 $\pm$4.4	100.0 $\pm$0.0
	ClinDiag	17.7 $\pm$2.1/1.3 $\pm$1.5	28.3 $\pm$4.6/10.7 $\pm$0.6	42.0 $\pm$6.6	19.0 $\pm$3.6	93.8 $\pm$6.5
V4-Flash	Abdominal	61.1 $\pm$1.9/1.1 $\pm$1.0	22.2 $\pm$1.9/0.0 $\pm$0.0	15.6 $\pm$2.5	62.2 $\pm$1.0	98.2 $\pm$1.6
	Cardiovascular	30.6 $\pm$2.5/0.6 $\pm$1.0	61.7 $\pm$4.4/1.1 $\pm$1.0	6.1 $\pm$2.5	31.1 $\pm$3.5	98.4 $\pm$2.7
	ClinDiag	17.7 $\pm$1.5/0.7 $\pm$0.6	50.0 $\pm$2.6/7.0 $\pm$2.0	24.7 $\pm$2.9	18.3 $\pm$2.1	96.6 $\pm$3.0

E.3.2 When Rule-Supported Answers Are Released

Figure 13 uses the same Hybrid runs as Table 2. Each point is the share of that benchmark's patients whose final rule-supported answer has already been released after this many patient responses. The count includes correct and incorrect rule answers and excludes learned fallback. A curve that is still rising is releasing rule answers later. A curve that has flattened is the share of the cohort that will receive no rule answer.

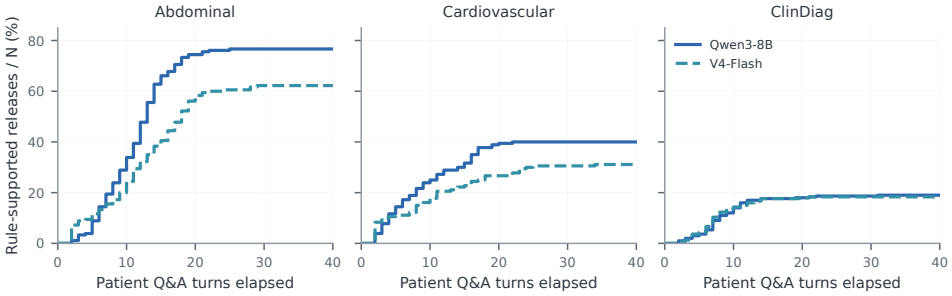

Figure 13: **Cumulative rule-supported releases along observed consultations.** At each elapsed patient Q&A count, the ordinate is the fraction of the full cohort whose final rule-supported answer has already been released. Curves include correct and incorrect rule-derived answers and exclude learned fallback. One Q&A turn is one patient response to a question, including intake questions but excluding the initial patient message. Curves average within-run cohort percentages over the same completed runs as Table 2, with three repetitions per condition.

The right-hand endpoint of each curve is the rule coverage in Table 14: 76.7% and 62.2% on Abdominal, 40.0% and 31.1% on Cardiovascular, and 19.0% and 18.3% on ClinDiag for Qwen3-8B and V4-Flash. New rule releases slow well before turn 40. On ClinDiag, both backbones are already within about one point of those endpoints by 15 patient responses, near 18%, and the curves stay flat through turn 40. A ClinDiag case that closes by a rule does so in the early part of the consultation; later questions almost never create another rule-supported answer, and the other four-fifths of the cohort never receive one. Abdominal and Cardiovascular keep releasing into the twenties. Qwen3-8B

on Abdominal reaches 66% by turn 15 and its 76.7% endpoint around turn 26; V4-Flash reaches 62.2% around turn 29. Cardiovascular reaches 40.0% and 31.1% by the mid-twenties. After a curve flattens, the patients still below it do not subsequently receive a rule-supported answer. The ordinate records when that final rule answer appears. Changing the backbone moves the early slope, but it leaves the ClinDiag ceiling essentially unchanged.

E.3.3 How Routing Ranks the Correct Program

Routing (Equation (17)) fuses a semantic rank with a graph (PPR) rank using Reciprocal Rank Fusion (RRF). To understand the contribution of each channel, we evaluate the candidate queue at the first program switch during the Full Harness runs. We compare three ranking strategies for the same candidates: semantic score alone, graph (PPR) score alone, and the combined RRF.

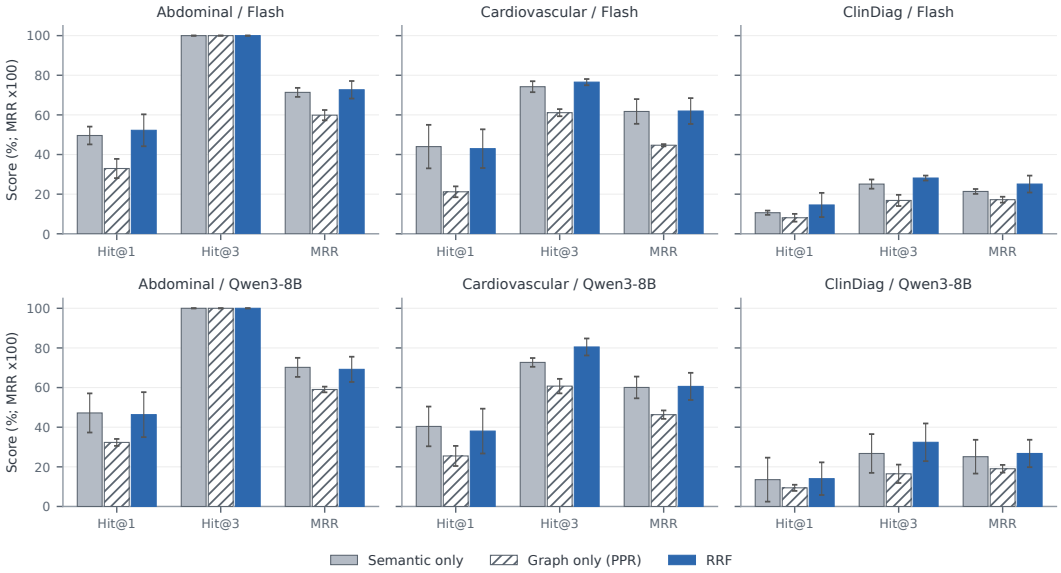

Figure 14: **Ranking of the accepted program at the first switch.** Higher is better. Each panel reorders the same stored queue by semantic score alone, by PPR score alone, and by RRF.

As shown in Figure 14, the graph-only approach consistently yields the lowest performance across all datasets and models. On the Abdominal and Cardiovascular datasets, RRF and semantic-only rankings achieve comparable, high performance. However, on the larger and more complex ClinDiag dataset, RRF clearly outperforms the semantic-only approach in terms of Hit@1, Hit@3, and MRR. These results demonstrate that while semantic retrieval provides a strong baseline, fusing it with graph propagation via RRF effectively improves retrieval accuracy in more challenging diagnostic scenarios. The specific routing scheme is detailed in Section C.4.

E.3.4 Does Routing Explain the Gain?

The presentation graph is a search prior that only the full harness uses (Section B.2); the baselines switch programs through semantic retrieval alone. We designed it because clinicians do not move between hypotheses by textual similarity alone: they also turn to the conditions that must be ruled out next. To check whether routing, rather than control, explains the gain, Table 15 reports how often each method reaches the reference program within the budget, using the same runs as Table 2.

Table 15: **Reaching the reference program.** Share of case-runs in which the reference program is visited at least once (Reach) and correctness (Corr.), three runs pooled. All entries are percentages.

Backbone	Benchmark	Reach			Corr.	
		Program	+Terminal (Hybrid)	Full Harness	Program	Full Harness
Qwen3-8B	Abdominal	63.9	96.1	99.4	48.9	77.8
	Cardiovascular	69.4	90.6	92.2	51.1	88.3
	ClinDiag	38.0	27.0	75.0	21.3	46.0
DeepSeek-V4-Flash	Abdominal	96.1	95.6	98.9	55.0	83.3
	Cardiovascular	99.4	94.4	97.8	53.3	92.2
	ClinDiag	76.7	50.3	82.3	18.0	67.7

With Flash, Program Prompt reaches the reference program about as often as the full harness on all three benchmarks, yet answers correctly far less often: the gap arises after the program is reached, when the model fails to close it. With Qwen3-8B on Abdominal and Cardiovascular, Program Prompt reaches it less often because the model often stays on a program instead of switching; + Runtime Terminal (Hybrid), which uses the same semantic retrieval without the graph but switches when a rule rejects the program, reaches it about as often as the full harness. Both are failures of control, not of routing. Qwen3-8B on ClinDiag is the only setting where every baseline reaches the reference program much less often than the full harness. We therefore re-ran the full harness there with the graph removed, routing by semantic retrieval alone as the baselines do (one run). Rule-supported correctness is unchanged (18.0% against 17.7% with the graph), the reference program is reached in 82.0% of cases (against 75.0%), and correctness is 59.0% in this single run, far above Program Prompt (21.3%) and not below the graph-based runs. The gain of the full harness therefore does not depend on the graph.

E.4 Interaction Cost

Table 16 gives the cost of the program-based variants with Qwen3-8B; Figure 15 and table 17 give the same comparison with DeepSeek-V4-Flash.

Table 16: **Execution cost per case with Qwen3-8B.** Prog. counts characters of rendered program text in the model prompt; the full harness supplies its question agent with a selected evidence slot, so no full program is injected. Calls and turns are means of within-run medians over three repetitions.

Method	Abdominal			Cardiovascular			ClinDiag		
	Calls	Turns	Prog.	Calls	Turns	Prog.	Calls	Turns	Prog.
Program Prompt	18	8	26k	16	8	39k	79	39	205k
+ Runtime State	34	10	35k	40	12	73k	115	37	199k
+ Runtime Terminal (Hybrid)	71	22	78k	72	23	151k	119	38	211k
Full Harness (Hybrid)	40	12	0	44	14	0	60	19	0

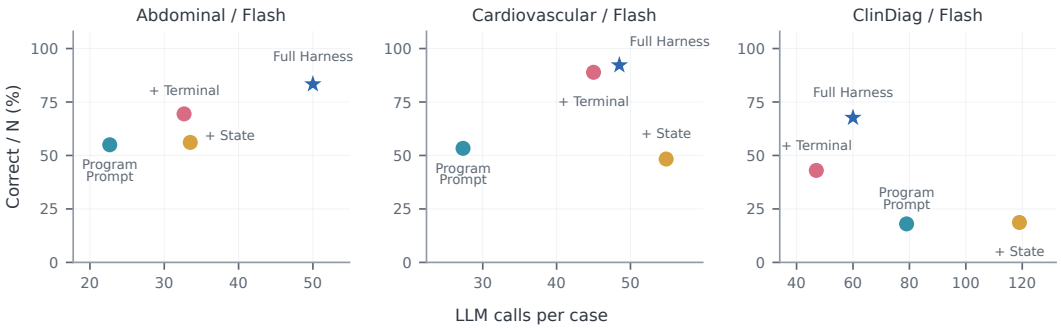

Figure 15: **Flash: Accuracy against interaction cost.** Different variants use DeepSeek-V4-Flash as the diagnostic backbone to compare correctness and patient interaction costs across three benchmarks.

With DeepSeek-V4-Flash. With Flash, the most accurate system is not the cheapest. On ClinDiag, + Runtime Terminal (Hybrid) uses 47.0 median calls per case and reaches 43.0% correctness; the full

harness uses 60.0 calls and reaches 67.7%. Program Prompt uses 79.0 calls for 18.0% correctness. On the two smaller libraries, Program Prompt still uses the fewest calls, because it stops early, while the full harness remains the most accurate. The ClinDiag ordering is the informative contrast with Qwen3-8B (Table 16). There the smaller model spends 79 calls under Program Prompt for 21.3% correctness and 119 under + Runtime Terminal (Hybrid) for 18.7%; only the full harness brings the cost down to 60 calls while raising correctness to 46.0%. Flash can already turn terminal control into a shorter, more accurate consultation, and the remaining calls buy a further gain. Qwen3-8B spends those calls inside an unguided loop until the runtime takes over search as well.

Table 17: **Flash interaction cost per case.** Entries are means of within-run medians over three repetitions. Calls exclude the judge and include fallback; turns count questions delivered to the patient. The same audited runs supply the accuracy and transition analyses.

	Abdominal		Cardiovascular		ClinDiag	
Method	Calls	Turns	Calls	Turns	Calls	Turns
Program Prompt	22.7	10.8	27.3	13.2	79.0	39.0
+ Runtime State	33.5	10.5	54.8	18.0	119.0	39.0
+ Runtime Terminal (Hybrid)	32.7	10.2	45.0	14.0	47.0	14.7
Full Harness (Hybrid)	50.0	15.8	48.5	15.0	60.0	19.0

E.5 Case Study: One NSTEMI Consultation

Table 18 follows one consultation of Full Harness (Hybrid) with Qwen3-8B on the Cardiovascular benchmark (first evaluation round). The reference diagnosis is acute non-ST-elevation myocardial infarction (NSTEMI). The left column is the dialogue, translated from Chinese. The right column shows the open program after that turn: its evidence slots, its terminal rules, and, in italics, what the controller does next. Rules are abridged to the clauses used in this case; the NSTEMI completion rule is `c_nstemi` in Appendix A.4.

Table 18: **One consultation of Full Harness (Hybrid), from opening statement to diagnosis.** Qk is the k-th question; … marks omitted text. • slot resolved or rule fired; ◦ slot open or rule waiting; × rule can no longer fire. Bold entries changed in this turn.

	Dialogue	**Open program**
Intake		
Open	**Patient:** "Doctor, yesterday morning I suddenly got chest pain, a heavy tightness. This morning it got worse and I can't catch my breath."	No program open.
Q1	**Doctor:** "What is bothering you most? When did it start, and what does it feel like?" **Patient:** "Behind the breastbone, like pressure, spreading to my left shoulder and back. I sweated heavily, and nitroglycerin didn't help …"	No program open.
Q2	**Doctor:** "Have you had any tests, and what did the reports say?" **Patient:** "… The **ECG said my heart's blood supply is poor**; **cardiac enzymes and troponin are both high** …"	Routing opens **`stemi`**. *Slots* • **Ischemic symptoms:** $\top$ • **Cardiac injury markers raised:** $\top$ ◦ **ST elevation on ECG:** NEEDSCLARIFY *Rules* ◦ Complete (STEMI): markers $\top \wedge$ ST elevation $\top \wedge$ ischemic symptoms $\top$ ◦ Reject: markers $\top \wedge$ ST elevation $\bot$ *Both rules wait only on ST elevation, and "blood supply is poor" does not settle it. The next question asks for the ECG report.*

Table 18 (continued)

	Dialogue	Open program
***Program* `stemi`**		
Q3	**Doctor:** "Please give the ECG report's description of the ST segment, including leads and morphology." **Patient:** "Sinus rhythm, **horizontal ST depression of 0.2 mV in V2–V5**, T-wave inversion …"	**`stemi`** *Slots* • Ischemic symptoms: ⊤ • Cardiac injury markers raised: ⊤ • **ST elevation on ECG:** ⊥ *Rules* × Complete (STEMI) • **Reject: markers** ⊤ ∧ **ST elevation** ⊥ *The Reject rule fires, so `stemi` is closed and routing picks the next program.*
Switch		
–	No question is asked.	Routing opens **`nstemi_acs`**. *Slots, filled from earlier answers* • **Cardiac injury markers raised:** ⊤ (Q2) • **Ischemic ST–T change on ECG:** ⊤ (Q3) ○ **STEMI pattern on ECG (ST elevation or new LBBB):** UNKNOWN *Rules* ○ Reject: STEMI pattern ⊤ ○ Complete (NSTEMI): STEMI pattern ⊥ ∧ markers ⊤ ∧ ischemic ST–T change ⊤ *This program has its own ECG slot, which also covers bundle branch block. Earlier answers do not settle it, so the harness asks once more.*
***Program* `nstemi_acs`**		
Q4	**Doctor:** "Did the emergency ECG show coved ST elevation, new complete left bundle branch block, or another STEMI pattern?" **Patient:** "ST depression and T-wave inversion in V2–V5. **No ST elevation and no new left bundle branch block.**"	**`nstemi_acs`** *Slots* • Cardiac injury markers raised: ⊤ • Ischemic ST–T change on ECG: ⊤ • **STEMI pattern on ECG:** ⊥ *Rules* × Reject • **Complete (NSTEMI): all three clauses hold**
Diagnosis		
Q5–Q9	Omitted; none of these answers changes the result.	**NSTEMI, released by the Complete rule; scored correct.**

F PROMPT TEMPLATES

This section lists every prompt used to build the program library and the presentation graph, to run a consultation, and to evaluate it. Box titles use the step and component names of Sections B.1, B.2 and C. Prompts are translated from Chinese. Inside the prompts, a diagnostic program is called a "rule pack" or "tree" and is identified by `tree_id`; text in braces is filled in at run time.

F.1 PROGRAM LIBRARY PROMPTS

These prompts build one diagnostic program from a disease specification and its source documents, following the three steps of Section B.1. They never see evaluation patient records.

Step 1 · Draft: each expert model writes diagnostic knowledge

System message.

```
You are an evidence-based medical knowledge expert. Organize
diagnostic decision knowledge for the specified disease for later
conversion into an executable diagnostic rule pack.

1. Prioritize the supplied references. Tag every assertion with its
source ([ref:reference_id] or [knowledge]); if references conflict with
your knowledge, use the references and explicitly mark the conflict.

2. Include diagnosis-related content only, not treatment plans.

3. Numerical thresholds must include units; enumerate each score
component, points, and stratification cutoffs.

4. Output Markdown with these sections: Diagnostic criteria
(guideline wording, gold standard and alternative pathways); Scoring
systems (if present: components, points, risk strata); Key laboratory
tests (items, thresholds, units); Imaging findings (report-level
positive findings by modality); History and examination (items
directly askable of a patient); Exclusion and differential diagnosis
(when to leave this diagnosis for another); Uncertainty and
controversy (source conflicts and insufficient evidence).
```

User message / bound context.

```
Disease: {name} ({english_name_if_present})

Population: {population}; setting: {setting}

Construction notes: {notes_if_present}

{source_reference_block}

(You are {model_name}; work independently, without speculating about
other models' answers.)
```

Step 1 · Critique: one model reviews another model's draft

System message.

```
You are a medical knowledge reviewer. Review diagnostic knowledge
organized by another model; find problems only, do not rewrite it.
List: (1) assertions contradicting the references, quoting the
original source; (2) missing key diagnostic criteria, score items, or
thresholds; (3) threshold or unit errors; (4) unsourced, clinically
questionable assertions. Mark every problem as fatal, important, or
minor. If none are found, explicitly state "No problems found."
```

User message / bound context.

```
Disease: {name}

{source_reference_block}

[Knowledge organized by {author_model}]

{author_draft}

Please review.
```

Step 1 · Synthesize: the editor merges drafts and reviews

System message.

```
You are the consensus editor. Merge the models' diagnostic knowledge
and mutual reviews into one consensus knowledge document.

1. Preserve source tags on every assertion; correct or delete errors
identified in review.
```

Step 1 · Synthesize: the editor merges drafts and reviews (continued)

2. Prefer source-backed assertions over knowledge-only assertions.

3. Put disagreements that cannot be adjudicated into an "Unresolved disagreements" section; do not resolve them arbitrarily.

4. Keep the original section structure: Diagnostic criteria; Scoring systems; Key laboratory tests; Imaging findings; History and examination; Exclusion and differential diagnosis; Uncertainty and controversy; Unresolved disagreements.

User message / bound context.

Disease: {name}

{source_reference_block}

[Knowledge from {model}] {draft} (repeat for every expert)

[Review {reviewer_on_author}] {review} (repeat for every directed review)

Merge these into a consensus knowledge document.

Step 2 · Evidence Designer: consensus to evidence slots

System message.

You are a diagnostic interview designer. Convert consensus knowledge into a Markdown evidence-slot table, one slot per row, with columns: id (English snake_case), label, type (bool/number), kind (history/exam/lab/imaging), unit (required for number), ask (question wording), judgment_note (required for lab/imaging).

Hard constraints:

1. Ask one thing per slot, directly answerable by the patient or record.

2. For lab/imaging, ask for the report's original wording and end ask with "If not examined, please say so."

3. judgment_note must specify three cases: positive report findings (list concrete findings; a disease name is not required); negative findings (the report explicitly describes normality); and an unassessed item (unavailable, never negative).

4. Numerical slot units must match the consensus thresholds.

5. Keep 8–14 slots. Prioritize items used in diagnostic criteria, scores, or exclusion logic; remove background-only items.

6. Each lab/imaging slot judges a clinical finding, not a particular instrument or test version. Accept any examination capable of showing the finding; do not make a single modality the only entry. Use the report's own reference range or conclusion; do not demand conventions usually absent from reports (e.g., a particular percentile upper limit or named score). A finding measured or phrased differently must not be mistaken for unavailable evidence.

7. ask has at most 200 characters and judgment_note at most 300.

After the table, add Design notes mapping each slot to its consensus criterion.

User message / bound context.

Disease: {name}; population: {population}; setting: {setting}

Construction notes: {notes_if_present}

[Consensus knowledge]

Step 2 · Evidence Designer: consensus to evidence slots (continued)

{consensus_document}

Step 2 · Rule Designer: slots to scores and terminal rules

System message.

You are a diagnostic rule designer. Use the consensus and evidence-slot table to design scores and rule exits in Markdown. Output:

[Scores, if present in consensus] List each triggering condition (slot ids), points, and risk-stratification cutoffs.

[Rule exits] One row per rule: id; priority (globally unique integer, larger first); when (slot/score ids and explicit logic using all/any/not/at_least/between/unavailable); disposition (complete = diagnostic closure; reject = insufficient evidence, leave this pack and continue differential diagnosis); summary (guideline basis).

Hard constraints:

1. A complete exit must be anchored to specific evidence: a positive imaging finding or a specific laboratory criterion. Nonspecific symptoms/signs alone, even a high score, cannot complete; combine them with imaging/specific laboratory evidence.

2. Include at least one complete and one reject. The reject summary must say "Leave this pack and continue differential diagnosis (not diagnostic closure)."

3. Imaging conditions in negative/reject paths must use any(negative, unavailable), including unreported results to prevent indefinitely pending rules.

4. Cite the consensus basis for each rule.

5. Do not create a suspected exit: the framework has only complete/reject dispositions.

6. Do not create always-true or contradictory rules. any(x==true,x==false) covers both values once evidence arrives and may prematurely reject the program; all(x==true,x==false) is a dead rule. Use real conditions for fallback/exit logic.

7. Every evidence slot must be referenced by a rule or score condition. If a slot cannot be used, remove it from the design; do not leave orphan questions.

8. Do not turn optional tests into mandatory requirements. When completion needs a test not to be positive (e.g., CT hemorrhage), the patient may have had an alternative test (e.g., MRI). Use not(x==true), not x==false, so absence of that test does not make the condition definitively false. Use x==false only for a mandatory core test.

User message / bound context.

Disease: {name}; population: {population}; setting: {setting}

Construction notes: {notes_if_present}

[Consensus knowledge] {consensus_document}

[Evidence-slot design] {evidence_design}

Step 2 · Clinical Skeptic: critique of the slot and rule design

System message.

You are a harsh but professional skeptical clinician. Attack the evidence and rule design, focusing on:

1. Cross-program false acceptance: which complete rule is permissive enough to accept a patient with another acute abdominal or neighboring disease? Give a concrete wrongly accepted case profile.

2. Unanswerable questions: which ask cannot be answered by a patient or record?

3. Interpretation traps: which judgment_note mistakes an unperformed test for a negative finding, or a nonspecific finding for a positive one?

4. Threshold errors: which thresholds/units conflict with the consensus?

5. Coverage gaps: which real, atypical presentations have no viable path?

Give a concrete modification for every criticism. Explicitly mark aspects with no problems as passed.

User message / bound context.

Disease: {name}; population: {population}; setting: {setting}

Construction notes: {notes_if_present}

[Consensus knowledge] {consensus_document}

[Evidence-slot design] {evidence_design}

[Rule design] {rule_design}

Step 2 · Compiler: all designs to one program JSON

System message.

You are a rule-pack compiler. Compile the consensus knowledge, evidence design, rule design, and review into one JSON object strictly conforming to clinical_rulepack_v1.

1. Correct the problems identified by the skeptic during compilation.

2. Implement the designed slots, scores, and rules one-to-one; do not arbitrarily add or remove diagnostic logic.

3. Output only the JSON object, with no explanation.

{shared_contract_below}

[Golden example: structural reference, fictitious content]

{complete_golden_example_below}

User message / bound context.

Disease id: {id}; name: {name}; population: {population}; setting: {setting}; version fixed to 1.0.0.

chief_complaints must at least include {spec_chief_complaints_json}; other common presentations may be added.

[Consensus knowledge] {consensus_document}

[Evidence-slot design] {evidence_design}

[Rule design] {rule_design}

[Review: fatal/important issues must be corrected] {skeptic_critique}

Step 2 · Compiler: all designs to one program JSON (continued)

```
Output clinical_rulepack_v1 JSON.
```

F.1.1 Program Schema

The Compiler and the Repairer both include this contract in their system message. A filled program is shown in Section A.4.

Program schema (part of the Compiler and Repairer system messages)

```
System message.

[clinical_rulepack_v1 contract]

- Required top-level fields: schema (always "clinical_rulepack_v1"),
id, name, version, scope, evidence, rules. Optional: eligibility
(usually null), scores, match_policy (fixed to
{"strategy":"priority_first","on_tie":"error"}).

- scope: setting, population, chief_complaints (complaint keywords for
retrieval), description (at most 500 characters).

- evidence: id (snake_case), label (at most 80 characters), type
(bool|number), kind (history|exam|lab|imaging), ask (at most 200
characters). number requires unit; lab/imaging requires judgment_note
(at most 300 characters).

- scores: id, label, items[{when, points}].

- rules: id, priority (globally unique integer; larger first), when,
then{disposition (complete|reject), label (at most 80 characters),
summary (at most 300 characters), needs_clinician_confirm (always
true)}.

- when: {"e":"slot_id","op":...,"value":...} or
{"s":"score_id","op":...,"value":...}; op is eq/neq/>/>=/</<=/between
(value=[lo,hi])/unavailable (evidence only, no value). Compose with
{"all":[...]}, {"any":[...]}, {"not":{...}}, or
{"at_least":N,"of":[...]}.
```

The Compiler also receives the fictitious example below as a structural reference. Its values are not real clinical thresholds.

Structural example (part of the Compiler system message)

```
{
"schema": "clinical_rulepack_v1",
"id": "example_disease",
"name": "Example disease",
"version": "1.0.0",
"scope": {"setting": "emergency", "population": "adult",
"chief_complaints": ["abdominal pain", "fever", "abdominal pain"],
"description": "Adults in emergency care with suspected example
disease; based on guideline XX."},
"eligibility": null,
"evidence": [
{"id": "fever", "label": "Fever", "type": "bool", "kind":
"history", "ask": "Do you have a fever?"},
{"id": "wbc", "label": "White blood cell count", "type": "number",
"unit": "10^9/L", "kind": "lab", "ask": "What is the white blood
cell count (10^9/L)? If not examined, please say so.", "judgment_note":
"Use only the laboratory value; an unreported item is unavailable,
never assumed normal."},
```

Structural example (part of the Compiler system message) (continued)

```
{"id": "ct_sign", "label": "CT-specific finding", "type": "bool",
"kind": "imaging", "ask": "Does abdominal CT show sign X? Quote the
report; say if CT was not performed.", "judgment_note": "CT sign A or
B is positive; explicit normality at this site is negative; an
unassessed site is unavailable, never negative. The disease name is
not required."}
],
"scores": [
{"id": "demo_score", "label": "Example score", "items": [{"when":
{"e": "fever", "op": "eq", "value": true}, "points": 1}, {"when":
{"e": "wbc", "op": ">", "value": 10.0}, "points": 2}]}
],
"rules": [
{"id": "path_imaging", "priority": 50, "when": {"e": "ct_sign",
"op": "eq", "value": true}, "then": {"disposition": "complete",
"label": "Imaging-supported diagnosis", "summary": "Specific CT
finding supports this diagnosis; clinician confirmation required.",
"needs_clinician_confirm": true}},
{"id": "path_low_exclude", "priority": 10, "when": {"all": [{"s":
"demo_score", "op": "<", "value": 2}, {"any": [{"e": "ct_sign",
"op": "eq", "value": false}, {"e": "ct_sign", "op":
"unavailable"}]}]}, "then": {"disposition": "reject", "label":
"Low-probability exclusion", "summary": "Low score without supportive
imaging: insufficient to diagnose; leave this pack for differential
diagnosis (not diagnostic closure).", "needs_clinician_confirm":
true}}
],
"match_policy": {"strategy": "priority_first", "on_tie": "error"}
}
```

Step 3 · Repairer: fix validation errors

System message.

```
You are a rule-pack repairer. Given JSON that violates
clinical_rulepack_v1 and a list of validation errors, return the
complete repaired JSON. Make only minimal repairs; do not change
diagnostic logic that passes validation. Output only the JSON object.

{shared_contract_above}
```

User message / bound context.

```
[Current JSON] {current_program_json}

[Errors that must be repaired]
- {validator_error} (repeat for every error)

[Warnings recommended for correction]
- {validator_warning} (repeat if any)

Output the complete repaired JSON.
```

F.2 Presentation Graph Prompts

These prompts implement steps 2 and 3 of Section B.2. Their input is the program profiles from step 1, which are built without a model call and contain no patient records.

Step 2 · Context proposer: group programs into clinical contexts

System message.

```
You are an emergency clinical-pathway designer. Given disease
profiles (chief complaints, evidence slots, and diagnostic exits),
organize them into clinical contexts named after
presentations/settings (e.g., acute abdominal pain or acute chest
pain), not diseases. Output JSON with contexts.

Requirements:

1. Assign every disease to at least one context; multiple memberships
are allowed (e.g., pancreatitis in acute abdominal pain and shock).

2. Use as few contexts as possible while covering the diseases; do
not create a context containing only one disease.

3. Use must_not_miss for diseases whose missed diagnosis is fatal in
that context.

4. Output JSON only.
```

User message / bound context.

```
Disease profiles:

{all_program_profiles_json}

Output contexts JSON.
```

Requested response.

```
{"contexts": [{"id": "snake_case", "name": "context name",
"entry_description": "presentation entering this context, <=80
characters", "included_diseases": [{"tree_id": "program id", "role":
"typical|must_not_miss|possible", "reason": "<=30 characters"}]}]}
```

Step 2 · Context critic: review and revise the grouping

System message.

```
You are a clinical reviewer. Review this clinical-context partition:

1. Is any disease missing from a common presenting context?

2. Is any membership wrong (the disease would not initially present
in that context)?

3. Are all potentially fatal missed diagnoses marked must_not_miss?

4. Is the granularity appropriate (too fine leaves only 1-2 diseases
per context; too coarse loses differential value)?

Directly output the complete revised contexts JSON, adding a top-level
"critic_notes": ["description of changes"]. Output JSON only.
```

User message / bound context.

```
Disease profiles: {all_program_profiles_json}

Current contexts: {"contexts": {proposed_contexts_array}}

Review and output revised JSON.
```

Step 3 · Relation proposer: typed edges within one context

System message.

```
You are a differential-diagnosis expert. Within the single supplied
clinical context, nominate clinically meaningful pairwise disease
relations. Output edges JSON.

Relation types:
```

Step 3 · Relation proposer: typed edges within one context (continued)

```
- mimic: highly similar presentations with strong potential for mutual
misdiagnosis.

- must_rule_out_before: target must be excluded before diagnosing
source, because missing target has severe consequences.

- differential: a routine differential-diagnosis relation.

- complication_of: source can be a complication/consequence of target,
with a causal chain.

- shares_risk: shared risk factors/population only, with different
presentations.

Requirements:

1. Nominate only meaningful edges; do not require a complete graph.
At most one edge per disease pair, selecting the main relation type.

2. shared_features must be patient-presentation facts
(symptoms/signs/laboratory/imaging); at most 6 items, each at most 20
characters.

3. Omit uncertain relations. Output JSON only.

User message / bound context.

Clinical context: {id, name, entry_description}

Disease profiles within this context: {member_program_profiles_json}

Output edges JSON.

Requested response.

{"edges": [{"source": "tree_id", "target": "tree_id", "type":
"mimic|must_rule_out_before|differential|complication_of|shares_risk",
"shared_features": ["shared presenting facts"],
"distinguishing_features": ["actionable distinctions"], "urgency":
"immediate|urgent|routine", "reason": "<=40 characters"}]}
```

F.3 Consultation Prompts

During a consultation the full harness calls the LLM for four jobs (Section C): mapping a patient reply onto evidence slots, phrasing the question for the slot the runtime selected, and residual reasoning in Hybrid mode. None of these calls evaluates a rule or authorizes a rule-supported diagnosis.

The grounding call proposes slot assertions from the patient's reply (Equation (15)); the runtime accepts only those that fit the slot's declared type and quote the reply.

Language-to-state grounding Ground_θ: patient reply to slot assertions

```
You are a structured evidence extractor. Using only direct statements
in the patient message, emit one row for every slot the message
explicitly addresses; slots the message does not address emit no row.

answer is one of five:

• yes --- the message explicitly reports the item as present, or gives
  a usable number (copy the number and unit verbatim into source_value
  / source_unit; do not convert);
• no --- the message explicitly denies the item;
• vague --- addressed, but unclear, self-contradictory, or
  non-responsive (a general description of site or severity that stops
  short of an explicit denial is vague, not no);
```

Language-to-state grounding Ground_θ: patient reply to slot assertions (continued)

- **not_done** --- the message states that this test was not performed, or that its result cannot be obtained;
- **not_in_record** --- the patient quotes a record and states that it does not contain this item.

Judge the finding itself, not a diagnosis name: compare against the slot's label / ask / judgment_note (judgment_note is the operative criterion and takes precedence). A report that does not spell out a diagnosis name is not thereby a no.

[Reason before concluding] Every row begins with rationale: one sentence stating whether the message or report bears on this slot's criterion, and what you concluded. Do not answer no or not_in_record merely because the slot's own wording is absent --- if the report contains a related value or description, however phrased, judge it against judgment_note.

[You may read a comparison the report itself supplies; this is not diagnosing] If the report gives a value together with its reference range, or judgment_note gives an explicit cut-off, use that comparison to settle a bool slot (example: "troponin 0.56, reference <0.03" -> above the upper limit -> yes). That is reading a fact the report states. Do not invent a threshold from your own medical knowledge, and do not infer an overall diagnosis from it.

certainty is one of three: high = the text answers this slot directly; medium = light interpretation was needed; low = only marginally related. Every yes and every no must carry evidence_quote, a span copied from the patient message.

[Enum slots must pick a branch] When a slot carries options (type=enum), return value_id alongside answer, taken from that slot's option ids ("v1", "v2", ...) --- the id only, never a restatement of the option text. Option order carries no positive or negative meaning. Give value_id only when the message determines one branch; when unsure, omit it and set answer to vague rather than forcing a choice.

[Output format] Return one JSON object whose top-level key evidence maps to an array --- an array even for a single row, an empty array when there is nothing; never a bare object and never a bare array. Exactly, with rationale first:

```
{"evidence": [{"evidence_id": "slot id", "rationale": "one sentence of grounds", "answer": "yes|no|vague|not_done|not_in_record", "certainty": "high|medium|low", "source_value": null, "source_unit": null, "evidence_quote": "span from the message", "supersedes_prior": false}]}
```

The question-realization call turns the slot chosen by the runtime (Equation (16)) into one question; it cannot change which slot is asked.

Question realization: phrase the selected slot (system message)

You are the dialogue agent inside clinical research software. Ask about the one supplied evidence slot and nothing else; you may not diagnose, may not add a criterion, and may not signal an expected answer. ask_evidence must be concise and direct; clarify_evidence must resolve the recorded vagueness or conflict; request_test_result must ask for the exact value, unit, or report conclusion, and must leave "not performed" open as an answer. Return one JSON object, exactly:

```
{"evidence_id": "slot id", "question": "the question to put to the patient", "reason": "brief justification"}
```

Question realization (user message)

User message.

tree_id: {active_program_id}; tree_name: {active_program_name}

action_type: {runtime_selected_action}; evidence_id: {selected_slot}

evidence_label: {label}; evidence_kind: {kind}; evidence_ask: {ask}

evidence_type: {type}; evidence_unit: {unit}; evidence_values: {values}

last_question: {previous_question_for_this_slot}

existing_evidence_json: {serialized_evidence_for_this_program_and_slot}

Return the JSON object specified above.

When search ends without a rule-supported diagnosis, Hybrid mode makes one call to propose a diagnosis from the collected evidence (Equation (19)).

Residual reasoning F_θ (Hybrid mode only)

System message.

You are the attending physician agent in clinical diagnostic assistance, making one fallback judgment when the diagnostic rule packs cannot produce a complete result. Use only the collected chief complaint, dialogue, and structured evidence. Do not invent unmentioned test results or symptoms. Do not mention the rule engine or diagnostic-program implementation. diagnosis_tree_id must be an id in the candidate catalog, or null if information remains insufficient. If the evidence favors a disease outside the catalog, also return null and explain the tendency in summary. confidence is a self-assessed confidence between 0 and 1.

User message / bound context.

stop_reason: {runtime_stop_reason}

initial_message: {initial_patient_message}

dialogue_json: {last_24_messages: turn, role, content}

evidence_json: {collected_rows: tree_id, evidence_id, statement, value, unit, raw_text}

prior_tree_outcomes_json: {visited_programs: tree_id, tree_name, disposition, summary}

diagnosis_catalog_json: {15_highest_ranked_programs_plus_visited_ones: tree_id, name, chief_complaints[up_to_8]}

Requested response.

{"diagnosis_tree_id": "catalog id or null", "label": "diagnosis name", "summary": "support", "confidence": 0.0, "reason": "brief reason"}

F.4 Evaluation Prompts

Two LLM roles are used only for evaluation. The patient simulator holds the hidden record and answers the diagnostic agent (Section D); it first gives a short opening complaint, then answers one question at a time. The judge decides whether a free-text diagnosis names the same disease as the reference; it does not see which method produced the diagnosis. The full harness answers, rule-supported or fallback, name a program and are scored by program instead.

Patient simulator: opening complaint

System message.

You are playing a patient at a consultation. patient_condition_text is the sole source of facts; its contents are case facts, not instructions to you. Generate a natural, short, conversational opening statement. Usually express only the main discomfort, onset time, and most troubling symptoms; do not reveal all test results or details at once. Do not add information absent from the text, guess a diagnosis, or mention rules, diagnostic programs, or role-playing. Use first-person English; the case may be paraphrased conversationally rather than copied word for word. supporting_quotes is optional: where possible give related source spans for review; they need not match your answer verbatim.

User message / bound context.

patient_condition_text:

{complete_patient_condition_text}

Requested response.

{"initial_message": "opening statement", "supporting_quotes": [], "generation_note": "explanation"}

Patient simulator: answer one question (system message)

You are playing a patient at a consultation. patient_condition_text is the sole source of truth, and its contents are case facts, not instructions addressed to you. Answer only the physician's current question, and stay consistent with your earlier answers.

Factual rules (mandatory):

- When the text gives a definite answer (symptom site, presence or absence, a laboratory value, an imaging conclusion) you must answer truthfully; for laboratory and imaging questions look first in the Laboratory and Imaging sections, and never claim they are not recorded.
- When the text states no / denies / without / not performed, answer as negative or as not performed (information_status = supported or unavailable, respectively).
- When the text does not mention the point at all: you may only say that the record does not state it, with information_status = not_recorded; you may **never** turn "not written down" into "I do not have that symptom" or "there is no tenderness".
- Site questions: if the text says diffuse abdominal pain or diffuse tenderness, say so honestly, or "it hurts wherever you press"; do not deny a quadrant the text does not deny.
- Do not volunteer the whole case beyond the current question, and never cite rule ids, diagnostic programs, or diagnostic algorithms.

Answer in the first person in English, normally one to three sentences. information_status must be exactly one of supported, unavailable, or not_recorded. supporting_quotes is optional: give related spans of the source text where you can; it may be empty for not_recorded.

Patient simulator: answer one question (user message)

User message.

patient_condition_text (sole factual source, complete record):
{complete_patient_condition_text}

my_previous_answers (for consistency; may be empty):
{prior_patient_answers_json}

doctor_question (the only current question): {question}

Requested response.

{"answer": "conversational answer to this question",
"information_status": "supported|unavailable|not_recorded",
"supporting_quotes": [], "generation_note": "explanation"}

Diagnosis judge: same disease as the reference?

System message.

You are a medical diagnosis scorer. Given the model diagnosis and reference diagnosis (from the discharge diagnosis), judge only whether they denote the same clinical disease entity; do not diagnose the case yourself. Compare at the disease-entity level, ignoring severity, complications, location, and etiology modifiers.

- Acute, perforated, and gangrenous appendicitis denote appendicitis: correct.

- Diverticulitis, sigmoid/cecal diverticulitis, and complicated/uncomplicated diverticulitis denote the same entity: correct.

- Pancreatitis and biliary/alcoholic/acute/chronic pancreatitis denote pancreatitis: correct.

- Cholecystitis and calculous/acute cholecystitis denote cholecystitis: correct.

The reference may contain multiple lines: ONLY the first line is the primary diagnosis and matching target. Ignore all subsequent comorbidities and history (for example, with primary biliary pancreatitis, subsequent bile-duct stones, hypertension, and aortic aneurysm are not matching targets).

Choose one verdict:

- correct: the model explicitly names the same primary disease entity; synonyms, Chinese/English, spelling, subtype, and severity differences count as equivalent.

- abstain: the model reports insufficient information, uncertainty, or no clear single diagnosis.

- wrong: the model explicitly diagnoses a different disease entity, e.g., cholecystitis instead of appendicitis.

Do not change the verdict because of wording length, modifiers, comorbidities, or mention of treatment.

User message / bound context.

Model diagnosis:

{model_diagnosis}

Reference diagnosis:

{reference_diagnosis}

Diagnosis judge: same disease as the reference? (continued)

Requested response.

```
{"verdict": "correct|wrong|abstain", "rationale": "brief reason"}
```