DIGNOSIS: VERIFICATION AND REPAIR OF TASK PLANS VIA CONDITIONAL GRAPH DIFFUSION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Task planning frameworks solve complex user queries by selecting a set of available tools and their dependencies, naturally represented as a plan graph. Whether they rely on large language models (LLMs), prompted or fine-tuned, or augment them with graph-based components, task planners can produce incorrect plan graphs, with steps of the query left uncovered, missing tool dependencies, or tools absent from the catalog. Verifying plan graphs before execution is therefore essential, yet most existing verifiers still rely on LLMs, which may be just as prone to errors as the planners they are meant to check. In this work, we propose DIGNOSIS, a planner-agnostic verifier that exploits the graph structure of the produced plans and recasts their verification as a graph denoising task. Concretely, we train a discrete graph diffusion model only on correct plan graphs, conditioned on the query that generated them. From this single denoiser, DIGNOSIS derives three signals for a candidate plan graph: a brokenness score, a per-node and per-edge diagnosis with preferred alternatives, and a one-shot repair. Score and diagnosis never involve an LLM, which merely proposes a repair candidate guided by the diagnosis, accepted only if it lowers the brokenness score. Extensive experiments on three datasets and four task planners of different nature show that DIGNOSIS improves the raw plan graph more often and by larger margins than four existing verifiers. Moreover, when the planner is fine-tuned on the same correct plan graphs, DIGNOSIS recognizes that its plans are largely correct and therefore abstains from repairing them.

1 INTRODUCTION

Task planning frameworks solve a complex user query by selecting a set of available tools and the dependencies among them, naturally represented as a *plan graph*, a directed graph whose nodes are the selected tools and whose edges encode their dependencies (Shen et al., 2024; Qin et al., 2024). Large language models (LLMs) are central to most of these frameworks, either as the planner itself, prompted once or iteratively (Shen et al., 2023; Yao et al., 2023a) or fine-tuned on tool-use data (Qin et al., 2024), or combined with graph-based components that select the tools over the catalog (Wu et al., 2024; Chen et al., 2026).

Regardless of their design, task planners can produce incorrect plan graphs. LLMs reason poorly over explicit graph structure (Wang et al., 2023; Fatemi et al., 2024) and hallucinate, while graph-based components inherit the query decomposition they operate on and rely on tool graphs with erroneous or missing edges (Chen et al., 2026; Kang and Li, 2026). Reasons for failure involve, for instance, steps of the query left uncovered, missing tool dependencies, or tools absent from the catalog, which makes verifying the plan graph before execution essential. Existing verifiers, however, largely rely on LLMs themselves, through self-critique (Madaan et al., 2023), per-step consistency checks (Ho et al., 2025), or rule checks followed by an LLM correction (Lee et al., 2025), and may thus be as error-prone as the planners they verify (Valmeekam et al., 2023; Kambhampati et al., 2024). A recent work trains a graph neural network to recognize flawed plan graphs (Hao et al., 2026), but it learns only from artificially corrupted examples and still relies on an LLM to fix them.

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.

To overcome these limitations, we exploit the graph structure of the produced plans and recast their verification as a graph denoising task. Rather than learning to recognize broken plan graphs, we learn the distribution of correct ones for a given query, and treat any candidate plan graph, whatever planner produced it, as its noisy version. Discrete graph diffusion models (Austin et al., 2021; Vignac et al., 2023a) are a natural fit, as they learn the validity constraints of small structured graphs from data alone, are trained to reconstruct clean graphs from corrupted ones, and have recently been used both to generate graphs for LLM agents (Jiang et al., 2026) and to edit existing graphs (Trivedi et al., 2024). On such a basis, we propose DIGNOSIS, a discrete graph diffusion model trained only on correct plan graphs and conditioned on their queries. From this single denoiser, DIGNOSIS derives three signals: a *brokenness score*, which flags the plan graphs to repair; a per-node and per-edge *diagnosis* with preferred alternatives; and a one-shot *repair*. Score and diagnosis never involve an LLM, which merely proposes a repair candidate guided by the diagnosis, accepted only if it lowers the brokenness score.

Extensive experiments on three datasets, four task planners of different nature and two LLM backbones show that DIGNOSIS improves the raw plan graph more often and by larger margins than four existing verifiers, without being retrained for each planner. When the planner is instead fine-tuned on the same correct plan graphs, DIGNOSIS recognizes that its plans are largely correct and abstains, while verifiers that rewrite every plan graph substantially degrade them. All things considered, our contributions may be summarized as follows:

- We observe that task planners of different nature produce incorrect plan graphs, and that existing verifiers largely rely on LLMs, which may be as error-prone as the planners they verify.
- We propose DIGNOSIS, the first planner-agnostic verifier that recasts plan verification and repair as a graph denoising task through conditional graph diffusion models, deriving score, diagnosis and repair from a single diffusion model trained only on correct plan graphs.
- Through extensive experiments, we show that DIGNOSIS improves the raw plan graph more often and by larger margins than four existing verifiers, and abstains when the planner's produced plans are largely correct.

2 PROBLEM FORMULATION

DEFINITION 2.1 (Tool graph) *Let $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_{|\mathcal{T}|}\}$ be the set of available tools (e.g., APIs or foundation models) in a tool repository. Each tool τ_i is represented through a textual description encoded by a pretrained language model (e.g., e5-large (Wang et al., 2022)) into an embedding $\mathbf{x}_i^T \in \mathbb{R}^f$. We define a set of directed edges $\mathcal{E}^T \subseteq \mathcal{T} \times \mathcal{T}$, where an edge $(\tau_i, \tau_j) \in \mathcal{E}^T$ indicates that tool τ_j can be invoked after tool τ_i according to a set of criteria (e.g., input-output compatibility or historical tool executions). Let $\mathbf{X}^T = \{\mathbf{x}_1^T, \ldots, \mathbf{x}_{|\mathcal{T}|}^T\}$ be the set of all tool embeddings. We define the tool graph as $\mathcal{G}^T = (\mathbf{X}^T, \mathcal{E}^T)$, directed by construction.*

DEFINITION 2.2 (LLMs-based task planning) *Let q be a complex user query expressed in natural language, and $\mathcal{G}^T$ the tool graph. LLMs-based task planning consists in defining a planner function (e.g., an LLM) that, given q, searches within $\mathcal{G}^T$ for a suitable plan graph $\hat{\mathcal{G}}_q^P$ able to solve the query. The plan graph is a directed graph whose nodes are the selected tools and whose edges represent the invocation dependencies among them.*

DEFINITION 2.3 (Failure reasons for the plan graph) *Regardless of the nature of the planner, a plan graph $\hat{\mathcal{G}}_q^P$ can be incorrect for reasons arising either during the interpretation of the query (e.g., its decomposition into subtasks, when performed) or during the plan graph search. We group these failures into three categories: (i) steps required by the query that are not recognized, and thus not covered by any tool in the plan graph; (ii) missing tool dependencies in the produced plan graph; (iii) selected tools that are not available in the tool catalog.*

Unlike molecules, whose validity depends on the graph alone (Jo et al., 2022; Vignac et al., 2023a), the correctness of a plan graph depends on the query it solves (Shen et al., 2024); we therefore condition verification and repair on the query itself.

DEFINITION 2.4 (Graph-based verification and repair) *Let q be a complex user query and $\hat{\mathcal{G}}_q^P$ a candidate plan graph produced by a task planner. Graph-based verification and repair of $\hat{\mathcal{G}}_q^P$ consists in designing a graph-based approach, trained on a set of ground truth plan graphs and without access to the ground truth plan graph $\mathcal{G}_q^P$ of q, that produces three signals: (i) a score Σ_q^P, indicating the level of brokenness of the whole plan graph; (ii) a diagnosis Δ_q^P, indicating the model's confidence in each node and edge and, where relevant, a preferred alternative; (iii) a repair Ω_q^P, namely a corrected plan graph for the same query.*

Given a query and a candidate plan graph produced by any task planner, our purpose is to verify the plan graph and, where needed, to correct it. We therefore recast graph-based verification and repair as a graph denoising problem, reinterpreting the failures affecting the candidate plan graph as noise on its node and edge attributes. We detail this approach, named DIGNOSIS, in the next section.

3 DIGNOSIS: CONDITIONAL GRAPH DIFFUSION FOR TASK PLAN VERIFICATION AND REPAIR

We present DIGNOSIS, which recasts graph-based plan verification and repair as graph denoising through conditional graph diffusion models. Such techniques (Vignac et al., 2023a) are the de facto standard for generating small, structured graphs such as molecules, whose topology is close to that of plan graphs. DIGNOSIS first trains a graph diffusion model only on ground truth plan graphs, conditioned on their queries, and then uses it to derive a score, a diagnosis and a repair for a candidate plan graph produced by any planner. We describe the plan graph and query representation, the conditional diffusion model, the three inference signals, and finally the verification pipeline combining them (Figure 1).

3.1 PLAN GRAPH AND QUERY REPRESENTATION

Let $\mathcal{Q} = \{q_1, \ldots, q_{|\mathcal{Q}|}\}$ denote the set of user queries recorded in our task planning system, each associated with a ground truth plan graph within $\mathcal{G}^P = \{\mathcal{G}_{q_1}^P, \ldots, \mathcal{G}_{q_{|\mathcal{Q}|}}^P\}$. For a query q, we indicate $\pi_q \subseteq \mathcal{T}$ for the ordered list of tools within its ground truth plan graph, and fix a capacity d of tool slots, strictly larger than $\max_{q \in \mathcal{Q}} |\pi_q|$, the largest number of tools required by any query in $\mathcal{Q}$; in our benchmarks, we fix $d = 12$ against reference plan graphs of at most 8 tools. Then, we assume (for ease of model design) that every plan graph uses a maximum fixed number of d tools; a candidate plan graph exceeding d tools is truncated and flagged as broken.

A candidate plan graph may contain a tool absent from the catalog, and a query may require fewer than d tools, leaving some slots unused. To account for this, we extend the tool catalog with two reserved tool categories, an empty class "$\varnothing$" and an out-of-catalog class "$\texttt{unk}$", obtaining $\mathcal{T}' = \mathcal{T} \cup \{\varnothing, \texttt{unk}\}$. We represent the nodes of a plan graph $\mathcal{G}_q^P$ with a matrix $\mathbf{X}_q^P \in \mathbb{R}^{|\mathcal{T}'| \times d}$, whose i-th column ($\mathbf{x}_i$) is the one-hot encoding, over $\mathcal{T}'$, of the tool assigned to slot i, with unused slots encoded as $\varnothing$. The empty class allows us to treat the insertion of a missing tool as an ordinary change of a single entry (from $\varnothing$ to a valid tool) rather than requiring a dedicated mechanism to expand the graph structure; conversely, the out-of-catalog class stands for every tool outside the catalog. Since $\texttt{unk}$ never appears in a ground truth plan graph, a model trained only on such graphs assigns it low probability, implicitly letting DIGNOSIS penalize out-of-catalog tools.

We represent the edges of $\mathcal{G}_q^P$ with a tensor $\mathbf{A}_q^P \in \{0,1\}^{2 \times d \times d}$, whose entry $\mathbf{a}_{ij}$ is the one-hot encoding of the presence or absence of a directed edge from slot i to slot j. The diagonal of $\mathbf{A}_q^P$ is fixed to the absence class, and, since the underlying plan graph is directed, $\mathbf{a}_{ij}$ and $\mathbf{a}_{ji}$ are independent and the tensor is never symmetrized. Thus, a ground truth plan graph is fully described by the pair $\mathcal{G}_q^P = (\mathbf{X}_q^P, \mathbf{A}_q^P)$, and remains, by construction, a directed acyclic graph.

Finally, we encode the query q itself with a pretrained language model $\phi(\cdot)$, for instance e5-large (Wang et al., 2022) as in our experiments, obtaining a fixed size embedding $\mathbf{H}_q = \phi(q) \in \mathbb{R}^{1 \times f}$. When the planner provides a textual description of the subtask associated with each slot, the same encoder embeds it as well. These two embeddings are the only signals through which the diffusion model becomes aware of the query a plan graph is meant to solve.

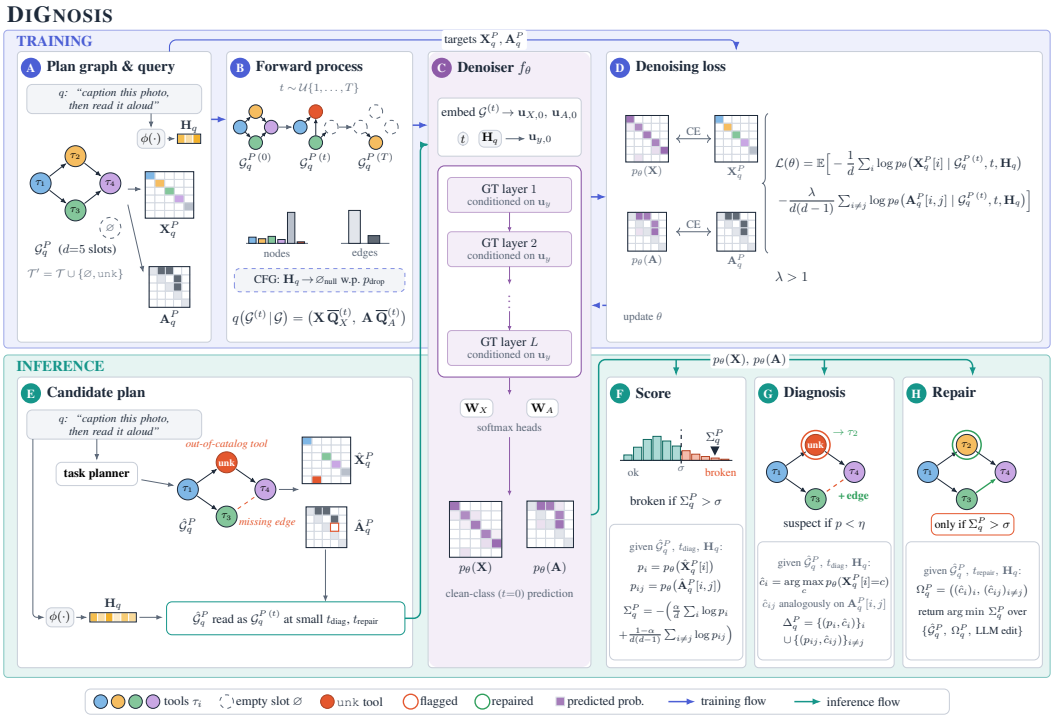


Figure 1: Overview of DIGNOSIS. Training (top): **(A)** a ground truth plan graph and its query are encoded into node matrix, edge tensor and query embedding; **(B)** the forward process corrupts nodes and edges toward their marginal distributions; **(C)** a graph transformer (GT) denoiser, conditioned on the query and the timestep, predicts the clean class of every node and edge, and **(D)** is trained with a cross-entropy loss. Inference (bottom): **(E)** a candidate plan graph from a task planner is read as a noised plan graph at a small timestep, from which the denoiser derives **(F)** the brokenness score, flagging the plan graph when $\Sigma_q^P > \sigma$, **(G)** the per-entry diagnosis with preferred alternatives, and **(H)** the one-shot repair for flagged plan graphs, with the lowest-score candidate returned.

3.2 CONDITIONAL AND DISCRETE GRAPH DIFFUSION MODEL

Given the ground truth plan graphs $\mathcal{G}^P$ and their query embeddings, we train a graph diffusion model to learn the joint distribution of node features and edges within ground truth plan graphs, conditioned on the query that generated them. As with other graph diffusion models (Jo et al., 2022; Vignac et al., 2023a), training relies on a *forward* process, which progressively corrupts a ground truth plan graph into noise, followed by a *reverse* process, in which a graph denoiser (e.g., a graph neural network or a graph transformer) learns to recover the clean plan graph from its noised version and the associated query (Ho et al., 2020). Both processes evolve over a fixed number of discrete time steps T, and, since node and edge attributes are categorical by construction, the denoiser is trained as a classifier predicting the categorical attributes of the nodes and edges in the plan graph (Austin et al., 2021). Additional background on discrete graph diffusion models is provided in Appendix B.

Forward process. Let $\mathbf{Q}_X^{(t)}$ and $\mathbf{Q}_A^{(t)}$ denote the transition matrices of, respectively, node and edge attributes at time step t, with entries: $\mathbf{Q}_X^{(t)}[i,j] = q\big(\mathbf{x}^{(t)} = \textsc{OneHot}(j) \mid \mathbf{x}^{(t-1)} = \textsc{OneHot}(i)\big)$ and $\mathbf{Q}_A^{(t)}[i,j] = q\big(\mathbf{a}^{(t)} = \textsc{OneHot}(j) \mid \mathbf{a}^{(t-1)} = \textsc{OneHot}(i)\big)$, where $\textsc{OneHot}(\cdot)$ is the function returning the one-hot encoding of a class, so that $\mathbf{Q}_X^{(t)}[i,j]$ is the probability that a node entry of class i moves to class j in one step, and analogously for $\mathbf{Q}_A^{(t)}$. To design $\mathbf{Q}_X^{(t)}$ and $\mathbf{Q}_A^{(t)}$, we follow the approach by Vignac et al. (2023a) and choose the two matrices so that the process converges to the empirical marginal distribution of node and edge classes observed in the training plan graphs. This choice is relevant in our setting, since the node and edge marginals are heavily imbalanced: the

empty class should (in principle) dominate the node marginal, as most queries require fewer than d tools, while the absence class likely dominates the edge marginal, as a plan graph is typically far sparser than the $d \times (d-1)$ ordered pairs it potentially contains. Concretely, corrupting a node entry at time step t amounts to sampling a new class from the row of $\mathbf{Q}_X^{(t)}$ indexed by its current class. For instance, a slot holding tool τ_i might, with some probability, be resampled into another tool τ_j, into the empty class $\varnothing$, or into the out-of-catalog class `unk`. As t grows, this sampling is increasingly dominated by the empty class, reflecting its higher probability in the node marginal. Corrupting an edge entry works analogously: a pair (i, j) currently holding an edge might be resampled into the absence class, or a pair currently without an edge might be resampled into one, again with the absence class becoming increasingly likely as t grows, consistently with the edge marginal. Ultimately, by applying the cumulative transition matrices column by column to the clean plan graph, we corrupt it as:

$$q\Big(\mathcal{G}_q^{P^{(t)}} \mid \mathcal{G}_q^P\Big) = \Big(\mathbf{X}_q^P\, \overline{\mathbf{Q}}_X^{(t)},\ \mathbf{A}_q^P\, \overline{\mathbf{Q}}_A^{(t)}\Big), \tag{1}$$

with $\overline{\mathbf{Q}}_X^{(t)} = \mathbf{Q}_X^{(1)} \cdots \mathbf{Q}_X^{(t)}$ and $\overline{\mathbf{Q}}_A^{(t)} = \mathbf{Q}_A^{(1)} \cdots \mathbf{Q}_A^{(t)}$. Since the plan graph is directed and its adjacency tensor is asymmetric, we never symmetrize $\mathbf{A}_q^P$ after corruption, and the diagonal of $\mathbf{A}_q^{P^{(t)}}$ is kept at the absence class throughout.

Reverse process. The reverse process is performed by a denoiser network $f_\theta(\cdot)$, implemented as a graph transformer (Dwivedi and Bresson, 2020) with L layers, following the architecture of Vignac et al. (2023a) for discrete graph diffusion. Given the noised plan graph $\mathcal{G}_q^{P^{(t)}} = (\mathbf{X}_q^{P^{(t)}}, \mathbf{A}_q^{P^{(t)}})$, node and edge entries are first embedded into initial node representations $\mathbf{u}_{X,0}[i]$ and edge representations $\mathbf{u}_{A,0}[i, j]$, while a global vector $\mathbf{u}_{y,0}$ is initialized from the query embedding $\mathbf{H}_q$ and an embedding of the timestep t. At the last layer, two linear heads, $\mathbf{W}_X$ for nodes and $\mathbf{W}_A$ for edges, map the resulting node and edge representations, $\mathbf{u}_{X,L-1}[i]$ and $\mathbf{u}_{A,L-1}[i, j]$, to distributions over the clean classes:

$$p_\theta\big(\mathbf{X}_q^P[i] = c \mid \mathcal{G}_q^{P^{(t)}}, t, \mathbf{H}_q\big) = \mathrm{softmax}\big(\mathbf{W}_X\, \mathbf{u}_{X,L-1}[i]\big)_c, \qquad c \in \mathcal{T}', \tag{2}$$

$$p_\theta\big(\mathbf{A}_q^P[i, j] = c \mid \mathcal{G}_q^{P^{(t)}}, t, \mathbf{H}_q\big) = \mathrm{softmax}\big(\mathbf{W}_A\, \mathbf{u}_{A,L-1}[i, j]\big)_c, \qquad c \in \{0, 1\},\ i \neq j. \tag{3}$$

Following Austin et al. (2021), $f_\theta(\cdot)$ is trained to directly predict, for every node and edge, the distribution over their clean class at $t = 0$, which we later reinterpret, at inference time, as a confidence score for each of the planner's choices. Combined with the forward posterior, known in closed form, this prediction defines each reverse step from t to $t-1$, and a plan graph is generated from pure noise by iterating it from $t = T$ down to 1. Additionally, and inspired by classifier-free guidance (Ho and Salimans, 2022), during training we replace $\mathbf{H}_q$ with a learned null vector with probability p_{drop}, a form of conditioning dropout that regularizes the denoiser. At inference time, DIGNOSIS uses the query-conditioned prediction only, and the unconditional prediction is never mixed in.

Loss function. Given a ground truth plan graph $\mathcal{G}_q^P$ and its query embedding $\mathbf{H}_q$, we sample a timestep $t \sim \textsc{Uniform}(\{1, \ldots, T\})$ and the corresponding noised plan graph $\mathcal{G}_q^{P^{(t)}} \sim q\Big(\mathcal{G}_q^{P^{(t)}} \mid \mathcal{G}_q^P\Big)$ through the forward process, and train the denoising function $f_\theta(\cdot)$ to recover the original node and edge attributes by categorical cross-entropy:

$$\mathcal{L}(\theta) = \mathbb{E}\left[-\frac{1}{d}\sum_{i=1}^{d} \log p_\theta\big(\mathbf{X}_q^P[i] \mid \mathcal{G}_q^{P^{(t)}}, t, \mathbf{H}_q\big) - \frac{\lambda}{d(d-1)} \sum_{i \neq j} \log p_\theta\big(\mathbf{A}_q^P[i, j] \mid \mathcal{G}_q^{P^{(t)}}, t, \mathbf{H}_q\big)\right], \tag{4}$$

where the expectation is taken over queries, timesteps, and the resulting noised plan graphs, while $\lambda > 1$ upweights the edge term to ensure sparse edges in the ground truth plan graphs still contribute to the graph diffusion model training procedure.

3.3 VERIFICATION AND REPAIR OF PLAN GRAPHS

At inference time, DIGNOSIS receives a query q, a candidate plan graph $\hat{\mathcal{G}}_q^P = (\hat{\mathbf{X}}_q^P, \hat{\mathbf{A}}_q^P)$ produced by a task planner, and the tool catalog $\mathcal{T}$, and returns three fundamental signals to verify

and (optionally) repair the plan graph: (i) a score Σ_q^P, (ii) a diagnosis Δ_q^P, and (iii) a repair Ω_q^P, computed from two $f_\theta(\cdot)$ inference passes (one for the score and the diagnosis, one for the repair). We regard the candidate as the noised plan graph produced at a small, fixed timestep; that is, we directly set $\mathcal{G}_q^{P(t)} := \hat{\mathcal{G}}_q^P$ by convention, with $t = t_{\text{diag}}$ for the score and the diagnosis and $t = t_{\text{repair}}$ for the repair, and feed it to $f_\theta(\cdot)$ to obtain, for every node and edge, the clean class distributions $p_\theta\big(\mathbf{X}_q^P[i] = c \mid \hat{\mathcal{G}}_q^P, t, \mathbf{H}_q\big)$ and $p_\theta\big(\mathbf{A}_q^P[i,j] = c \mid \hat{\mathcal{G}}_q^P, t, \mathbf{H}_q\big)$ defined in the previous section.

Score. The first signal DIGNOSIS provides is a single scalar summarizing how trustworthy a candidate plan graph is overall. Since $f_\theta(\cdot)$ has only ever seen correct plan graphs during training, it assigns high probability to node and edge choices that are consistent with the learned correct data distribution, and low probability to incorrect choices. Therefore, we compute, for every node and edge of the candidate plan graph, the probability that $f_\theta(\cdot)$ assigns to the planner's own choice:

$$p_i = p_\theta\big(\hat{\mathbf{X}}_q^P[i] \mid \hat{\mathcal{G}}_q^P, t_{\text{diag}}, \mathbf{H}_q\big), \qquad p_{ij} = p_\theta\big(\hat{\mathbf{A}}_q^P[i,j] \mid \hat{\mathcal{G}}_q^P, t_{\text{diag}}, \mathbf{H}_q\big) \quad (i \neq j), \tag{5}$$

and aggregate them into a single *brokenness* score, indicating to what extent the predicted plan graph is incorrect according to DIGNOSIS:

$$\Sigma_q^P = -\left(\frac{\alpha}{d}\sum_{i=1}^{d}\log p_i + \frac{1-\alpha}{d(d-1)}\sum_{i\neq j}\log p_{ij}\right), \tag{6}$$

where $\alpha \in [0,1]$ balances the node and edge terms. The negative sign ensures that low probabilities, which correspond to possibly incorrect choices, translate into a high level of brokenness. A plan graph is then flagged as broken when Σ_q^P exceeds a threshold σ; we refer to this test as the *gate*. Since $f_\theta(\cdot)$ assigns each entry a probability below 1 even on correct plan graphs, Σ_q^P has a non-degenerate distribution on them: we compute it on a held out set of ground truth plan graphs, which are correct by construction, and calibrate σ as the ρ-quantile of this distribution, so that only a fraction $1-\rho$ of correct plan graphs scores above σ. In this way, ρ fixes the false-alarm rate of the gate, without requiring any example of broken plan graph.

Diagnosis. While the score suggests the likelihood of a candidate plan graph to be broken, it does not indicate where the source of brokenness lies. The diagnosis addresses this by localizing the judgment made above down to individual nodes and edges, pairing the same probabilities p_i, p_{ij} used for the score, now interpreted as an entrywise confidence, with the alternative choice $f_\theta(\cdot)$ would have preferred in their place:

$$\hat{c}_i = \arg\max_{c\in\mathcal{T}'} p_\theta\big(\mathbf{X}_q^P[i] = c \mid \hat{\mathcal{G}}_q^P, t_{\text{diag}}, \mathbf{H}_q\big), \qquad \hat{c}_{ij} = \arg\max_{c\in\{0,1\}} p_\theta\big(\mathbf{A}_q^P[i,j] = c \mid \hat{\mathcal{G}}_q^P, t_{\text{diag}}, \mathbf{H}_q\big). \tag{7}$$

We collect these quantities into the diagnosis $\Delta_q^P = \{(p_i, \hat{c}_i)\}_{i=1}^d \cup \{(p_{ij}, \hat{c}_{ij})\}_{i\neq j}$, flagging as suspect every entry whose confidence falls below a threshold η, together with the alternative the model prefers in its place. The threshold η is not tuned: we fix $\eta = 0.5$, so that an entry is suspect whenever the planner's choice holds less than half of the probability mass.

Repair. The last signal goes one step further than diagnosing individual entries. While the diagnosis reports where the plan graph is wrong, the repair applies the alternatives the model prefers jointly, producing a corrected plan graph for the same query. Since $f_\theta(\cdot)$ already produces, for every node and edge, a distribution over its most likely clean class, a natural candidate repair is obtained by simply taking the most likely class everywhere, this time reading the candidate at the timestep t_{repair}:

$$\Omega_q^P = \Big(\arg\max_{c\in\mathcal{T}'} p_\theta(\mathbf{X}_q^P[i] = c \mid \hat{\mathcal{G}}_q^P, t_{\text{repair}}, \mathbf{H}_q)\Big)_{i=1}^{d}, \quad \Big(\arg\max_{c\in\{0,1\}} p_\theta(\mathbf{A}_q^P[i,j] = c \mid \hat{\mathcal{G}}_q^P, t_{\text{repair}}, \mathbf{H}_q)\Big)_{i\neq j}. \tag{8}$$

We call Ω_q^P the *one-shot* repair, since a single pass of $f_\theta(\cdot)$ produces it, instead of the full reverse chain described in Section 3.2. We do not apply the repair unconditionally to every candidate, but only to the plan graphs the gate flags as broken, leaving untouched those whose score does not exceed σ. Since the repair accesses the candidate at a small timestep, it denoises the planner's plan graph rather than generating a new one from scratch; thus, entries the model is confident about are preserved, and only the suspect ones change.

3.4 VERIFICATION PIPELINE

Given the three signals described above, our proposed verification pipeline works as follows.

1. **Score and gate.** DIGNOSIS computes Σ_q^P for the candidate plan graph; if the score does not exceed σ, the plan graph is returned unchanged.
2. **Candidates.** For a flagged plan graph, DIGNOSIS builds a small set of candidates: the plan graph itself, the one-shot repair Ω_q^P, applied either to the whole plan graph or to the suspect entries only, and an *LLM edit*, obtained by asking an LLM to correct the plan graph given the query, the tool catalog and the diagnosis Δ_q^P; that is, the suspect nodes and edges with the alternatives the model prefers (the prompt is reported in Appendix C.1).
3. **Selection.** Every candidate is scored with Σ_q^P and the one with the lowest brokenness is returned. Since the original plan graph is always among the candidates, a repair is accepted only if it lowers the brokenness score of the original (self-acceptance), so that the final decision always rests with DIGNOSIS rather than with the LLM.
4. **Chain rewriting.** As the benchmarks we use only contain chain-shaped reference plan graphs, every plan graph returned by the pipeline is rewritten as the chain implied by its tool order; this is a deterministic step that is applied identically to the graph-based verifier we compare with (Section 4).
5. **Calibration.** The threshold quantile ρ, the repair timestep t_{repair} and the repair policy (applying Ω_q^P to the whole plan graph, to the suspect entries only, or not at all) are calibrated, per planner and benchmark, on the planner's own plan graphs for a held out set of validation queries, never on the test set; for planners without validation plan graphs of their own, we use those of DIRECT with the same backbone.

The LLM edit is part of the pipeline evaluated in RQ1 from Section 5, where it is produced by the same LLM the other verifiers use, and RQ2 from the same section measures its contribution by removing it. The complete algorithms describing the training and inference procedures of DIGNOSIS are reported in Appendix C (Algorithms 1–2).

4 EXPERIMENTAL SETUP

We summarize the protocol shared by every method; Appendix D details datasets, planners, verifiers, backbones and hyperparameters, and Appendix D.5 the evaluation pipeline.

Datasets. The benchmarks are HuggingFace and Multimedia from TaskBench (Shen et al., 2024) (23 and 40 tools) and UltraTool (Huang et al., 2024) (260 tools), each with a fixed test split of 500 queries and their reference plan graphs, used only for evaluation; DIGNOSIS is trained on the correct query–plan graph pairs of each training pool, and 300 held-out validation queries per benchmark select the configuration of every verifier (Table 4).

Task planners and LLMs. The planners are of four types: DIRECT, the single-call prompting protocol of TaskBench; REACT (Yao et al., 2023a), an iterative thought–action–observation loop; GNN4PLAN (Wu et al., 2024), a training-free graph-augmented planner over the DIRECT decomposition; GTOOL (Chen et al., 2026), a soft-prompt planner on a frozen Vicuna-7B (Chiang et al., 2023). Planners and LLM-based verifiers run on two open-source backbones served locally, Qwen3-235B-A22B-Instruct (Team, 2025) in the main paper and Mistral-Small-3.2-24B-Instruct in Appendix E.5, so that, for a given backbone, every verifier is evaluated on identical plan graphs.

Verifiers. Every verifier receives the query, the plan graph and the catalog, without ever accessing the reference plan graph; then, it returns its possibly repaired plan graph. We consider the following verifiers: SELF-REFINE (Madaan et al., 2023) rewrites the plan graph after a self-critique; VERICODER (Ho et al., 2025) checks each subtask against its tool and issues one fix call; VERIPLAN (Lee et al., 2025) runs deterministic checks (catalog membership, edge admissibility, acyclicity) with one corrective call on violation; GNNVERIFIER (Hao et al., 2026), the closest verifier to ours, trains a GNN on hand-corrupted plan graphs and leverages an LLM to edit the flagged ones (refer to Appendix D.3 for further details). DIGNOSIS is trained once per benchmark and shared by every planner and backbone; only ρ, t_{repair} and the repair policy are calibrated per planner on its validation plan graphs (Section 3.4), and its LLM edit is produced by the same backbone as the other verifiers.

Table 1: Verification and repair of the plan graphs of four planners with Qwen3. Metrics are n-F1, l-F1, and Accuracy on the three datasets, with mean and standard deviation over three random repetitions. Bold and underline mark the best and second best value across planners and metrics. The first row of each block represents the raw planner without any verifier.

	HuggingFace			Multimedia			UltraTool		
Method	n-F1↑	l-F1↑	Acc↑	n-F1↑	l-F1↑	Acc↑	n-F1↑	l-F1↑	Acc↑
Direct									
No verifier	82.23	56.91	33.00	90.18	73.16	54.80	71.65	42.47	29.40
SELF-REFINE	75.89±0.11	45.74±0.20	21.40±0.00	88.99±0.17	70.92±0.25	51.53±0.64	71.99±0.00	41.98±0.00	29.20±0.00
VERICODER	80.24±0.12	55.06±0.06	31.53±0.12	89.77±0.12	72.73±0.27	54.27±0.42	72.13±0.00	43.11±0.00	29.80±0.00
VERIPLAN	82.12±0.02	57.05±0.00	33.20±0.00	**90.46**±0.04	73.87±0.13	55.13±0.12	72.35±0.00	46.27±0.00	30.80±0.00
GNNVERIFIER	74.19±8.49	50.93±10.29	32.67±9.29	84.85±4.54	68.59±6.68	53.33±10.02	71.69±0.16	43.68±0.24	30.33±0.58
DIGNOSIS	**82.65**±0.16	**61.87**±0.16	**43.20**±0.20	90.17±0.09	**75.67**±0.10	**64.20**±0.20	**77.07**±0.79	**49.95**±0.63	**36.00**±1.06
ReAct									
No verifier	82.29	62.61	45.40	90.77	77.16	65.60	75.51	48.82	41.68
SELF-REFINE	75.38±0.19	46.86±0.23	24.80±0.35	89.17±0.34	71.87±0.59	51.20±0.69	75.93±0.28	48.21±0.27	40.61±0.12
VERICODER	80.27±0.03	58.93±0.12	41.13±0.23	89.99±0.14	76.59±0.12	64.93±0.12	75.50±0.07	48.38±0.06	41.08±0.00
VERIPLAN	81.90±0.04	62.42±0.04	45.20±0.00	**90.81**±0.00	**77.66**±0.03	**66.00**±0.00	75.63±0.04	50.23±0.03	42.42±0.12
GNNVERIFIER	65.77±1.24	42.38±1.13	26.00±1.00	77.73±3.02	58.56±4.29	39.33±2.52	75.79±1.38	50.55±1.84	42.33±2.52
DIGNOSIS	**82.92**±0.33	**63.00**±0.25	**45.87**±0.31	90.54±0.49	76.88±0.70	64.87±0.95	**79.09**±0.70	**54.18**±1.03	**45.29**±0.92
GNN4Plan									
No verifier	76.60	54.21	35.60	87.24	71.73	58.60	70.39	44.54	29.80
SELF-REFINE	76.84±0.00	47.79±0.03	24.60±0.00	88.70±0.01	71.35±0.04	53.47±0.12	71.67±0.00	43.79±0.00	29.60±0.00
VERICODER	81.05±0.00	59.53±0.00	39.60±0.00	**89.80**±0.00	**75.62**±0.00	**63.60**±0.00	72.92±0.00	44.63±0.00	31.00±0.00
VERIPLAN	76.60±0.00	54.21±0.00	35.60±0.00	87.24±0.00	71.73±0.00	58.60±0.00	70.39±0.00	44.54±0.00	29.80±0.00
GNNVERIFIER	64.69±0.79	39.25±0.74	21.00±1.00	76.55±2.66	57.37±3.77	38.00±3.00	70.32±0.08	44.77±0.18	29.67±0.58
DIGNOSIS	**82.17**±0.54	**61.27**±0.96	**42.40**±1.44	89.11±0.11	74.45±0.19	62.00±0.40	**78.07**±0.37	**50.09**±1.03	**35.13**±0.42
GTool									
No verifier	75.77	50.69	29.80	76.62	53.63	33.80	81.29	61.14	53.31
SELF-REFINE	77.78±0.22	50.54±0.62	27.20±0.35	88.92±0.12	71.29±0.11	51.07±0.46	81.28±0.00	56.48±0.00	47.29±0.00
VERICODER	81.51±0.03	59.77±0.09	40.33±0.23	**89.86**±0.07	**75.18**±0.33	**62.27**±0.58	80.64±0.00	57.70±0.00	46.09±0.00
VERIPLAN	80.21±0.06	58.20±0.12	36.53±0.12	82.65±0.01	64.54±0.00	47.20±0.00	**86.76**±0.00	**67.30**±0.00	**58.32**±0.00
GNNVERIFIER	66.27±6.41	38.79±7.28	19.33±6.66	70.32±5.48	46.65±6.62	25.67±8.14	81.35±0.07	61.14±0.00	53.00±0.00
DIGNOSIS	**81.71**±0.46	**59.95**±0.45	**41.07**±0.70	86.93±0.32	70.31±0.20	56.00±1.40	81.43±0.26	59.90±0.35	51.10±0.80

Metrics. Node-F1 (**n-F1**) on the tool sets, link-F1 (**l-F1**) on the directed edge sets, and Accuracy (**Acc**), the fraction of test queries matching the reference on both; non-deterministic methods use three seeds (of the denoiser, for DIGNOSIS), we report mean and standard deviation, and bold and underline mark the best and second best value of each planner and column.

5 RESULTS AND DISCUSSION

Our research questions are: **RQ1)** Does DIGNOSIS verify and repair the plan graphs of different planners more effectively than existing LLM- and graph-based verifiers? **RQ2)** Which components of DIGNOSIS account for its performance? **RQ3)** Is a verifier still useful when the planner itself can be fine-tuned on the same correct plan graphs? Additional experimental results are in Appendix E.

Overall performance (RQ1). Table 1 reports every verifier on top of every planner with the Qwen3 backbone. DIGNOSIS improves the raw plan graph in 10 of 12 configurations, most where the planner is weakest (up to +22 Accuracy points on GTOOL–Multimedia). Since the repair mostly rewires the tools the planner already chose, the gains concentrate on l-F1 rather than n-F1 (e.g., +5.0 against +0.4 on DIRECT–HuggingFace), except on UltraTool, whose large catalog makes wrong and out-of-catalog tools more frequent and where n-F1 improves as well (up to +7.7). Among verifiers, DIGNOSIS achieves the best 3-benchmark mean of every metric on three of the four planners, and the second best on GTOOL, behind VERICODER. The other verifiers either rarely intervene (VERIPLAN), help weak planners but degrade strong ones (VERICODER), or degrade the raw plan graph in most configurations (SELF-REFINE, GNNVERIFIER, the latter also with high variance across runs). DIGNOSIS falls behind in two situations (Appendix F): on Multimedia, where near-synonymous tools make broken plan graphs harder to detect, and on already strong planners, where the gate occasionally flags correct plan graphs. Overall, DIGNOSIS improves the raw plan graph more often and by a larger average margin than any other verifier (+6.4 Accuracy points, against +2.3 for VERIPLAN and +2.9 for VERICODER).

Ablation study (RQ2). We run the ablation on GNN4PLAN and REACT with the Qwen3 backbone, since both already produce chain-shaped plan graphs: chain rewriting has no effect on them, so each component can be measured in isolation. On DIRECT, by contrast, chain rewriting alone accounts for most of the gain on HuggingFace and Multimedia. In the pipeline (Section 3.4), we remove the score gate (every plan graph is repaired), the LLM edit (the one-shot repair is the only alternative), the diagnosis Δ_q^P from the prompt (the LLM is only told the plan graph was flagged), and the score-based selection (a random candidate is returned). In the denoiser (Section 3.2), retrained per variant with the same three seeds, we remove $\mathbf{H}_q$, both $\mathbf{H}_q$ and the subtask texts (unconditional denoiser), edge direction (symmetric $\mathbf{A}_q^P$), and the marginal noise model (uniform transitions). Table 2 reports results for GNN4PLAN on HuggingFace, where every removal lowers every metric. Query conditioning is the most important component, followed by the score-based selection and the LLM edit, a ranking that holds across all three benchmarks. On REACT, whose plan graphs are wrong less often, the gate matters most, as it protects plan graphs that are already correct.

Table 2: Ablation study on the verification pipeline and the denoiser for HuggingFace, GNN4PLAN, Qwen3.

Variant	n-F1 ↑	l-F1 ↑	Acc ↑
No verifier	76.60	54.21	35.60
Full method	**82.17**±**0.54**	**61.27**±**0.96**	**42.40**±**1.44**
Verification pipeline			
w/o score gate	81.95±0.12	60.50±0.27	42.07±0.58
w/o LLM edit	80.77±0.48	58.40±0.24	39.53±0.42
w/o Δ in prompt	82.12±0.50	61.19±0.89	42.27±1.29
random selection	79.96±0.08	57.89±0.68	38.67±0.50
Denoiser			
w/o query cond.	81.12±0.56	60.23±0.65	41.00±1.00
unconditional	77.19±0.22	54.92±0.31	37.13±0.23
symmetric edges	81.76±1.08	60.57±1.09	41.93±1.62
uniform noise	81.54±1.29	60.04±1.55	40.87±1.81

Verification on top of a fine-tuned planner (RQ3). To answer RQ3, we add SFT-DIRECT, a LoRA fine-tuning (Hu et al., 2022) of Qwen3-14B on the training pools of Section 4, with the DIRECT prompt and without the 300 validation queries, which calibrate the verifiers as for the other planners (Appendix D.2). Table 3 reports results for Multimedia; the other benchmarks and the Mistral backbone are in Appendices E.1 and E.5. SFT-DIRECT is by far the strongest planner of our study (55.6 to 70.0 Accuracy points, against at most 65.6 in Table 1). DIGNOSIS recognizes, through its calibration, that its plan graphs are largely correct and abstains on every benchmark and seed, changing them only through the chain rewriting shared with GNNVERIFIER (+1.0 Accuracy and −0.5 l-F1). VERIPLAN abstains as well, whereas the verifiers that rewrite every plan graph degrade it, losing up to 21 Accuracy points. Moreover, fine-tuning requires access to the planner's weights and one adapter per planner and benchmark, whereas a single DIGNOSIS model, trained in one to two minutes (Appendix E.2), serves every planner: its value therefore concentrates on prompted planners and closed backbones, where fine-tuning is not an option.

Table 3: Verifiers on top of SFT-DIRECT (LoRA on Qwen3-14B, trained on the same correct plan graphs as DIGNOSIS) on Multimedia, Qwen3 backbone.

Method	n-F1↑	l-F1↑	Acc↑
No verifier	**92.96**	80.29	70.00
SELF-REFINE	89.41±0.15	70.48±0.76	49.33±0.99
VERICODER	92.39±0.12	79.06±0.09	67.67±0.12
VERIPLAN	92.94±0.00	**80.52**±**0.00**	70.00±0.00
GNNVERIFIER	83.38±5.66	65.58±7.69	49.00±7.94
DIGNOSIS	**92.96**±**0.00**	79.78±0.00	**71.00**±**0.00**

6 CONCLUSION AND FUTURE WORK

In this work, we recast the verification and repair of task plan graphs as conditional graph denoising, and introduced DIGNOSIS, a discrete graph diffusion model trained only on correct plan graphs and conditioned on their queries. From a single denoiser, DIGNOSIS derives a score, a diagnosis and a repair, while an LLM merely proposes a repair candidate, accepted only if it lowers the brokenness score. Across three datasets, four task planners of different nature and two LLM backbones, DIGNOSIS improves the raw plan graph more often and by larger margins than four existing verifiers, and, on a fine-tuned planner, recognizes that its plan graphs are largely correct and abstains from repairing them. Future directions of this work will involve the extension of DIGNOSIS to DAG-shaped and longer-horizon plans, propose to condition node classes on tool descriptions to handle catalog shift and near-synonymous tools, and try to use its score and diagnosis to guide planning rather than only verify its output.

AI USE STATEMENT

Large language models were employed only as writing assistants, to polish and refine the language of the manuscript. They played no part in the design of the methodology, the implementation, the execution of the experiments, or the analysis of the results. The authors carefully reviewed, verified, and edited all content, and take full responsibility for the entire work.

ETHICS STATEMENT

Our study relies exclusively on publicly available task planning benchmarks and open-source language models, used in compliance with their licenses; it involves no human participants and no personal or sensitive data. DIGNOSIS is designed to detect and repair errors in plans generated by LLM-based planners, such as hallucinated tools or missing dependencies, and may therefore contribute to more reliable agentic systems. Nonetheless, its verification is probabilistic and cannot guarantee the correctness of every plan: in high-stakes applications, it should complement rather than replace human oversight. Beyond these considerations, we do not foresee specific ethical concerns related to this work.

REPRODUCIBILITY STATEMENT

To support the replication of our findings, the appendix documents every component of the experimental pipeline. The training and inference procedures of DIGNOSIS are formalized in Appendix C, together with the loss function and the exact prompt used for the LLM edit (Appendix C.1). The benchmarks and their splits are described in Appendix D.1, the task planners and the verifiers we compare against in Appendices D.2 and D.3, and the language models, with their versions and quantization, in Appendix D.4. Appendix D.5 details the evaluation protocol, including the calibration of the thresholds and the number of repetitions, while Appendix D.6 reports all hyperparameters, distinguishing the values fixed in advance from those selected on validation. Source code, the decision log of the study, and the scripts that generate every table of the paper are available at an anonymous repository (Appendix G), and will be made publicly available upon acceptance.

REFERENCES

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. Taskbench: Benchmarking large language models for task automation. In *NeurIPS*, 2024.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *ICLR*. OpenReview.net, 2024.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving AI tasks with chatgpt and its friends in hugging face. In *NeurIPS*, 2023.

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 *ICLR*. OpenReview.net, 2023a.

Xixi Wu, Yifei Shen, Caihua Shan, Kaitao Song, Siwei Wang, Bohang Zhang, Jiarui Feng, Hong Cheng, Wei Chen, Yun Xiong, and Dongsheng Li. Can graph learning improve planning in llm-based agents? In *NeurIPS*, 2024.

Wenjie Chen, Di Yao, Wenbin Li, Xuying Meng, Chang Gong, and Jingping Bi. GTool: Graph enhanced tool planning with large language model. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=bn47cqGQ7l.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. Can language models solve graph problems in natural language? In *NeurIPS*, 2023.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models. In *ICLR*. OpenReview.net, 2024.

Chen Kang and Kunyan Li. A lifecycle-oriented survey of llm-based agents: Workflow, graph augmentation, and future directions. *Neurocomputing*, 704:134838, 2026. doi: 10.1016/j.neucom.2026.134838.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *NeurIPS*, 2023.

Chia-Tung Ho, Haoxing Ren, and Brucek Khailany. Verilogcoder: Autonomous verilog coding agents with graph-based planning and abstract syntax tree (ast)-based waveform tracing tool. In *AAAI*, pages 300–307. AAAI Press, 2025.

Christine P. Lee, David Porfirio, Xinyu Jessica Wang, Kevin Chenkai Zhao, and Bilge Mutlu. Veriplan: Integrating formal verification and llms into end-user planning. In *CHI*, pages 247:1–247:19. ACM, 2025.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models - A critical investigation. In *NeurIPS*, 2023.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. Position: Llms can't plan, but can help planning in llm-modulo frameworks. In *ICML*, volume 235 of *Proceedings of Machine Learning Research*, pages 22895–22907. PMLR / OpenReview.net, 2024.

Yu Hao, Qiuyu Wang, Cheng Yang, Yawen Li, Zhiqiang Zhang, and Chuan Shi. Gnnverifier: Graph-based verifier for LLM task planning. *CoRR*, abs/2603.14730, 2026.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *NeurIPS*, pages 17981–17993, 2021.

Clément Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. In *ICLR*. OpenReview.net, 2023a.

Eric Hanchen Jiang, Levina Li, Frank Wan, Xiao Liang, Sophia Yin, Yuchen Wu, Xinfeng Li, Yizhou Sun, Wei Wang, Kai-Wei Chang, and Ying Nian Wu. Dynamic generation of multi LLM agents communication topologies with graph diffusion models. In *ACL (1)*, pages 38042–38060. Association for Computational Linguistics, 2026.

Puja Trivedi, Ryan A. Rossi, David Arbour, Tong Yu, Franck Dernoncourt, Sungchul Kim, Nedim Lipka, Namyong Park, Nesreen K. Ahmed, and Danai Koutra. Editing partially observable networks via graph diffusion models. In *ICML*, volume 235 of *Proceedings of Machine Learning Research*, pages 48608–48629. PMLR / OpenReview.net, 2024.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *CoRR*, abs/2212.03533, 2022.

Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 10362–10383. PMLR, 2022.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *CoRR*, abs/2012.09699, 2020.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *CoRR*, abs/2207.12598, 2022.

Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, Xin Jiang, Ruifeng Xu, and Qun Liu. Planning, creation, usage: Benchmarking llms for comprehensive tool utilization in real-world complex scenarios. In *ACL (Findings)*, volume ACL 2024 of *Findings of ACL*, pages 4363–4400. Association for Computational Linguistics, 2024.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality. LMSYS Org blog, `https://lmsys.org/blog/2023-03-30-vicuna/`, 2023.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *ICLR*. OpenReview.net, 2022.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*, 2023b.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *AAAI*, pages 17682–17690. AAAI Press, 2024.

Xukun Liu, Zhiyuan Peng, Xiaoyuan Yi, Xing Xie, Lirong Xiang, Yuchen Liu, and Dongkuan Xu. Toolnet: Connecting large language models with massive tools via tool graph. *CoRR*, abs/2403.00839, 2024a.

Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Ziheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, and Wenhai Wang. Controlllm: Augment language models with tools by searching on graphs. In *ECCV (12)*, volume 15070 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 2024b.

Linfeng Gao, Yaoxiang Wang, Minlong Peng, Jialong Tang, Yuzhe Shang, Mingming Sun, and Jinsong Su. Tool graph retriever: Exploring dependency graph-based tool retrieval for large language models. *CoRR*, abs/2508.05152, 2025.

Bhrij Patel, Davide Belli, Amir Jalalirad, Maximilian Arnold, Aleksandr Ermolov, and Bence Major. Dynamic tool dependency retrieval for lightweight function calling. In *ACL (Findings)*, pages 33649–33672. Association for Computational Linguistics, 2026.

Fangru Lin, Emanuele La Malfa, Valentin Hofmann, Elle Michelle Yang, Anthony G. Cohn, and Janet B. Pierrehumbert. Graph-enhanced large language models in asynchronous plan reasoning. In *ICML*, volume 235 of *Proceedings of Machine Learning Research*, pages 30108–30134. PMLR / OpenReview.net, 2024.

Shiqi Zhang, Xinbei Ma, Zouying Cao, Zhuosheng Zhang, and Hai Zhao. Plan-over-graph: Towards parallelable LLM agent schedule. *CoRR*, abs/2502.14563, 2025.

Xinyi Gao, Xinyu Ren, Junliang Yu, Tong Chen, Quoc Viet Hung Nguyen, and Hongzhi Yin. GRAFT: graph-tokenized llms for tool planning. *CoRR*, abs/2605.11706, 2026.

Xiuhui You, Jiayi Luo, Zichao Shen, Qingyun Sun, and Ziwei Zhang. Toollift: Lifting tool-specific trajectories into function-level graphs for generalizable tool planning. *CoRR*, abs/2608.03468, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. In *NeurIPS*, 2024.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. Gpt4tools: Teaching large language model to use tools via self-instruction. In *NeurIPS*, 2023.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. Toolgen: Unified tool retrieval and calling via generation. In *ICLR*. OpenReview.net, 2025.

Ananth Hariharan, Vardhan Dongre, Dilek Hakkani-Tür, and Gokhan Tur. Plan verification for llm-based embodied task completion agents. *CoRR*, abs/2509.02761, 2025.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *ICLR*. OpenReview.net, 2024a.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: large language models can self-correct with tool-interactive critiquing. In *ICLR*. OpenReview.net, 2024.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Yuanzhuo Wang, and Jian Guo. A survey on llm-as-a-judge. *CoRR*, abs/2411.15594, 2024.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Lingpeng Kong, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators. In *ACL (1)*, pages 9440–9450. Association for Computational Linguistics, 2024.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: empowering large language models with optimal planning proficiency. *CoRR*, abs/2304.11477, 2023.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. In *NeurIPS*, 2023.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *ICLR*. OpenReview.net, 2024.

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-Tau Yih, Sida I. Wang, and Xi Victoria Lin. LEVER: learning to verify language-to-code generation with execution. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 26106–26128. PMLR, 2023.

William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. Self-critiquing models for assisting human evaluators. *CoRR*, abs/2206.05802, 2022.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct. In *ICLR*. OpenReview.net, 2023.

Michihiro Yasunaga and Percy Liang. Graph-based, self-supervised program repair from diagnostic feedback. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 10799–10808. PMLR, 2020.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *NeurIPS*, pages 11895–11907, 2019.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *ICLR*. OpenReview.net, 2021.

Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. Permutation invariant graph generation via score-based generative modeling. In *AISTATS*, volume 108 of *Proceedings of Machine Learning Research*, pages 4474–4484. PMLR, 2020.

Kilian Konstantin Haefeli, Karolis Martinkus, Nathanaël Perraudin, and Roger Wattenhofer. Diffusion models for graphs benefit from discrete state spaces. *CoRR*, abs/2210.01549, 2022.

Zhe Xu, Ruizhong Qiu, Yuzhong Chen, Huiyuan Chen, Xiran Fan, Menghai Pan, Zhichen Zeng, Mahashweta Das, and Hanghang Tong. Discrete-state continuous-time diffusion for graph generation. In *NeurIPS*, 2024.

Xiaohui Chen, Jiaxing He, Xu Han, and Liping Liu. Efficient and degree-guided graph generation via discrete diffusion modeling. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 4585–4610. PMLR, 2023.

Lingkai Kong, Jiaming Cui, Haotian Sun, Yuchen Zhuang, B. Aditya Prakash, and Chao Zhang. Autoregressive diffusion model for graph generation. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 17391–17408. PMLR, 2023.

Lingxiao Zhao, Xueying Ding, and Leman Akoglu. Pard: Permutation-invariant autoregressive diffusion for graph generation. In *NeurIPS*, 2024.

Manuel Madeira, Clément Vignac, Dorina Thanou, and Pascal Frossard. Generative modelling of structurally constrained graphs. In *NeurIPS*, 2024.

Mengchun Zhang, Maryam Qamar, Taegoo Kang, Yuna Jung, Chenshuang Zhang, Sung-Ho Bae, and Chaoning Zhang. A survey on graph diffusion models: Generative AI in science for molecule, protein and material. *CoRR*, abs/2304.01565, 2023.

Hongyang Chen, Can Xu, Lingyu Zheng, Qiang Zhang, and Xuemin Lin. Diffusion-based graph generative methods. *IEEE Trans. Knowl. Data Eng.*, 36(12):7954–7972, 2024b.

Yuntao Shou, Wei Ai, Tao Meng, and Keqin Li. Graph diffusion models: A comprehensive survey of methods and applications. *Comput. Sci. Rev.*, 59:100854, 2026.

Han Huang, Leilei Sun, Bowen Du, and Weifeng Lv. Conditional diffusion based on discrete graph structures for molecular graph generation. In *AAAI*, pages 4302–4311. AAAI Press, 2023.

Clément Vignac, Nagham Osman, Laura Toni, and Pascal Frossard. Midi: Mixed graph and 3d denoising diffusion for molecule generation. In *ECML/PKDD (2)*, volume 14170 of *Lecture Notes in Computer Science*, pages 560–576. Springer, 2023b.

Zhiqing Sun and Yiming Yang. DIFUSCO: graph-based diffusion solvers for combinatorial optimization. In *NeurIPS*, 2023.

Sohyun An, Hayeon Lee, Jaehyeong Jo, Seanie Lee, and Sung Ju Hwang. Diffusionnag: Predictor-guided neural architecture generation with diffusion models. In *ICLR*. OpenReview.net, 2024.

Shreyas Kapur, Erik Jenner, and Stuart Russell. Diffusion on syntax trees for program synthesis. In *ICLR*. OpenReview.net, 2025.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2256–2265. JMLR.org, 2015.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, pages 3942–3951. AAAI Press, 2018.

APPENDIX

TABLE OF CONTENTS

A Related work **15**

B Background on Discrete Graph Diffusion Models **17**

C Additional Methodological Details of DiGnosis **18**

C.1 Prompt of the LLM edit 18

D Experimental Setup **20**

D.1 Datasets 20

D.2 Task planners 21

D.3 Plan verifiers 21

D.4 Large language models 22

D.5 Evaluation pipeline 22

D.6 Hyperparameters and reproducibility details 23

E Additional Experimental Results **24**

E.1 Verifiers on top of the fine-tuned planner: all benchmarks 24

E.2 Computational time 24

E.3 Sensitivity to the inference-time hyperparameters 25

E.4 Sensitivity to the training-time hyperparameters 25

E.5 Results with the Mistral backbone 26

F Case Studies **28**

F.1 Successful detection and repair 29

F.2 Bad case analysis 29

G Computational Resources and Codebase **30**

H Limitations **30**

A RELATED WORK

LLM task planning. Turning a natural language request into an executable plan graph of tool invocations is a standard capability target for LLM agents: TaskBench (Shen et al., 2024) provides the graph-structured benchmarks and metrics we build on, and UltraTool (Huang et al., 2024) extends the setting to large catalogs with terse tool descriptions. The most direct way to obtain such a plan graph is to prompt an LLM to decompose the request into subtasks and assign a tool to each of them (Shen et al., 2023; 2024), either in a single call or through an interleaved reason-act-observe loop that grows the plan one step at a time (Yao et al., 2023a; Shinn et al., 2023), possibly with tree- or

graph-structured reasoning (Yao et al., 2023b; Besta et al., 2024). Since tools are not independent of one another, a second line of work augments the LLM with the dependencies among tools, encoded in a tool graph derived from historical trajectories (Liu et al., 2024a) or from input-output compatibility (Liu et al., 2024b), and uses it to search the tool space (Liu et al., 2024a;b), retrieve dependent tools (Gao et al., 2025; Patel et al., 2026), or reason over graph-shaped plans (Lin et al., 2024; Zhang et al., 2025). Within this line, GNN4PLAN (Wu et al., 2024) selects the tools with a graph neural network over the subtasks decomposed by an LLM, and GTOOL (Chen et al., 2026) feeds a graph token to a small frozen LLM, an approach recently extended by graph-tokenized and function-level planners (Gao et al., 2026; You et al., 2026). Alternatively, the LLM itself can be specialized on tool-use data (Schick et al., 2023; Patil et al., 2024; Qin et al., 2024; Yang et al., 2023), possibly mapping each tool to a dedicated token (Wang et al., 2025); the LoRA fine-tuned planner of RQ3 (Section 5) is the minimal instance of this approach for a fixed catalog. None of these designs is immune to errors: prompting inherits the limited ability of LLMs to reason over graph structure (Wang et al., 2023; Fatemi et al., 2024), while graph augmentation relies on tool graphs that contain erroneous or missing edges (Chen et al., 2026; Gao et al., 2026; Kang and Li, 2026). We therefore verify planners of all these kinds, and of sharply different strength, since the usefulness of a verifier depends on how good the planner it verifies already is.

Verification and correction of LLM outputs. LLMs struggle to plan autonomously and to verify their own plans (Valmeekam et al., 2023; Kambhampati et al., 2024), which has motivated a growing body of plan verifiers, including for embodied agents (Hariharan et al., 2025). The simplest option lets the same LLM produce natural language feedback on its output and refine it accordingly (Madaan et al., 2023; Shinn et al., 2023; Chen et al., 2024a), possibly grounding the critique on external tools (Gou et al., 2024), in line with the broader use of LLMs as judges of generated content (Gu et al., 2024). LLM-based evaluation, however, suffers from biases, e.g., toward the order or length of the content (Wang et al., 2024), and can be misled by fluent but incorrect step descriptions (Hao et al., 2026). To mitigate this, other works check LLM outputs against explicit models or constraints: LLM-Modulo frameworks pair the LLM with external critics (Kambhampati et al., 2024), the task can be translated into a formal planning language solved by classical planners (Liu et al., 2023; Guan et al., 2023), and VERIPLAN (Lee et al., 2025) model-checks the plan against user-specified rules and feeds the violations back to the LLM for correction. Verification can also target the semantic consistency of the plan, with an LLM assistant checking that each subtask agrees with the task specification, as in the plan verification component of VERILOGCODER (Ho et al., 2025), which we call VERICODER following GNNVERIFIER. Finally, a separate model can be trained to score candidate solutions, on the final answer or step by step (Cobbe et al., 2021; Lightman et al., 2024; Ni et al., 2023), or to critique and correct them (Saunders et al., 2022; Welleck et al., 2023). Graph-based models trained on self-supervised corruptions have been used in this way to repair programs from diagnostic feedback (Yasunaga and Liang, 2020), and, for plan graphs, GNNVERIFIER (Hao et al., 2026) trains a graph neural network to separate correct plan graphs from hand-corrupted ones, and delegates their correction to an LLM. Across these approaches, two properties recur: none of them models the distribution of correct plan graphs directly, as learned verifiers are trained discriminatively against negatives specified in advance, and the model that scores a plan graph is generally not the one that repairs it. DIGNOSIS differs on both counts, and we compare it with one representative of each approach under a shared protocol (Section 4).

Discrete graph diffusion models. Diffusion models learn to reverse a gradual corruption of the data (Ho et al., 2020; Song and Ermon, 2019; Song et al., 2021), and their discrete variants operate directly on categorical states (Austin et al., 2021). On graphs, early score-based models perturbed continuous adjacency matrices (Niu et al., 2020; Jo et al., 2022), while discrete state spaces proved better suited to the categorical nature of nodes and edges (Haefeli et al., 2022). DiGress (Vignac et al., 2023a) diffuses node and edge classes toward their marginal distributions and denoises them with a graph transformer, and discrete graph diffusion has since been extended to continuous time (Xu et al., 2024), to sparse and degree-guided generation (Chen et al., 2023), to autoregressive and permutation-invariant variants (Kong et al., 2023; Zhao et al., 2024), and to hard structural constraints such as planarity or acyclicity (Madeira et al., 2024); we refer to recent surveys for a broader overview (Zhang et al., 2023; Chen et al., 2024b; Shou et al., 2026). Conditional graph diffusion has been used to generate molecules (Huang et al., 2023; Vignac et al., 2023b), solutions of combinatorial optimization problems (Sun and Yang, 2023), neural architectures for a given task (An et al., 2024) and, closest to our setting, communication topologies of LLM-based multi-agent

systems conditioned on the task query (Jiang et al., 2026). Beyond generation, diffusion models have also been used to edit existing structured objects, from programs represented by their syntax trees (Kapur et al., 2025) to partially observed networks, where a trained graph diffusion model removes spurious edges and recovers missing ones (Trivedi et al., 2024). DIGNOSIS shares this editing perspective, but conditions it on a query and complements the repair with a score and a diagnosis: rather than sampling new plan graphs from noise, it reads an existing plan graph as a noisy sample to verify it, localize its errors and repair them.

B BACKGROUND ON DISCRETE GRAPH DIFFUSION MODELS

Diffusion models learn a data distribution by defining a *forward* process that gradually corrupts a clean sample into noise, and training a network to reverse this corruption step by step (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2021). While originally designed for continuous data, where corruption takes the form of additive Gaussian noise, a graph is naturally described by categorical quantities, node types and the presence or absence of edges, for which the notion of additive noise does not directly apply (Haefeli et al., 2022). Austin et al. (2021) address this by defining diffusion directly over discrete state spaces, corrupting a categorical variable through a sequence of transition matrices rather than through additive noise, and Vignac et al. (2023a) adapt this construction to graphs, applying it independently to every node and edge. We recall this construction below, using the same notation later specialized to plan graphs in Section 3.2.

Forward process. Let $\mathcal{G} = (\mathbf{X}, \mathbf{A})$ be a graph, where $\mathbf{X}$ collects the categorical class of every node and $\mathbf{A}$ the categorical class of every ordered pair of nodes, e.g., encoding the presence or absence of an edge. A discrete diffusion model corrupts $\mathcal{G}$ through a Markov chain of transition matrices $\mathbf{Q}_X^{(1)}, \dots, \mathbf{Q}_X^{(T)}$ for node classes and $\mathbf{Q}_A^{(1)}, \dots, \mathbf{Q}_A^{(T)}$ for edge classes, one pair per timestep, whose entries give the probability of transitioning between classes:

$$\begin{aligned} \mathbf{Q}_X^{(t)}[i,j] &= q\big(\mathbf{x}^{(t)} = \textsc{OneHot}(j) \mid \mathbf{x}^{(t-1)} = \textsc{OneHot}(i)\big), \\ \mathbf{Q}_A^{(t)}[i,j] &= q\big(\mathbf{a}^{(t)} = \textsc{OneHot}(j) \mid \mathbf{a}^{(t-1)} = \textsc{OneHot}(i)\big). \end{aligned} \tag{9}$$

Composing these matrices up to step t, $\overline{\mathbf{Q}}_X^{(t)} = \mathbf{Q}_X^{(1)} \cdots \mathbf{Q}_X^{(t)}$ and $\overline{\mathbf{Q}}_A^{(t)} = \mathbf{Q}_A^{(1)} \cdots \mathbf{Q}_A^{(t)}$, gives the corruption of $\mathcal{G}$ after t steps in a single matrix product for every node and edge:

$$q\big(\mathcal{G}^{(t)} \mid \mathcal{G}\big) = \Big(\mathbf{X}^{(0)}\, \overline{\mathbf{Q}}_X^{(t)},\ \mathbf{A}^{(0)}\, \overline{\mathbf{Q}}_A^{(t)}\Big), \tag{10}$$

so that sampling a noised graph at any timestep does not require simulating the chain step by step. A common choice for $\mathbf{Q}_X^{(t)}, \mathbf{Q}_A^{(t)}$ drives the chain toward a uniform distribution over classes as t grows (Austin et al., 2021; Haefeli et al., 2022); Vignac et al. (2023a) instead choose them so that the chain converges to the empirical marginal distribution of node and edge classes observed in the training graphs, which better preserves rare classes throughout the corruption process, a property particularly relevant for graphs with sparse connectivity or imbalanced node types.

Reverse process. Reversing this corruption requires a denoiser $f_\theta(\cdot)$, trained to recover the clean graph $\mathcal{G}$ from a noised observation $\mathcal{G}^{(t)}$ and the timestep t. $f_\theta(\cdot)$ is typically implemented as a graph transformer (Dwivedi and Bresson, 2020; Vignac et al., 2023a) with L layers, processing node and edge representations jointly so that the predicted class of a node or an edge depends on the rest of the graph. Node and edge entries of $\mathcal{G}^{(t)}$ are first embedded into initial representations $\mathbf{u}_{X,0}[i]$ and $\mathbf{u}_{A,0}[i,j]$. At each layer l, with $0 \le l \le L-1$, node representations attend to one another through self-attention, with attention logits additionally biased by the representation of the connecting edge, so that the presence, absence, or type of an edge affects how strongly two nodes exchange information; edge representations are in turn updated from the attended node pairs. When generation is meant to be steered toward some external condition, an additional global vector $\mathbf{u}_{y,l}$ carries this conditioning signal, and modulates both the updated node and edge representations through a feature-wise linear transformation (Perez et al., 2018):

$$\mathrm{FiLM}\big(\mathbf{u}, \mathbf{u}_{y,l}\big) = \boldsymbol{\gamma}_l\big(\mathbf{u}_{y,l}\big) \odot \mathbf{u} + \boldsymbol{\beta}_l\big(\mathbf{u}_{y,l}\big), \tag{11}$$

where $\mathbf{u}$ denotes either a node representation $\mathbf{u}_{X,l}[i]$ or an edge representation $\mathbf{u}_{A,l}[i,j]$ after attention, $\boldsymbol{\gamma}_l$ and $\boldsymbol{\beta}_l$ are affine maps with parameters learned separately at each layer l, and $\odot$ is an

element-wise product. The global vector for the next layer, $\mathbf{u}_{y,l+1}$, is in turn obtained by pooling the resulting node and edge representations, so that the conditioning signal is progressively enriched, layer after layer, with an overall summary of the graph as it is being denoised.

At the last layer, two linear heads, $\mathbf{W}_X$ for nodes and $\mathbf{W}_A$ for edges, map the resulting representations, $\mathbf{u}_{X,L-1}[i]$ and $\mathbf{u}_{A,L-1}[i,j]$, to distributions over the clean classes:

$$\begin{aligned} p_\theta\big(\mathbf{X}[i] = c \mid \mathcal{G}^{(t)}, t\big) &= \mathrm{softmax}\big(\mathbf{W}_X\, \mathbf{u}_{X,L-1}[i]\big)_c \\ p_\theta\big(\mathbf{A}[i,j] = c \mid \mathcal{G}^{(t)}, t\big) &= \mathrm{softmax}\big(\mathbf{W}_A\, \mathbf{u}_{A,L-1}[i,j]\big)_c. \end{aligned} \tag{12}$$

Rather than predicting the noise applied at step t, as is customary for continuous diffusion models (Ho et al., 2020), $f_\theta(\cdot)$ is thus trained to directly predict, for every node and edge, a distribution over their clean class, an x_0 parameterization that avoids defining a notion of noise for categorical variables (Austin et al., 2021). Writing z generically for a node entry $\mathbf{X}[i]$ or an edge entry $\mathbf{A}[i,j]$, a single reverse step from $z^{(t)}$ to $z^{(t-1)}$ is obtained by combining this prediction with the forward posterior $q(z^{(t-1)} \mid z^{(t)}, z^{(0)})$, known in closed form from the transition matrices above:

$$p_\theta\big(z^{(t-1)} \mid \mathcal{G}^{(t)}\big) = \sum_c p_\theta\big(z^{(0)} = c \mid \mathcal{G}^{(t)}, t\big)\; q\big(z^{(t-1)} \mid \mathcal{G}^{(t)}, z^{(0)} = c\big), \tag{13}$$

and sampling a clean graph from noise amounts to iterating this step, for every node and edge, from $t = T$ down to $t = 1$. A model trained in this way can further be made to represent both this conditional distribution and the corresponding unconditional distribution over graphs, obtained by replacing the conditioning signal with a learned null vector during training with some probability, and combining the two predictions at inference time to control how strongly the generated graph should adhere to the given condition, an approach known as classifier-free guidance (Ho and Salimans, 2022). Graph diffusion models were first developed for generic and molecular graph generation (Niu et al., 2020; Jo et al., 2022; Vignac et al., 2023a), and the discrete construction above has since been adopted across various domains involving small, structured graphs (Sun and Yang, 2023; An et al., 2024; Jiang et al., 2026; Shou et al., 2026); in Section 3, we detail how we specialize it to plan graphs, adapting the node and edge classes, the conditioning signal, and the denoiser architecture to our setting.

C Additional Methodological Details of DiGnosis

Algorithms 1-2 depict the formalism behind the training and inference stages of DiGnosis as described in Section 3 from the main paper: the training procedure of the conditional denoiser (Section 3.2) and the verification and repair pipeline (Sections 3.3 and 3.4), with the two inference passes at t_{diag} and t_{repair}, the node/edge weight α, the repair policy and the chain rewriting.

C.1 Prompt of the LLM edit

The LLM edit of Section 3.4 is obtained with a single call to the backbone, using the fixed template below. The template receives the user query, the plan graph as a numbered list of steps with their tools, the current dependencies, the tool catalog and the diagnosis Δ_q^P, rendered as one line per suspect entry: a node whose confidence p_i falls below η is reported as "`Step` i `uses "`τ`", which looks wrong; most plausible tools here:` τ'_1, τ'_2, τ'_3", where $\tau'_1, \tau'_2, \tau'_3$ are the three most likely tools under $f_\theta(\cdot)$; a node of the out-of-catalog class is reported as "`Step` i `uses "`τ`", which is NOT in the tool catalog`"; an empty slot whose confidence falls below η as "`A step seems to be MISSING (suggested tool:` τ'`)`"; an edge whose confidence falls below η as "`The dependency "`τ_a`" -> "`τ_b`" looks spurious`" if present in the plan graph and "`seems to be missing`" otherwise. When no entry is suspect, the diagnosis reads "`The plan looks globally implausible`". The answer is parsed as JSON; a reply that cannot be parsed leaves the LLM edit out of the candidates.

Algorithm 1 DIGNOSIS training procedure

Require: Queries $\mathcal{Q}$ with their ground truth plan graphs $\mathcal{G}^P$ and, when available, subtask descriptions; extended tool catalog $\mathcal{T}'$; text encoder $\phi(\cdot)$; denoiser $f_\theta(\cdot)$; diffusion steps T; transition matrices $\mathbf{Q}_X^{(t)}, \mathbf{Q}_A^{(t)}$; conditioning dropout p_{drop}; edge weight λ

Plan graph and query representation
Encode each $\mathcal{G}_q^P$ into the node matrix $\mathbf{X}_q^P$ and the edge tensor $\mathbf{A}_q^P$ over $\mathcal{T}'$
Encode each query $q \in \mathcal{Q}$ into $\mathbf{H}_q \leftarrow \phi(q)$, and the description of each subtask, when available, into the embedding of its slot
for each training iteration **do**
    Sample a query q and its ground truth plan graph $\mathcal{G}_q^P = (\mathbf{X}_q^P, \mathbf{A}_q^P)$
    # Forward process
    Sample a timestep $t \sim \text{UNIFORM}(\{1, \dots, T\})$
    Sample the noised plan graph $\mathcal{G}_q^{P(t)} \sim q\big(\mathcal{G}_q^{P(t)} \mid \mathcal{G}_q^P\big)$ ▷ Eq. 1
    With probability p_{drop}, replace $\mathbf{H}_q$ with a learned null vector ▷ conditioning dropout
    # Reverse process
    $p_\theta\big(\mathbf{X}_q^P[i] = c\big), p_\theta\big(\mathbf{A}_q^P[i,j] = c\big) \leftarrow f_\theta\big(\mathcal{G}_q^{P(t)}, t, \mathbf{H}_q\big)$ for every node i and edge (i,j) ▷ Eq. 2, 3
    # Loss computation
    Compute $\mathcal{L}(\theta)$ via categorical cross-entropy against $\mathbf{X}_q^P$ and $\mathbf{A}_q^P$, with the edge term weighted by λ ▷ Eq. 4
    Update θ via backpropagation on $\mathcal{L}(\theta)$
end for
return $f_\theta(\cdot)$

```
You are verifying and fixing a tool-invocation task plan.

# USER REQUEST #
{query}

# CURRENT PLAN (steps with tools) #
1. [{tool_1}] {subtask description 1}
2. [{tool_2}] {subtask description 2}
...

# CURRENT DEPENDENCIES #
{tool_a} -> {tool_b}
...

# AVAILABLE TOOLS (use EXACTLY these names) #
{tool catalog, comma separated}

# VERIFIER DIAGNOSIS (from a learned plan verifier; may be imperfect) #
{diagnosis lines}

# TASK #
Fix ONLY what is actually wrong for this user request; keep every
correct part unchanged. Tools must come from AVAILABLE TOOLS.
Answer with ONLY a JSON object:
{"task_nodes": [{"task": "<tool name>"}, ...],
 "task_links": [{"source": "<tool name>", "target": "<tool name>"}, ...],
 "task_steps": ["<short description of each step,
                 same order as task_nodes>", ...]}
```

Algorithm 2 DIGNOSIS verification and repair procedure

Require: Query q; candidate plan graph $\hat{\mathcal{G}}_q^P = (\hat{\mathbf{X}}_q^P, \hat{\mathbf{A}}_q^P)$ from a task planner, with its subtask descriptions when available; tool catalog $\mathcal{T}$ and its extension $\mathcal{T}'$; text encoder $\phi(\cdot)$; trained denoiser $f_\theta(\cdot)$; timesteps t_{diag} and t_{repair}; node/edge weight α; brokenness threshold σ (ρ-quantile on held out ground truth plan graphs); confidence threshold η; repair policy; LLM

Plan graph and query representation
Encode q into $\mathbf{H}_q \leftarrow \phi(q)$, and the description of each subtask, when available, into the embedding of its slot
Set $\mathcal{G}_q^{P(t)} \leftarrow \hat{\mathcal{G}}_q^P$ by convention, read at $t = t_{\text{diag}}$ for score and diagnosis and at $t = t_{\text{repair}}$ for the repair
Score
$p_i \leftarrow p_\theta\big(\hat{\mathbf{X}}_q^P[i] \mid \hat{\mathcal{G}}_q^P, t_{\text{diag}}, \mathbf{H}_q\big)$, $p_{ij} \leftarrow p_\theta\big(\hat{\mathbf{A}}_q^P[i,j] \mid \hat{\mathcal{G}}_q^P, t_{\text{diag}}, \mathbf{H}_q\big)$ for every node i and edge (i,j) ▷ Eq. 5
$\Sigma_q^P \leftarrow -\Big(\frac{\alpha}{d}\sum_i \log p_i + \frac{1-\alpha}{d(d-1)}\sum_{i\neq j}\log p_{ij}\Big)$ ▷ Eq. 6
Diagnosis
$\hat{c}_i \leftarrow \arg\max_{c\in\mathcal{T}'} p_\theta\big(\mathbf{X}_q^P[i] = c \mid \hat{\mathcal{G}}_q^P, t_{\text{diag}}, \mathbf{H}_q\big)$, $\hat{c}_{ij} \leftarrow \arg\max_{c\in\{0,1\}} p_\theta\big(\mathbf{A}_q^P[i,j] = c \mid \hat{\mathcal{G}}_q^P, t_{\text{diag}}, \mathbf{H}_q\big)$ ▷ Eq. 7
$\Delta_q^P \leftarrow \{(p_i, \hat{c}_i)\}_{i=1}^d \cup \{(p_{ij}, \hat{c}_{ij})\}_{i\neq j}$, flagging as suspect the entries with $p_i < \eta$ or $p_{ij} < \eta$
Repair and selection
if $\Sigma_q^P > \sigma$ **then** ▷ gate: the plan graph is flagged as broken
    $\Omega_q^P \leftarrow$ entrywise $\arg\max$ of $f_\theta\big(\hat{\mathcal{G}}_q^P, t_{\text{repair}}, \mathbf{H}_q\big)$, applied to the whole plan graph or to the suspect entries only, as the repair policy prescribes ▷ Eq. 8
    $\mathcal{K} \leftarrow \{\hat{\mathcal{G}}_q^P, \Omega_q^P, \text{LLM edit}\}$, where the LLM edit corrects $\hat{\mathcal{G}}_q^P$ given q, $\mathcal{T}$ and Δ_q^P ▷ Appendix C.1
    $\hat{\mathcal{G}}_q^P \leftarrow \arg\min_{\mathcal{G}\in\mathcal{K}} \Sigma_q^P(\mathcal{G})$ ▷ self-acceptance: the original plan graph is a candidate
end if
Rewrite $\hat{\mathcal{G}}_q^P$ as the chain implied by its tool order ▷ chain-shaped benchmarks, Section 3.4
return Σ_q^P, Δ_q^P, $\hat{\mathcal{G}}_q^P$

D EXPERIMENTAL SETUP

D.1 DATASETS

We use three benchmarks for task planning, whose statistics are reported in Table 4. For each of them, every method receives the same official test split of 500 queries of the chain-shaped subset; each test query provides the query text, the reference plan graph (used only to compute the metrics) and the DIRECT decomposition into subtasks, reused by the planners that need one (GNN4PLAN and GTOOL). The remaining query-plan graph pairs form the training pool, from which 300 validation queries per benchmark are held out by seed-fixed splits to calibrate the verifiers, together with a further $\sim$10% used for the early stopping of the denoiser; the test identifiers are excluded upstream of every training and calibration stage by an assertion in the split code. Reference plan graphs use at most 8 tools; the test splits are chain-shaped only, whereas the training pools also contain single-tool plan graphs and, on HuggingFace and Multimedia, about 10% of DAG-shaped plan graphs. In every training pool, the node marginal is dominated by the empty class (0.78-0.80 with $d = 12$ slots) and the edge marginal by the absence class (0.988-0.989).

HuggingFace and Multimedia come from TaskBench (Shen et al., 2024), whose tool graphs collect, respectively, 23 HuggingFace models (e.g., image classification, translation, text-to-speech) and 40 multimedia tools (e.g., text summarization, audio effects, video stabilization), and whose queries and reference plan graphs are generated by sampling a subgraph of the tool graph and asking an LLM for a query that requires it.

Table 4: Statistics of the three benchmarks: number of tools; queries used to train the denoiser, for its early stopping, to calibrate the verifiers and to test; tools per reference plan graph; shape of the training plan graphs; marginal probability of the empty node class and of the edge-absence class in the training pools ($d = 12$ slots).

		Queries				Tools per reference plan graph		Shape of the training plan graphs (%)		Class marginals (training)	
Benchmark	Tools	Training	Early stopping	Calibration	Test	mean (training / test)	max	chain	single tool / DAG	∅ (nodes)	absence (edges)
HuggingFace	23	5940	656	300	500	2.50 / 3.81	8	42.8	46.8 / 10.4	0.792	0.989
Multimedia	40	4172	470	300	500	2.65 / 3.92	8	48.3	42.8 / 8.9	0.779	0.988
UltraTool	260	2454	273	300	500	2.44 / 2.38	5	100.0	0.0 / 0.0	0.797	0.989

UltraTool (Huang et al., 2024) covers real-world tool-use scenarios with a larger catalog of 260 tools with terse descriptions; we use the release adopted by Hao et al. (2026). Its reference plan graphs are all chains, of at most 5 tools.

D.2 TASK PLANNERS

We verify the plan graphs of the four planners below, plus a fine-tuned planner used only in RQ3 from Section 5.

DIRECT makes one LLM call with the full catalog, a fixed one-shot example and the query, and parses the JSON plan graph from the reply (replies wrapped in markdown fences are still parsed).

REACT (Yao et al., 2023a) builds the plan graph one tool at a time in a thought-action-observation loop, with a deterministic, catalog-only simulated observation and a cap of 10 turns; its parser tolerates small deviations from the action format.

GNN4PLAN (Wu et al., 2024), in its training-free variant, reuses the DIRECT decomposition, embeds subtasks and tools, propagates tool embeddings over the catalog dependency graph and greedily selects the nearest admissible successor; it makes no LLM call, so its plan graphs change with the backbone only through the decomposition.

GTOOL (Chen et al., 2026) compresses catalog and query into a soft-prompt token for a frozen Vicuna-7B and trains only the graph encoder, on our anti-contamination split; the released code covers the TaskBench benchmarks only, and we extended it to UltraTool. Its plan graphs do not depend on the backbone, and it reuses the subtask decomposition of a reference DIRECT run.

SFT-DIRECT , used only in RQ3 from Section 5, fine-tunes Qwen3-14B with LoRA adapters (Hu et al., 2022) (rank 16, $\alpha = 32$, dropout 0.05, on all attention and MLP projections, ≈64M trainable parameters) on the same correct plan graphs and the same prompt as DIRECT, for 3 epochs with learning rate 2×10^{-4} and greedy decoding; the adapter is trained without the 300 validation queries, which then serve as its own calibration pool.

REACT and GNN4PLAN are our re-implementations, GTOOL is our port of the released code, and DIRECT and SFT-DIRECT are ours.

D.3 PLAN VERIFIERS

We compare DIGNOSIS with the four verifiers below. All LLM-based steps of every verifier use the backbone of the corresponding table, with the same temperature and the same response cache.

SELF-REFINE (Madaan et al., 2023) is a free-form self-critique of the plan graph by the backbone, followed by a critique-guided rewrite.

VERICODER is abstracted from the plan-verification component of Ho et al. (2025), following Hao et al. (2026): a per-subtask LLM consistency check between the chosen tool, the subtask and the catalog description, followed by one fix call.

VERIPLAN is abstracted from Lee et al. (2025): a deterministic, LLM-free check of catalog membership, edge admissibility and acyclicity, which triggers a single corrective LLM call only on violation.

GNNVERIFIER (Hao et al., 2026) trains a GNN to discriminate reference plan graphs from hand-corrupted ones, and uses it to score the plan graph and its components; below a calibrated threshold, an LLM receives the flagged components with a shortlist of alternatives and proposes an edit, which an acceptance rule keeps or discards. We run the released code, with its default hyperparameters and its GNN retrained per benchmark and repetition. On the acceptance rule we follow the deployable reading of the method: the edit is kept when the model's own score improves. The released implementation also exposes a path in which the edit is compared against the reference plan graph of the benchmark before being accepted; since the reference is not available at deployment time, we believe this to be an unintended behaviour of the released code rather than a design choice of the method, and we do not report numbers obtained through it.

D.4 LARGE LANGUAGE MODELS

The two backbones are served locally with Ollama: Qwen3-235B-A22B-Instruct-2507[1] (q4_K_M, 16k context) and Mistral-Small-3.2-24B-Instruct-2506[2] (q8_0, 16k context, vendor system prompt removed). GTOOL runs a frozen Vicuna-7B[3] and SFT-DIRECT a fine-tuned Qwen3-14B[4], independently of the backbone. The text encoder of DIGNOSIS, of GNN4PLAN and of GNNVERIFIER is e5-large (Wang et al., 2022)[5], frozen.

D.5 EVALUATION PIPELINE

For every planner, verifier, benchmark and backbone, the evaluation proceeds through the same steps, so that the task planner (which produces the plan graph) and the verifier (which inspects and possibly repairs it) are never confused:

1. **Planning.** The planner receives the query and the tool catalog and returns a plan graph, i.e., a list of tools with their dependencies, together with a short description of each step. The plan graphs of a planner are generated once per backbone and shared by every verifier, which therefore start from identical inputs.
2. **Verification and repair.** The verifier receives the query, the plan graph with its step descriptions and the catalog, never the reference plan graph, and returns the plan graph, possibly repaired. LLM-based steps use the backbone of the table, with the same decoding parameters for every method and a response cache keyed by query, stage and prompt, so that any configuration is replayed exactly.
3. **Chain rewriting.** For the two graph-based verifiers, DIGNOSIS and GNNVERIFIER, the returned plan graph is rewritten as the chain implied by its tool order (Section 3.4); the LLM-based verifiers return the plan graph as produced.
4. **Scoring.** The returned plan graph is compared with the reference plan graph of the test split with n-F1, l-F1 and Accuracy.
5. **Repetitions.** Every non-deterministic method is run three times: with three seeds of the denoiser for DIGNOSIS, with a retrained GNN and fresh LLM calls for GNNVERIFIER, and with fresh LLM calls for the LLM-based verifiers; the graph-based planners run with fixed seeds.
6. **Calibration.** DIGNOSIS is calibrated per planner and benchmark on the planner's own plan graphs for the 300 validation queries (for REACT and GTOOL, on the validation plan graphs of the DIRECT planner with the same backbone, a transfer discussed in Appendix H); GNNVERIFIER calibrates its thresholds on validation as in its released code.

[1] `Qwen/Qwen3-235B-A22B-Instruct-2507` on the HuggingFace Hub.
[2] `mistralai/Mistral-Small-3.2-24B-Instruct-2506`.
[3] `lmsys/vicuna-7b-v1.5`.
[4] `Qwen/Qwen3-14B`.
[5] `intfloat/e5-large`.

Every design decision and every abandoned attempt of this study is recorded in a decision log released with the code (Appendix G).

D.6 HYPERPARAMETERS AND REPRODUCIBILITY DETAILS

Table 5 lists the hyperparameters of DIGNOSIS and of every planner and verifier, distinguishing, for DIGNOSIS, the values fixed in advance from the grids explored on validation. We detail the former below.

Table 5: Hyperparameters of DIGNOSIS, of the task planners and of the verifiers. For DIGNOSIS, the denoiser hyperparameters are fixed in advance and shared by every benchmark, planner and backbone; the timestep t_{diag} and the weight α of the score are selected per benchmark on validation; the quantile ρ, the timestep t_{repair} and the repair policy are calibrated per planner, benchmark and seed on the planner's own validation plan graphs. The test set is read once with the selected configuration.

Method	Hyperparameters
DIGNOSIS: denoiser (fixed in advance)	$d = 12$ slots; graph transformer with $L = 6$ layers, widths 256 (nodes), 64 (edges) and 128 (global), 8 heads, dropout 0.1 (≈6.4M parameters); $T = 500$ with a cosine schedule and marginal transition matrices; $\lambda = 5$, $p_{\text{drop}} = 0.1$; AdamW (learning rate 2×10^{-4}, AMSGrad), batch 256, up to 400 epochs with early stopping on the validation loss (patience 30); text encoder e5-large, frozen; $\eta = 0.5$.
DIGNOSIS: score (selected per benchmark on the validation AUROC)	$t_{\text{diag}} \in \{0.01, 0.02, 0.05\}\,T$ and $\alpha \in \{0.25, 0.5, 0.75\}$; selected $(0.02\,T, 0.5)$ on HuggingFace, $(0.01\,T, 0.25)$ on Multimedia and $(0.01\,T, 0.75)$ on UltraTool.
DIGNOSIS: pipeline (calibrated per planner, benchmark and seed on the validation n-F1 + l-F1)	$\rho \in \{0.70, 0.85, 0.95\}$; $t_{\text{repair}} \in \{0.1, 0.3, 0.5\}\,T$; repair policy $\in$ {no repair, every suspect entry, low-confidence entries only}.
DIRECT	temperature 0.2, top-p 0.1, at most 2000 generated tokens, one fixed demonstration in the prompt.
REACT	same decoding as DIRECT; at most 10 thought–action–observation turns.
GNN4PLAN	training-free; e5-large embeddings of subtasks and tools; one propagation step over the catalog dependency graph; greedy selection of the nearest admissible successor.
GTOOL	Vicuna-7B-v1.5 frozen; graph encoder trained with the released recipe (learning rate 10^{-5}, weight decay 0.05, effective batch 8, up to 10 epochs with early stopping on validation).
SFT-DIRECT	LoRA on Qwen3-14B: rank 16, $\alpha = 32$, dropout 0.05, on all attention and MLP projections (≈64M trainable parameters); learning rate 2×10^{-4} with cosine decay and 3% warmup, effective batch 16, 3 epochs; greedy decoding.
SELF-REFINE	one critique call and one rewrite call per plan graph; temperature 0.2, top-p 0.1.
VERICODER	one consistency-check call per subtask and one fix call per plan graph; same decoding.
VERIPLAN	deterministic checks of catalog membership, edge admissibility and acyclicity; one corrective call on violation; same decoding.
GNNVERIFIER	released defaults: 3-layer GNN with hidden size 1024, e5-large text encoder, learning rate 2×10^{-5}, 10 epochs with patience 5, ranking margin 0.2, loss weights 1.0 (ranking), 1.0 (graph) and 0.5 (gap), LLM temperature 0.2; GNN retrained per benchmark and repetition; edits accepted on its own score.

Encoding. $d = 12$ slots (no reference plan graph exceeds 8 tools; planner outputs occasionally reach 16, and anything past 12 is truncated and flagged).

Denoiser. Graph transformer with $L = 6$ layers, widths 256 (nodes), 64 (edges) and 128 (global), 8 heads, dropout 0.1, ≈6.4M parameters.

Diffusion. $T = 500$ with a cosine schedule; marginal transition matrices.

Training. Cross-entropy with $\lambda = 5$, $p_{\text{drop}} = 0.1$, AdamW (learning rate 2×10^{-4}, AMSGrad), batch 256, up to 400 epochs with early stopping on the validation loss (patience 30); one model per benchmark, three seeds, one to two minutes per model on one GPU (Table 7).

Inference. $(t_{\text{diag}}, \alpha)$ selected on the validation queries per benchmark, $(0.02\,T, 0.5)$ for HuggingFace, $(0.01\,T, 0.25)$ for Multimedia and $(0.01\,T, 0.75)$ for UltraTool; $\eta = 0.5$; for each planner, benchmark and seed, the threshold quantile $\rho \in \{0.70, 0.85, 0.95\}$, the repair timestep $t_{\text{repair}} \in \{0.1, 0.3, 0.5\}\,T$ and the repair policy (every suspect entry, low-confidence entries only, or no repair) are chosen on the planner's own plan graphs for the 300 validation queries by the sum of n-F1 and l-F1; the test set is read once with the chosen configuration.

E Additional Experimental Results

This section reports the verifiers on top of the fine-tuned planner on the three benchmarks (Appendix E.1), the computational cost of training and inference (Appendix E.2), the sensitivity of DiGnosis to its inference-time and training-time hyperparameters (Appendices E.3 and E.4), and the results with the second, smaller backbone, Mistral-Small-3.2-24B (Appendix E.5).

E.1 Verifiers on top of the fine-tuned planner: all benchmarks

Table 6 complements Table 3 of RQ3 from Section 5 with HuggingFace and UltraTool, and the picture is the same. The chain rewriting of the DiGnosis row, whose calibration selects the no-repair policy on every benchmark and seed, touches the dependencies of 8.0% of the plan graphs on HuggingFace (+0.8 Accuracy and −1.7 l-F1 points; 4.4% on Multimedia) and of none on UltraTool, where the row coincides with the raw plan graph; VeriPlan changes 3.0% and 6.9% of the plan graphs; Self-Refine and VeriCoder change 77% and 79% of the plan graphs on HuggingFace (14% and 15% on UltraTool) and lose up to 33 Accuracy points; GNNVerifier loses 25 points on HuggingFace and is neutral on UltraTool.

Table 6: Verifiers on top of the fine-tuned planner SFT-Direct (LoRA on Qwen3-14B, trained on the same correct plan graphs as DiGnosis, without the validation queries) on the three benchmarks, Qwen3-235B backbone; rows, metrics and marking as in Table 1.

	HuggingFace			Multimedia			UltraTool		
Method	n-F1↑	l-F1↑	Acc↑	n-F1↑	l-F1↑	Acc↑	n-F1↑	l-F1↑	Acc↑
SFT-Direct									
No verifier	88.82	72.20	55.60	**92.96**	80.29	70.00	89.58	74.81	69.20
Self-Refine	76.97±0.04	47.30±0.30	22.87±0.31	89.41±0.15	70.48±0.76	49.33±0.99	87.11±0.16	71.81±0.18	66.00±0.00
VeriCoder	85.52±0.00	67.14±0.00	48.60±0.00	92.39±0.12	79.06±0.09	67.67±0.12	87.73±0.00	72.96±0.00	67.40±0.00
VeriPlan	**88.90±0.02**	**72.35±0.05**	56.20±0.00	92.94±0.00	**80.52±0.00**	70.00±0.00	89.09±0.02	**75.66±0.04**	**70.20±0.00**
GNNVerifier	71.65±4.20	49.45±6.06	31.00±4.58	83.38±5.66	65.58±7.69	49.00±7.94	**89.59±0.04**	74.75±0.05	69.00±0.00
DiGnosis	88.82±0.00	70.55±0.00	**56.40±0.00**	**92.96±0.00**	79.78±0.00	**71.00±0.00**	89.58±0.00	74.81±0.00	69.20±0.00

E.2 Computational time

We compare the training and inference cost of DiGnosis with those of GNNVerifier, the only other verifier that trains a model, of the three LLM-based verifiers, and of the LoRA fine-tuning of SFT-Direct. Training times are measured per epoch and in total on HuggingFace (on the three benchmarks for SFT-Direct); inference times are the mean per query on a fixed subset of 100 test plan graphs per benchmark, with the Direct planner, the smaller Mistral backbone (Appendix E.5), a single request stream and empty caches, on one NVIDIA H200. Table 7 reports the results. Training the denoiser takes 0.80 s per epoch and 84 s in total (6.43M parameters), against 48.6 s per epoch for the GNN of GNNVerifier (36.5M parameters) and 3.4-11.9 h per benchmark for the fine-tuning of the planner. At inference, score and diagnosis cost 19-32 ms per plan graph, and the full pipeline 0.93-1.16 s per query, almost entirely spent in the LLM edit of the flagged plan graphs (0.17-0.35 LLM calls per plan graph): a cost in the range of VeriCoder and VeriPlan, and well below Self-Refine (7.4-12.6 s) and GNNVerifier (1.24-2.27 s).

Table 7: Training and inference cost on one NVIDIA H200. Training: parameters updated, seconds per epoch and total training time (HuggingFace; for SFT-DIRECT the three benchmarks). Inference: mean seconds per query on a fixed subset of 100 test plan graphs (DIRECT planner, Mistral backbone, single request stream, empty caches), HuggingFace / Multimedia / UltraTool; for DIGNOSIS the full pipeline and the score-and-diagnosis part alone.

Method	Trained params	s / epoch	Training (s)	Inference (s / query)
DIGNOSIS (denoiser; full pipeline)	6.43M	0.80	84	1.16 / 0.93 / 1.11
score and diagnosis only (Σ, Δ)				0.032 / 0.019 / 0.020
GNNVERIFIER (GNN; GNN + LLM refine)	36.5M	48.6	583	2.27 / 1.44 / 1.24
SELF-REFINE	–	–	–	8.08 / 7.36 / 12.57
VERICODER	–	–	–	0.90 / 0.65 / 0.68
VERIPLAN	–	–	–	0.86 / 0.41 / 5.24
SFT-Direct (LoRA on Qwen3-14B; a planner)	64.2M	5638 / 4097 / 14318	16915 / 12290 / 42954	–

E.3 SENSITIVITY TO THE INFERENCE-TIME HYPERPARAMETERS

Figure 2 reports the sensitivity of DIGNOSIS to its inference-time hyperparameters, on the test sets, with the Qwen3 backbone: every other hyperparameter is kept at the value selected on validation (Table 1) and one is varied along its grid. Across the grids of t_{repair} and t_{diag}, the 3-benchmark mean Accuracy moves by at most 0.29 points, within the seed-to-seed variation of the calibrated configuration (about 0.2 points). The score depends mostly on ρ and α at the extremes of their grids: $\rho = 0.99$, where almost no plan graph is flagged, costs 2.60 n-F1, 2.90 l-F1 and 3.47 Accuracy points on GNN4PLAN, and $\alpha = 0$, where Σ_q^P reads the edges only, costs 3.30, 4.17 and 3.98 points, which confirms the role of the gate and of the node term; between the extremes, the rows move by at most 0.62 Accuracy points from the calibrated configuration.

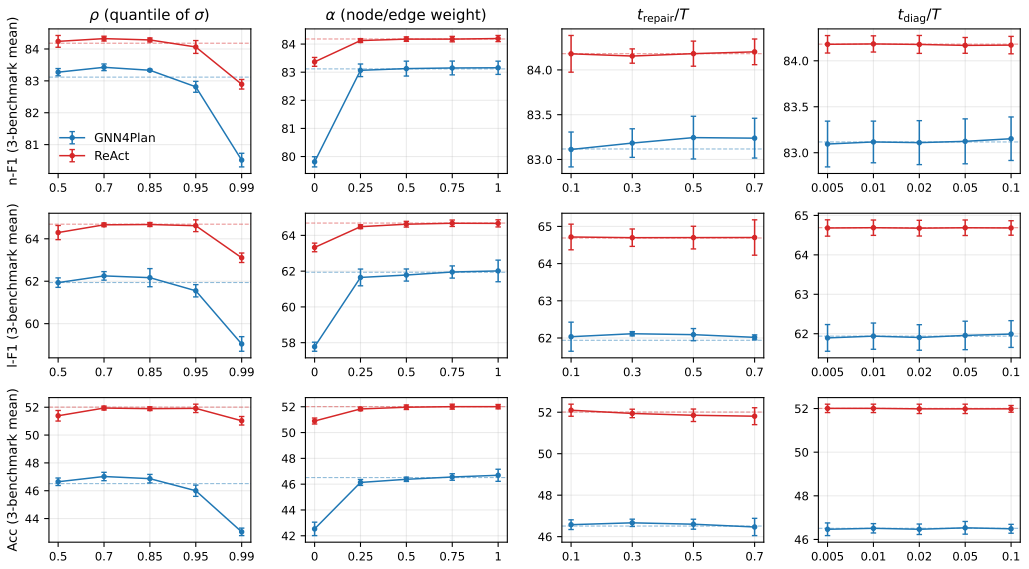


Figure 2: Sensitivity to the inference-time hyperparameters on the test sets (Qwen3 backbone): 3-benchmark mean of each metric (rows) as one parameter (columns) moves along its grid, everything else at the calibrated configuration, whose value is the dashed line; error bars are the standard deviation over three seeds.

E.4 SENSITIVITY TO THE TRAINING-TIME HYPERPARAMETERS

Figure 3 reports the sensitivity to the training-time hyperparameters: the denoiser is retrained with one parameter changed, $\lambda \in \{1, 2, 5, 10, 20\}$, $p_{\text{drop}} \in \{0, 0.1, 0.2, 0.3\}$, $T \in \{100, 250, 500, 1000\}$ or $d \in \{8, 12, 16\}$ slots, with the same three seeds, and evaluated on the final row with the configuration of Table 1. Every variant leaves the final row within 1.09 Accuracy points of 3-benchmark

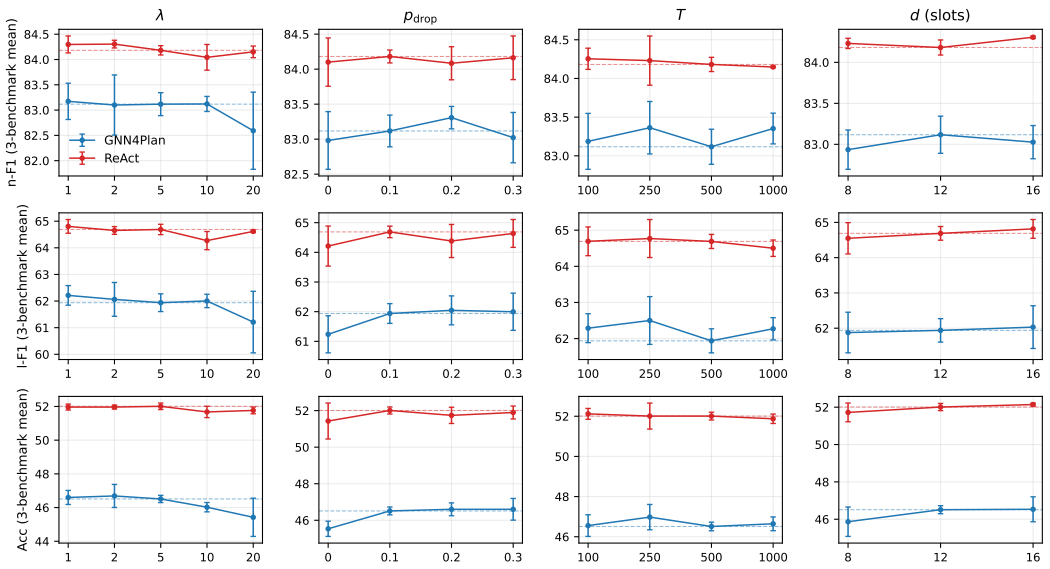


Figure 3: Sensitivity to the training-time hyperparameters: the denoiser is retrained with one parameter changed (three seeds each) and evaluated on the final row with the configuration of Table 1; dashed lines are the paper configuration.

mean of the paper configuration; the largest deviation is $\lambda = 20$ on GNN4PLAN, i.e., the edge weight matters only when it is too high, and the second largest is $p_{\text{drop}} = 0$ (0.98 points on the same planner), which confirms the regularizing role of the conditioning dropout.

E.5 RESULTS WITH THE MISTRAL BACKBONE

This section repeats the experiments of Section 5 on the overall performance and on the fine-tuned planner, and the sensitivity analyses of Appendices E.3 and E.4, with the second, smaller backbone, Mistral-Small-3.2-24B-Instruct (Appendix D.4): the plan graphs of DIRECT, REACT and GNN4PLAN are regenerated with this backbone, every LLM-based step of every verifier runs with it, and the DIGNOSIS and GNNVERIFIER models are the ones trained for Table 1, recalibrated on the validation plan graphs of this backbone (GTOOL and SFT-DIRECT do not depend on the backbone, so their raw plan graphs are unchanged).

Overall performance. Table 8 reports every verifier on top of every planner. The picture of Table 1 is confirmed: DIGNOSIS has the highest 3-benchmark mean of every metric on DIRECT, REACT and GNN4PLAN and the second highest on GTOOL, behind VERICODER, and it improves the Accuracy of the raw plan graph in every configuration but REACT-Multimedia and GTOOL-UltraTool, by +5 to +11 points on DIRECT and GNN4PLAN and +19 on GTOOL-Multimedia. The configurations where it is not the best on Accuracy are the same four as with Qwen3, plus REACT-HuggingFace, where VERIPLAN edges it by 0.1 points, within the seed-to-seed variation. The LLM-based verifiers are more harmful with the weaker backbone: SELF-REFINE degrades the Accuracy of the raw plan graph in 11 configurations out of 12, by up to 24 points, and GNNVERIFIER, which delegates its correction to the backbone and accepts it on its own score, falls 18-41 Accuracy points below the raw plan graph on HuggingFace and Multimedia.

Verification on top of the fine-tuned planner. Table 9 applies every verifier to the plan graphs of SFT-DIRECT with the Mistral backbone. Since SFT-DIRECT does not depend on the backbone, the raw plan graphs and the DIGNOSIS row, whose calibration again selects the no-repair policy on every benchmark, are identical to Table 6; the LLM-based verifiers, whose calls now run with Mistral, degrade the fine-tuned planner more than with Qwen3: SELF-REFINE loses 31-36 Accuracy

Table 8: Verification and repair of the plan graphs of four planners with the Mistral-Small-3.2-24B-Instruct backbone; rows, metrics and marking as in Table 1. The plan graphs of DIRECT, REACT and GNN4PLAN are regenerated with this backbone, every LLM-based step of every verifier runs with it, and the DIGNOSIS and GNNVERIFIER models are the ones of Table 1, recalibrated on the validation plan graphs of this backbone (mean and standard deviation over three repetitions, as in Table 1).

	HuggingFace			Multimedia			UltraTool		
Method	n-F1↑	l-F1↑	Acc↑	n-F1↑	l-F1↑	Acc↑	n-F1↑	l-F1↑	Acc↑
Direct									
No verifier	81.45	57.67	37.20	**88.55**	72.08	53.80	70.35	37.77	25.00
SELF-REFINE	73.20±0.04	41.68±0.16	18.20±0.20	83.80±0.24	61.18±0.70	40.67±0.61	66.42±0.20	30.40±0.94	20.93±0.58
VERICODER	**81.60±0.00**	57.49±0.00	37.00±0.00	88.25±0.07	71.85±0.10	53.87±0.12	70.56±0.06	37.85±0.00	25.00±0.00
VERIPLAN	80.85±0.03	57.77±0.03	37.80±0.00	88.32±0.00	72.18±0.00	54.00±0.00	72.68±0.12	46.05±0.05	30.40±0.00
GNNVERIFIER	60.71±0.34	30.90±0.26	12.67±0.58	65.68±1.55	41.00±2.01	20.67±1.53	70.23±0.11	39.22±0.02	25.00±0.00
DIGNOSIS	81.37±0.38	**60.56±0.10**	**43.47±0.70**	88.28±0.05	**73.67±0.22**	**59.53±0.12**	**77.28±0.57**	**50.06±0.69**	**34.87±0.64**
ReAct									
No verifier	81.09	58.92	41.80	87.93	72.01	60.00	75.84	49.28	41.28
SELF-REFINE	73.10±0.24	42.68±0.28	21.87±0.83	81.44±0.36	58.49±0.36	36.53±0.81	68.85±0.39	35.40±0.56	26.79±1.03
VERICODER	81.14±0.02	59.03±0.02	41.87±0.12	88.05±0.11	72.14±0.14	60.00±0.00	75.96±0.00	49.53±0.00	41.48±0.00
VERIPLAN	80.78±0.06	59.24±0.06	**42.40±0.00**	**88.12±0.05**	**72.70±0.02**	**60.60±0.00**	76.09±0.08	50.54±0.19	42.95±0.12
GNNVERIFIER	61.35±0.59	32.31±0.25	14.00±1.00	65.29±2.14	40.10±3.23	19.33±3.21	73.31±0.38	49.13±0.20	40.67±0.58
DIGNOSIS	**81.43±0.27**	**59.68±0.33**	42.27±0.31	87.81±0.27	71.43±0.76	58.87±0.92	**79.56±0.37**	**55.04±0.75**	**45.96±0.46**
GNN4Plan									
No verifier	71.62	47.66	30.60	81.36	64.43	48.60	67.56	43.03	29.00
SELF-REFINE	74.18±0.34	45.61±0.69	22.00±0.69	83.32±0.30	59.67±0.60	37.67±1.03	68.42±0.36	34.59±0.54	22.40±0.40
VERICODER	79.51±0.06	57.88±0.02	39.60±0.00	**85.66±0.03**	**70.03±0.08**	**55.27±0.12**	75.26±0.15	47.12±0.13	31.40±0.00
VERIPLAN	71.62±0.00	47.66±0.00	30.60±0.00	81.36±0.00	64.43±0.00	48.60±0.00	67.56±0.00	43.03±0.00	29.00±0.00
GNNVERIFIER	55.85±0.20	25.66±0.46	9.00±0.00	61.17±1.59	36.43±2.14	16.33±2.52	67.66±0.25	43.19±0.07	29.00±0.00
DIGNOSIS	**80.51±0.23**	**59.22±0.62**	**41.60±0.40**	85.51±0.16	69.40±0.33	54.00±0.92	**77.48±0.46**	**49.43±0.59**	**33.93±0.64**
GTool									
No verifier	75.77	50.69	29.80	76.62	53.63	33.80	81.29	61.14	53.31
SELF-REFINE	76.86±0.19	48.33±0.13	25.67±0.42	85.73±0.21	64.21±0.53	44.33±0.23	74.14±0.36	41.46±0.67	30.86±0.40
VERICODER	80.98±0.08	58.74±0.06	39.87±0.12	**88.08±0.08**	**72.99±0.16**	**57.60±0.20**	83.86±0.21	63.21±0.46	54.98±0.31
VERIPLAN	79.72±0.10	57.42±0.14	35.87±0.12	81.47±0.14	62.05±0.18	44.33±0.31	**85.77±0.07**	**65.28±0.06**	**56.45±0.12**
GNNVERIFIER	59.37±0.34	26.96±1.23	8.33±0.58	61.93±1.79	36.51±3.16	16.33±3.21	81.27±0.03	61.12±0.04	53.00±0.00
DIGNOSIS	**81.34±0.58**	**59.43±0.86**	**40.20±1.31**	85.94±0.52	68.66±1.31	52.47±1.60	80.52±0.73	57.94±0.62	49.90±0.72

Table 9: Verifiers on top of the fine-tuned planner SFT-DIRECT, Mistral-Small-3.2-24B backbone (mean and standard deviation over three repetitions); rows, metrics and marking as in Table 1.

	HuggingFace			Multimedia			UltraTool		
Method	n-F1↑	l-F1↑	Acc↑	n-F1↑	l-F1↑	Acc↑	n-F1↑	l-F1↑	Acc↑
SFT-Direct									
No verifier	**88.82**	**72.20**	55.60	**92.96**	80.29	70.00	**89.58**	74.81	69.20
SELF-REFINE	74.47±0.13	43.87±0.39	19.47±0.23	83.11±0.29	59.84±0.19	36.00±1.11	76.40±0.38	47.49±1.62	37.87±1.10
VERICODER	86.76±0.12	68.72±0.19	52.07±0.31	91.90±0.05	77.77±0.18	65.80±0.40	**89.58±0.00**	74.42±0.04	68.27±0.12
VERIPLAN	88.73±0.00	72.04±0.00	55.60±0.00	92.93±0.02	**80.49±0.00**	70.40±0.00	89.19±0.03	**75.88±0.12**	**70.27±0.12**
GNNVERIFIER	69.35±4.02	41.61±9.01	23.00±7.94	78.15±4.33	55.16±6.41	35.67±5.51	89.55±0.02	74.76±0.04	69.00±0.00
DIGNOSIS	**88.82±0.00**	70.55±0.00	**56.40±0.00**	**92.96±0.00**	79.78±0.00	**71.00±0.00**	**89.58±0.00**	74.81±0.00	69.20±0.00

points on the three benchmarks, GNNVERIFIER 33 and 34 points on HuggingFace and Multimedia, and VERICODER up to 4 points, while VERIPLAN moves the raw plan graph by at most 1.1 points.

Sensitivity. Figures 4–5 repeat the sensitivity analysis of Appendices E.3–E.4 with the Mistral backbone (same denoisers and checkpoints, plan graphs and LLM edits produced by that backbone, calibrated configuration equal to the DIGNOSIS rows of Table 8). The figures show the same flat profiles as with Qwen3: at inference, only $\rho \geq 0.95$ and $\alpha = 0$ cost more than one Accuracy point of 3-benchmark mean, on GNN4PLAN (1.1 and 3.9 points for $\rho = 0.95$ and 0.99, 4.9 points for $\alpha = 0$); at training, only $\lambda = 20$ does (1.1 points on GNN4PLAN), while $\lambda = 1$ gains 1.1 points on the same planner.

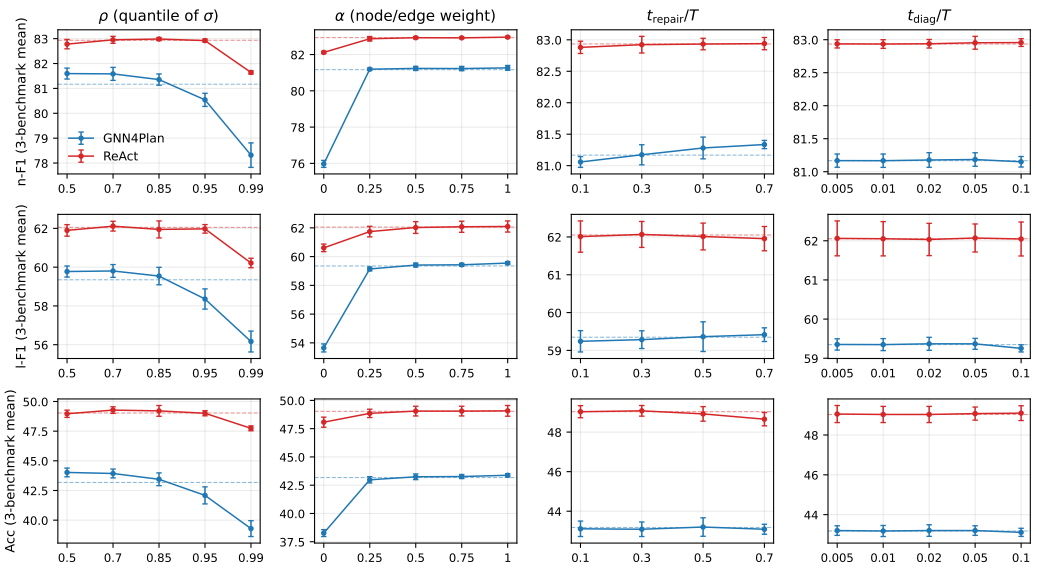


Figure 4: Mistral backbone: sensitivity to the inference-time hyperparameters on the test sets, 3-benchmark mean of each metric (rows) as one parameter (columns) moves along its grid, everything else at the configuration calibrated for this backbone, whose value is the dashed line; error bars are the standard deviation over three seeds.

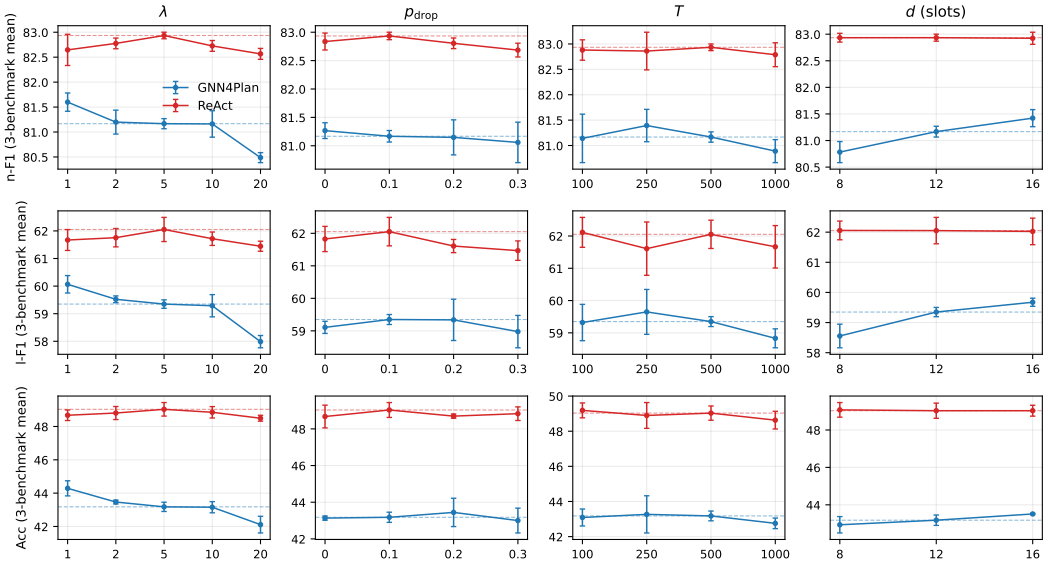


Figure 5: Mistral backbone: sensitivity to the training-time hyperparameters, the denoiser retrained with one parameter changed (three seeds each) and evaluated on the final row with the configuration of Table 8; dashed lines are the paper configuration.

F Case Studies

This section examines two representative test queries of the DiGnosis rows of Table 1 (Qwen3 backbone, Multimedia, seed 0), one on a GNN4Plan plan graph and one on a ReAct plan graph, and traces what the verifier did on each: the brokenness score against the calibrated threshold, the

diagnosis Δ_q^P that is injected into the LLM edit prompt, the three candidates (original plan graph, one-shot repair Ω_q^P, LLM edit) with their scores, and the plan graph the selection step kept. The first example (Table 10) is a wrong plan graph that DIGNOSIS detects and repairs; the second example (Table 11) is a correct plan graph that DIGNOSIS damages. Tools that differ from the ground truth are set in bold.

F.1 SUCCESSFUL DETECTION AND REPAIR

Table 10: A wrong GNN4PLAN plan graph detected and repaired by DIGNOSIS (Multimedia, id 29300562, seed 0). The planner's own subtask description contradicts the tool it chose; the diagnosis localizes the error to that node and the one-shot repair replaces the tool. Tools differing from the ground truth are in bold.

Query	I have a short text which is 'Me encanta jugar al fútbol en mi tiempo libre'. I need it to be paraphrased and then translated to English. Also, I have an audio recording 'example.wav' with my voice. I would like to change the pitch of the audio to be higher.	
Ground truth	Text Paraphraser → Text Translator → Voice Changer	
GNN4PLAN plan graph	Text Paraphraser → Text Translator → **Text-to-Audio** Step 1: Paraphrase the given Spanish text … Step 2: Translate the paraphrased Spanish text into English. Step 3: Modify the pitch of the audio file 'example.wav' to make it higher …	
Verifier	Brokenness $\Sigma_q^P = 0.034$, threshold $\sigma = 0.012$: flagged. Per-node confidence $p_i = (1.00,\ 0.99,\ 0.19)$. Diagnosis Δ_q^P injected into the LLM edit prompt: *- Node 3 uses "Text-to-Audio", which looks wrong; most plausible tools here: Voice Changer, Audio Effects.*	
Candidates	original: … → Text-to-Audio	$\Sigma_q^P = 0.034$
	one-shot repair Ω_q^P: … → Voice Changer	$\Sigma_q^P = 0.0001$
	LLM edit: … → **Audio Effects**	$\Sigma_q^P = 0.0002$
Selected	one-shot repair Ω_q^P: Text Paraphraser → Text Translator → Voice Changer (n-F1/l-F1/Acc: $66.7/50.0/0 \rightarrow 100/100/1$)	

A tool that is inconsistent with its own subtask (Table 10). The query asks to raise the pitch of an existing recording, and the planner's subtask description says exactly that, but the tool attached to it is Text-to-Audio, which synthesizes audio from text. The verifier assigns the plan graph a brokenness score three times the threshold, and the per-node confidence isolates the fault: nodes 1-2 keep confidence above 0.99 while node 3 drops to 0.19, so the diagnosis names one node only and ranks Voice Changer first among the alternatives. The one-shot repair takes that alternative and reaches a score two orders of magnitude lower than the original. The LLM editor, given the same diagnosis, picked the second alternative, Audio Effects, which is equally type-consistent (audio and text in, audio out) but does not match the query; the selection step kept the one-shot candidate because it scored lower. The outcome is the same in all three seeds, and in seed 1 the LLM also proposed Voice Changer. The case illustrates the intended division of labour: the verifier localizes, both repair candidates propose, and the score arbitrates, without granting the LLM candidate any privilege (Table 10).

F.2 BAD CASE ANALYSIS

A plausible neighbour that the score cannot tell apart (Table 11). The query asks to find the relevant topics of an article, condense its main points and search for more information about them, and the REACT plan graph is identical to the ground truth. Topic Generator and Keyword Extractor, however, have overlapping descriptions ("generates a list of relevant topics" versus "extracts the most important keywords"), identical input and output types, and appear in the same positions of the same chains, so the learned score barely separates them: the flag itself is marginal ($\Sigma_q^P = 0.016$ against $\sigma = 0.012$) and the confidence on node 1 is 0.47, just under the threshold $\eta = 0.5$ below which an entry is reported as suspect. The diagnosis, however, is phrased categorically, and both repair candidates comply with it: the one-shot repair by construction, the LLM editor despite the prompt's warning that the diagnosis may be imperfect. Because the same score that produced the

Table 11: A correct REACT plan graph damaged by DIGNOSIS (Multimedia, id 24934626, seed 0). The score is barely above σ and the confidence on node 1 barely below η, yet the diagnosis is stated categorically; both repair candidates follow it and the score confirms the substitution.

Query	I have an article about the benefits of artificial intelligence on different industries, please help me find relevant topics, condense the main points and search for more information about those points. Here is the article text: 'Artificial Intelligence has revolutionized many industries, including healthcare, finance, and agriculture. It has led to advancements in areas such as diagnostics, investment strategies, and crop yield optimization.'	
Ground truth	Topic Generator → Text Summarizer → Text Search	
REACT plan graph	Topic Generator → Text Summarizer → Text Search (identical to the ground truth) Step 1: First, I need to extract relevant topics from the given article text … Step 2: Next, I should summarize the article to condense its main points … Step 3: …search the internet for more information on those topics using a text search.	
Verifier	Brokenness $\Sigma_q^P = 0.016$, threshold $\sigma = 0.012$: flagged. Per-node confidence $p_i = (0.47,\ 1.00,\ 1.00)$. Diagnosis Δ_q^P injected into the LLM edit prompt: *- Node 1 uses "Topic Generator", which looks wrong; most plausible tools here: Keyword Extractor, Article Spinner.*	
Candidates	original: Topic Generator → …	$\Sigma_q^P = 0.016$
	one-shot repair Ω_q^P: **Keyword Extractor** → …	$\Sigma_q^P = 0.0002$
	LLM edit: **Keyword Extractor** → …	$\Sigma_q^P = 0.0001$
Selected	LLM edit: **Keyword Extractor** → Text Summarizer → Text Search	(n-F1/l-F1/Acc: $100/100/1 \to 66.7/50.0/0$)

diagnosis also judges the candidates, the substituted plan scores two orders of magnitude lower and the selection has no independent evidence to keep the original. A likely cue for the confusion is the planner's own wording, "extract relevant topics", which reads like a Keyword Extractor step. The confusion runs in both directions: on a GNN4PLAN plan graph for a query that asks for "the main topics discussed" (id 17489153), the planner used Keyword Extractor where the ground truth has Topic Generator, and the verifier gave that step confidence 0.998 and did not flag the plan in any seed. Near-synonymous tools are the dominant residual error of the verifier, in both false alarms and misses. The case is also seed-sensitive: in seed 1 the same plan graph is flagged but node 1 stays above η, no diagnosis line is emitted, the LLM returns the original tools, and the final plan is correct. Such damaged plan graphs are rare on this benchmark (1-11 of 500 test plans per seed for REACT, 0-1 for GNN4PLAN, against 15-20 wrong GNN4PLAN plan graphs repaired per seed), but they are the price of a verifier whose threshold and diagnosis are calibrated on correct plan graphs only (Table 11).

G COMPUTATIONAL RESOURCES AND CODEBASE

All experiments ran on one server with two NVIDIA H200 GPUs (141 GB each), with the LLM backbones served locally by Ollama. Training one DIGNOSIS denoiser takes one to two minutes on one GPU (Table 7), and the whole DIGNOSIS study of this paper required about 160 denoiser trainings, i.e., a few GPU-hours; the cost of the campaign is dominated by the LLM calls of the planners and of the LLM-based verifiers, roughly one GPU-day per repetition of the full grid with either backbone, and by the fine-tuning of SFT-DIRECT (3.4-12 hours per benchmark). The code, the released decision log and the scripts that generate every table of this paper are available at the following anonymous repository: `https://anonymous.4open.science/r/Graph-Validation-for-Agentic-AI-EFA6`.

H LIMITATIONS

Limitations of the model. DIGNOSIS inherits the shape of its benchmarks: every reference test plan graph is a chain of at most 8 tools, the encoding uses a fixed capacity of $d = 12$ slots, and the chain rewriting step exploits that shape, so that general DAG-shaped plans and long-horizon

workflows are out of the scope of this evaluation. The encoding itself already represents DAG-shaped plan graphs, since the edge tensor is a full directed $d \times d$ tensor; extending DIGNOSIS to them requires training on DAG-shaped reference plan graphs, which are a minority of the training pools (Table 4), and dropping the chain rewriting, which we leave to future work. The verifier is trained per benchmark on its correct plan graphs: a new tool catalog requires collecting correct plan graphs and retraining, which is inexpensive (about one minute) but not free, whereas prompting-based verifiers apply out of the box. Finally, near-synonymous tools are the dominant residual error of the verifier: on Multimedia, the score separates correct and broken plan graphs less well than elsewhere (test AUROC 0.58-0.65 on GNN4PLAN and REACT, against 0.67-0.86 on the other benchmarks), and a verifier that rewrites every plan graph can recover errors ours never flags.

Limitations of the experimental protocol. Calibration relies on 300 validation queries and on the planner's own plan graphs; for the planners that had no validation plan graphs of their own (REACT and GTOOL) it is transferred from the DIRECT planner with the same backbone, and on GTOOL-UltraTool this transfer flags too many correct plan graphs and leaves the row below the raw plan graph: a calibration on GTOOL's own validation plan graphs abstains there and returns the raw plan graph, without recovering the gap to VERIPLAN, whose catalog check repairs the out-of-catalog tools this planner invents. The three benchmarks measure correctness against a single reference plan graph per query, so that plan graphs that solve the query differently from the reference count as wrong for every method. Finally, the strongest planner we could build in domain, a LoRA-fine-tuned DIRECT, leaves little for any verifier to add: the value of verification concentrates on prompted planners and closed backbones, where retraining the planner is not an option.