MOSAIC: CONFIDENCE-AWARE CONTRASTIVE LEARNING OVER MULTIPLEX TOOL GRAPHS FOR ROBUST TASK PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Task planning requires LLM-based agents to translate a complex user request into an ordered sequence of tool invocations. Graph-based task planners address this problem by learning over a tool graph that encodes which tools can be invoked in sequence, and consistently outperform planners that rely on LLMs alone. However, the tool graph is rarely available and is built heuristically, either from historical tool invocations or from input-output type compatibility; therefore, its edges vary widely in reliability, yet existing planners treat them as equally trustworthy binary connections and typically rely on a single construction. We propose MOSAIC (**M**ultiplex c**O**nfidence-aware **S**tructure-**A**ugmented **I**dentity-aware **C**ontrastive learning), a model-agnostic plug-in method that makes the tool representations of graph-based planners robust to unreliable tool graphs. MOSAIC (i) combines the historical and dependency graphs into a confidence-weighted multiplex tool graph; (ii) trains relation-specific graph encoders with a contrastive objective whose augmentations drop low-confidence edges more aggressively and which aligns each tool across the two relations; and (iii) fuses the resulting embeddings with learnable, graph-agnostic tool identity embeddings. Across three TaskBench domains, three LLMs, and two state-of-the-art planners, MOSAIC consistently improves planning performance (with relative accuracy gains of up to 22.25%), remains more robust under structural noise injected into the tool graphs, and adds only marginal computational overhead.

1 INTRODUCTION

Large language model (LLM) agents increasingly solve complex user requests by orchestrating external tools, such as APIs or foundation models. At the core of this capability lies *task planning*: turning a natural language request into an ordered sequence of tool invocations. Early solutions prompt the LLM to decompose the request into subtasks and assign a tool to each of them (Shen et al., 2023; 2024), possibly through tree- or graph-structured reasoning (Yao et al., 2023a; Besta et al., 2024); later ones fine-tune LLMs on large tool corpora (Qin et al., 2024; Yang et al., 2023), map each tool to a dedicated token (Wang et al., 2025), or refine plans through execution feedback at the price of repeated LLM calls (Yao et al., 2023b). A recurring observation is that tools are not independent, since the output of one often feeds another, and that neglecting these dependencies yields invalid plans (Chen et al., 2026; Gao et al., 2026; Kang & Li, 2026).

Tool graphs make such dependencies explicit, connecting tools that can be invoked in sequence, so that a plan corresponds to a path in the graph. Yet LLMs reason poorly over graph-structured inputs: when graphs are serialized into text, they are sensitive to the input format and to spurious correlations (Guo et al., 2023; Wang et al., 2023), a gap that instruction tuning only partially closes (Luo et al., 2024; Ye et al., 2023; Jin et al., 2024). This has motivated *graph-based task planners*, which use the tool graph to search the tool space (Liu et al., 2024a;b), reason over graph-shaped plans (Lin et al., 2024; Zhang et al., 2025), retrieve dependent tools (Gao et al., 2025; Patel et al., 2026), or plan over function-level workflows (You et al., 2026). Among them, GNN4Plan (Wu et al., 2024) delegates tool selection to a graph neural network (GNN), while GTool (Chen et al., 2026) encodes a request-specific tool graph with a GNN and injects the resulting graph token into the LLM.

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.

However, all these planners take the tool graph as given, although it is rarely available and must be built heuristically (Kang & Li, 2026). Two constructions prevail. *Historical* (trajectory-derived) graphs connect tools invoked consecutively in past executions (Liu et al., 2024a); they flatten all transitions to unit weight, so that a pair observed once counts as much as one observed hundreds of times, and they unavoidably contain erroneous and missing edges (Chen et al., 2026). *Dependency* (schema-derived) graphs connect tools whose output and input types match (Liu et al., 2024b); since compatibility is checked at a coarse type level, a single common type (e.g., plain text) links many functionally unrelated tools. The literature widely acknowledges this unreliability: GNN4Plan lists the manual construction of the graph among its limitations (Wu et al., 2024), static dependency graphs bias agents toward repeated calls (Patel et al., 2026), planning quality depends heavily on graph quality (Gao et al., 2026), and rarely used tools remain sparsely connected (You et al., 2026). Still, existing planners treat edges of very different reliability as equally trustworthy binary connections, which GNNs then propagate into the tool representations, and typically commit to a single construction, discarding the complementary evidence of the other. The only component acting on the edges, the missing-dependency prediction of GTool (Chen et al., 2026), rooted in link prediction (Kipf & Welling, 2016; Zhang & Chen, 2018), adds edges but does not quantify how much each can be trusted.

To address these limitations, we propose MOSAIC (**M**ultiplex c**O**nfidence-aware **S**tructure-**A**ugmented **I**dentity-aware **C**ontrastive learning), a model-agnostic plug-in method that strengthens the tool representation learning of existing graph-based planners without modifying their planning architecture. Our contributions may be summarized as follows:

- We identify two limitations of graph-based task planners: their reliance on heuristically constructed tool graphs with unreliable edges, and their typical use of a single graph construction paradigm.
- We propose MOSAIC, which (i) combines the historical and dependency graphs into a sparsified *multiplex* tool graph whose edges are weighted by confidence scores; (ii) trains relation-specific GNN encoders with a *confidence-aware* graph contrastive objective (Zhu et al., 2020; You et al., 2020), whose augmented views drop low-confidence edges more aggressively (Zhu et al., 2021) and whose cross-relation term aligns each tool across the two relations (Hassani & Khasahmadi, 2020; Park et al., 2020); and (iii) fuses the graph-aware embeddings with learnable *identity* embeddings, agnostic to the tool graph, to absorb its residual noise.
- We mount MOSAIC on GNN4Plan and GTool and evaluate it on three TaskBench domains (Shen et al., 2024) with three LLMs. MOSAIC improves every metric in all main configurations, with relative accuracy gains of up to 22.25%, while making both planners more robust to noise injected into the tool graphs at only a marginal computational cost.

2 PRELIMINARIES

DEFINITION 2.1 (Tool graph) *Let $\mathcal{T} = \{t_1, t_2, \ldots, t_{|\mathcal{T}|}\}$ be the set of available tools (e.g., APIs or foundation models) in a tool repository. Each tool t_i is described in natural language, and its description is encoded by a pretrained language model into an embedding $\mathbf{x}_i^T \in \mathbb{R}^d$; let $\mathbf{X}^T \in \mathbb{R}^{|\mathcal{T}|\times d}$ collect all such embeddings. We define a set of directed edges $\mathcal{E}^T \subseteq \mathcal{T} \times \mathcal{T}$, where an edge $(t_i, t_j) \in \mathcal{E}^T$ indicates that tool t_j can be invoked after tool t_i according to a given construction criterion (e.g., input-output compatibility or historical tool executions). The tool graph is then defined as $\mathcal{G}^T = (\mathbf{X}^T, \mathcal{E}^T)$, with adjacency matrix $\mathbf{A}^T \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{T}|}$.*

DEFINITION 2.2 (Graph-based task planning) *Given a user query q expressed in natural language and a tool graph $\mathcal{G}^T$, task planning consists in predicting a sequence of tools $\tilde{\pi}_q = \langle \tilde{t}_1, \tilde{t}_2, \ldots, \tilde{t}_m \rangle$ that solves q, where consecutive tools correspond to edges of $\mathcal{G}^T$, so that the plan forms a valid path (or, more generally, a subgraph) in the tool graph. A graph-based task planner first learns tool representations from the tool graph, $\mathbf{T} = f_\theta(\mathbf{X}^T, \mathbf{A}^T) \in \mathbb{R}^{|\mathcal{T}|\times d}$, typically through a graph neural network (GNN), and then predicts the plan as $\tilde{\pi}_q = \mathit{plan}(\mathbf{T}, q)$. The planner is trained by comparing $\tilde{\pi}_q$ against the ground-truth tool sequence $\pi_q = \langle t_1, t_2, \ldots, t_m \rangle$ through a task planning loss $\mathcal{L}_{\mathit{task}} = \ell(\mathit{plan}(\mathbf{T}, q), \pi_q)$.*

Note that plan$(\cdot)$ abstracts the planner-specific way in which the query is processed. For instance, GNN4Plan (Wu et al., 2024) first decomposes q into subtasks through an LLM and retrieves one tool per subtask, whereas GTool (Chen et al., 2026) injects a request-specific graph token into the LLM, which then generates the tool sequence. In both cases, the quality of $\mathbf{T}$ directly depends on the reliability of $\mathcal{G}^T$, which is the component MOSAIC acts upon.

3 MOSAIC: ROBUST GRAPH-BASED TASK PLANNING

In this section, we present MOSAIC, a model-agnostic approach that can be mounted on top of existing graph-based task planners, such as GNN4Plan (Wu et al., 2024) and GTool (Chen et al., 2026), to make their tool representations robust to unreliable tool graphs. As depicted in Figure 1, MOSAIC (i) combines confidence-weighted and sparsified versions of the historical and dependency tool graphs into a multiplex tool graph; (ii) trains relation-specific encoders through a confidence-aware graph contrastive learning module, with intra- and cross-relation objectives; and (iii) fuses the resulting embeddings with learnable tool identity embeddings. Finally, we describe how MOSAIC is integrated into the backbone planner.

3.1 MULTIPLEX TOOL GRAPH REPRESENTATION LEARNING

The edge set $\mathcal{E}^T$ of the tool graph (Definition 2.1) is rarely available in practice, and graph-based planners derive it heuristically according to one of two criteria: (i) input-output *dependency*, adding an edge (t_i, t_j) whenever the output type of t_i matches the input type of t_j; (ii) *historical* executions, adding an edge (t_i, t_j) whenever t_j has been invoked right after t_i in a past tool-invocation trajectory. We refer to the resulting graphs as the *dependency* and *historical* tool graphs. As discussed in Section 1, existing planners typically rely on only one of them, and neither construction, taken in isolation, yields reliable tool-tool connections. MOSAIC instead exploits both graphs, and improves the construction of each by weighting its edges according to their confidence, as described next.

3.1.1 DEPENDENCY TOOL GRAPH

Let c_i^{out} and c_j^{in} denote the output type of tool t_i and the input type of tool t_j. The dependency tool graph contains a directed edge $(t_i, t_j) \in \mathcal{E}^T_{\text{dep}}$ whenever $c_i^{\text{out}} = c_j^{\text{in}}$, and existing approaches assign the same unit weight $w_{ij}^{\text{dep}} = 1$ to every such edge (Wu et al., 2024).

Type compatibility, however, is a coarse criterion: it connects every pair of tools sharing a type, regardless of their functional relatedness, so that a single popular type (e.g., plain text) can generate a large clique of edges between otherwise unrelated tools. Unit weights make these edges indistinguishable from meaningful ones. Therefore, we weight each edge by combining two signals. The first is the semantic similarity between the tool descriptions, $\sigma_{ij}^{\text{dep}} = \cos(\mathbf{x}_i^T, \mathbf{x}_j^T)$ rescaled to be within the range [0, 1]. The second is the specificity of the type inducing the edge: letting $f_c = |\{(t_k, t_l) \in \mathcal{E}^T_{\text{dep}} \mid c_k^{\text{out}} = c_l^{\text{in}} = c\}|/|\mathcal{E}^T_{\text{dep}}|$ be the fraction of dependency edges induced by type c, we follow an inverse-document-frequency formulation and define $\varphi_{ij}^{\text{dep}} = \log(1/f_c)$, with $c = c_i^{\text{out}} = c_j^{\text{in}}$, so that edges induced by rarer, more discriminative types receive higher weight. The final edge weight is:

$$\hat{w}_{ij}^{\text{dep}} = \sigma_{ij}^{\text{dep}} \cdot \varphi_{ij}^{\text{dep}}. \tag{1}$$

3.1.2 HISTORICAL TOOL GRAPH

Let $\mathcal{P}$ be a collection of historical tool-invocation trajectories (in our experiments, the ground-truth plans of the training queries), each listing tools in their execution order. The historical tool graph contains a directed edge $(t_i, t_j) \in \mathcal{E}^T_{\text{hist}}$ whenever t_j is invoked right after t_i in at least one trajectory of $\mathcal{P}$. Existing approaches assign the same unit weight $w_{ij}^{\text{hist}} = 1$ to every such edge (Chen et al., 2026), so that a transition observed once is treated as equally reliable as one observed in many trajectories.

Conversely, we weight each edge by the evidence supporting it. Let ν_{ij} be the number of times the transition (t_i, t_j) occurs across all trajectories in $\mathcal{P}$, and let $\bar{\nu} = \frac{1}{|\mathcal{E}^T_{\text{hist}}|} \sum_{(t_k,t_l)\in\mathcal{E}^T_{\text{hist}}} \nu_{kl}$ be its average

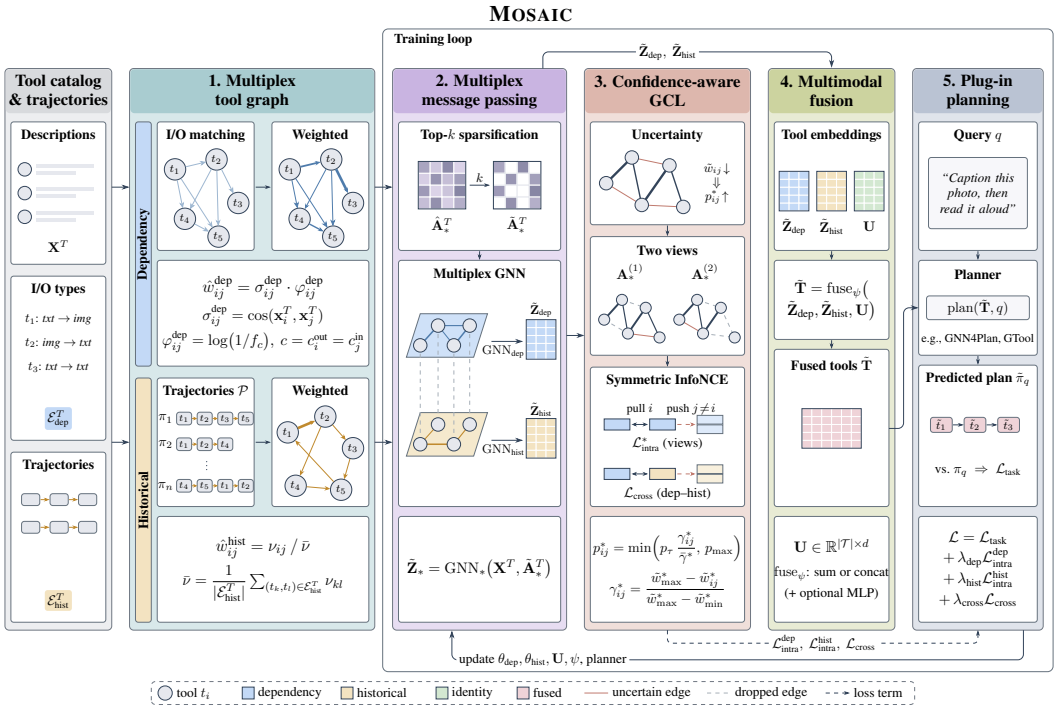


Figure 1: Overview of the MOSAIC pipeline, integrated on top of any graph-based task planner. **(1)** From the dependency and historical relations among tools, we build a weighted multiplex tool graph. **(2)** Both adjacency matrices are sparsified via top-k selection and processed by separate GNNs. **(3)** The resulting node embeddings feed a graph contrastive learning module, which generates two views of each graph through confidence-aware augmentation and optimizes an intra-level loss alongside a cross-level loss that aligns the two graph types. **(4)** All node representations are fused, together with a learnable identity embedding for each tool, into a multimodal tool embedding. **(5)** The fused embeddings are injected into the task planner, and the whole model is trained jointly with the planner-specific loss and the contrastive objectives.

over all edges. The edge weight is defined as:

$$\hat{w}_{ij}^{\text{hist}} = \frac{\nu_{ij}}{\bar{\nu}}, \tag{2}$$

so that frequently observed transitions receive larger weights, while the normalization by $\bar{\nu}$ keeps the average weight at one, as in the unweighted graph.

3.1.3 MULTIPLEX TOOL GRAPH MESSAGE PASSING

Graph sparsification. Although weighted, both graphs remain densely connected: the dependency graph links every pair of type-compatible tools, while the historical graph retains transitions observed in very few trajectories. Let $\hat{\mathbf{A}}_{\text{dep}}^T, \hat{\mathbf{A}}_{\text{hist}}^T \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{T}|}$ be the weighted adjacency matrices, with $\hat{\mathbf{A}}_*^T[i,j] = \hat{w}_{ij}^*$ from Equations 1 and 2, where $* \in \{\text{dep}, \text{hist}\}$. We sparsify both matrices by retaining, for each tool, only its k highest-weighted outgoing edges:

$$\tilde{\mathbf{A}}_*^T[i,j] = \begin{cases} \hat{\mathbf{A}}_*^T[i,j] & \text{if } \hat{\mathbf{A}}_*^T[i,j] \in \text{top-}k\big(\hat{\mathbf{A}}_*^T[i,:]\big), \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

and denote by $\tilde{\mathcal{E}}_*^T$ the retained edge set, with weights $\tilde{w}_{ij}^*$.

Multiplex graph neural networks. We treat the two sparsified graphs as the two relations of a *multiplex* tool graph over the same set of tools, and encode each relation with its own GNN:

$$\tilde{\mathbf{Z}}_* = \text{GNN}_*\big(\mathbf{X}^T, \tilde{\mathbf{A}}_*^T\big), \qquad * \in \{\text{dep}, \text{hist}\}, \tag{4}$$

where θ_{dep} and θ_{hist} denote the parameters of the two encoders. The encoders share the input features $\mathbf{X}^T$ but not their parameters, following relation-specific encoding in multiplex graph learning (Park et al., 2020); the two relations carry edge weights of different nature and scale, and separate encoders prevent their representations from collapsing onto each other, which keeps the cross-relation alignment described in the next section informative.

3.2 Confidence-aware graph contrastive learning

Edge weights let each GNN modulate messages according to their confidence, but they expose the model to a single, static version of each graph, so that residual noise in the retained edges still propagates into the tool representations. Therefore, we add a self-supervised objective based on graph contrastive learning (Velickovic et al., 2019; Zhu et al., 2020; You et al., 2020) (recalled in Appendix B), operating at two levels: an *intra-level* term, which makes the representations learned on each graph stable under confidence-aware perturbations, and a *cross-level* term, which aligns the representations of each tool across the two graphs.

Both terms rely on the InfoNCE loss (van den Oord et al., 2018; Chen et al., 2020). Given two representation matrices $\mathbf{Z}^{(1)}, \mathbf{Z}^{(2)} \in \mathbb{R}^{|\mathcal{T}| \times d}$ of the same tools, it pulls together the two representations of each tool t_i while pushing apart those of different tools:

$$\ell_i\big(\mathbf{Z}^{(1)}, \mathbf{Z}^{(2)}\big) = -\log \frac{\exp\big(\cos(\mathbf{z}^{(1)}[i], \mathbf{z}^{(2)}[i])/\tau\big)}{\sum_{j=1}^{|\mathcal{T}|} \exp\big(\cos(\mathbf{z}^{(1)}[i], \mathbf{z}^{(2)}[j])/\tau\big)}, \tag{5}$$

where $\mathbf{z}^{(v)}[i]$ denotes the i-th row of $\mathbf{Z}^{(v)}$ and τ is a temperature hyperparameter. Following Zhu et al. (2020), we always symmetrize it, so that neither side acts as a fixed anchor.

Intra-level graph contrastive learning. For each graph $* \in \{\text{dep}, \text{hist}\}$, we generate two views $\mathbf{A}_*^{(v)} = \tilde{\mathbf{A}}_*^T \odot \mathbf{M}_*^{(v)}$, $v \in \{1, 2\}$, through random edge-dropping masks $\mathbf{M}_*^{(v)}$. Standard practice samples each mask entry with a uniform dropout rate (Rong et al., 2020; Zhu et al., 2020), thereby removing reliable connections as often as spurious ones. Inspired by adaptive augmentation (Zhu et al., 2021), we instead drive dropout with the confidence weights computed at construction time. We first turn each weight into an uncertainty score through min-max normalization:

$$\gamma_{ij}^* = \frac{\tilde{w}_{\max}^* - \tilde{w}_{ij}^*}{\tilde{w}_{\max}^* - \tilde{w}_{\min}^*} \in [0, 1], \qquad (t_i, t_j) \in \tilde{\mathcal{E}}_*^T, \tag{6}$$

and then rescale it around a target average dropout rate p_τ, capped at $p_{\max} < 1$ so that no edge is dropped deterministically:

$$p_{ij}^* = \min\left(p_\tau \cdot \frac{\gamma_{ij}^*}{\bar{\gamma}^*},\ p_{\max}\right), \qquad \bar{\gamma}^* = \frac{1}{|\tilde{\mathcal{E}}_*^T|} \sum_{(t_k, t_l) \in \tilde{\mathcal{E}}_*^T} \gamma_{kl}^*. \tag{7}$$

Less confident edges are thus more likely to be dropped, while the average dropout probability equals p_τ before capping; if all weights coincide, we fall back to uniform dropout. The masks are sampled as:

$$M_*^{(v)}[i, j] \sim \textsc{Bernoulli}(1 - p_{ij}^*), \qquad v \in \{1, 2\}, \tag{8}$$

and both views are encoded by the relation-specific GNN of Equation 4, $\tilde{\mathbf{Z}}_*^{(v)} = \text{GNN}_*(\mathbf{X}^T, \mathbf{A}_*^{(v)})$. The symmetrized intra-level loss is:

$$\mathcal{L}_{\text{intra}}^* = \frac{1}{2|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} \Big[\ell_i\big(\tilde{\mathbf{Z}}_*^{(1)}, \tilde{\mathbf{Z}}_*^{(2)}\big) + \ell_i\big(\tilde{\mathbf{Z}}_*^{(2)}, \tilde{\mathbf{Z}}_*^{(1)}\big)\Big]. \tag{9}$$

Cross-level graph contrastive learning. The two graphs describe the same tools from different perspectives. Following multi-view and multiplex graph contrastive learning (Hassani & Khasahmadi, 2020; Park et al., 2020; Wang et al., 2021), we treat them as two natural views of each tool, whose positive pairs are given by the data rather than by sampled augmentations, and align the representations $\tilde{\mathbf{Z}}_{\text{dep}}$ and $\tilde{\mathbf{Z}}_{\text{hist}}$ of Equation 4:

$$\mathcal{L}_{\text{cross}} = \frac{1}{2|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} \Big[\ell_i\big(\tilde{\mathbf{Z}}_{\text{hist}}, \tilde{\mathbf{Z}}_{\text{dep}}\big) + \ell_i\big(\tilde{\mathbf{Z}}_{\text{dep}}, \tilde{\mathbf{Z}}_{\text{hist}}\big)\Big]. \tag{10}$$

3.3 MULTIMODAL TOOL LEARNING AND FUSION

Identity tool embeddings. Both graph-aware representations are tied to the tool graphs, which may still carry residual noise despite confidence weighting and contrastive regularization. Inspired by latent-factor models in recommender systems (Rendle et al., 2009), we therefore complement them with a representation that is entirely agnostic to the graph structure; each tool is mapped to a learnable *identity* embedding, collected in $\mathbf{U} \in \mathbb{R}^{|\mathcal{T}| \times d}$ and trained end-to-end with the planner. Being learned directly from the planning supervision, these embeddings can capture tool-specific signals that neither the textual descriptions nor the tool graphs encode.

Multimodal fusion. Each tool is thus represented by three complementary embeddings (interpretable as different modalities), which we combine into a single representation before feeding it to the planner:

$$\tilde{\mathbf{T}} = \text{fuse}_{\psi}\big(\tilde{\mathbf{Z}}_{\text{dep}}, \tilde{\mathbf{Z}}_{\text{hist}}, \mathbf{U}\big), \tag{11}$$

where $\text{fuse}_{\psi}(\cdot)$ is either an element-wise sum or a concatenation, optionally followed by a multi-layer perceptron with parameters ψ. We treat the fusion strategy as a hyperparameter selected on the validation set (Appendix E.2).

3.4 INTEGRATING MOSAIC WITHIN GRAPH-BASED TASK PLANNING

MOSAIC only acts on the tool representations, and can therefore be mounted on top of any graph-based task planner that learns tool representations from a tool graph. Following Definition 2.2, such a planner predicts a tool sequence $\tilde{\pi}_q = \text{plan}(\mathbf{T}, q)$ from tool representations $\mathbf{T}$ learned over a single tool graph. MOSAIC replaces $\mathbf{T}$ with the fused representations $\tilde{\mathbf{T}}$ of Equation 11, leaving the prediction function unchanged, and adds its contrastive terms to the planner's original task loss $\mathcal{L}_{\text{task}} = \mathcal{L}_{\text{plan}}\big(\text{plan}(\tilde{\mathbf{T}}, q), \pi_q\big)$:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_{\text{dep}}\mathcal{L}^{\text{dep}}_{\text{intra}} + \lambda_{\text{hist}}\mathcal{L}^{\text{hist}}_{\text{intra}} + \lambda_{\text{cross}}\mathcal{L}_{\text{cross}}, \tag{12}$$

where $\lambda_{\text{dep}}, \lambda_{\text{hist}}, \lambda_{\text{cross}}$ weight the intra-level losses on the two graphs and the cross-level loss, and are selected on the validation set (Appendix D.5). The whole model is trained end-to-end; Appendix C reports the complete procedure (Algorithm 1), and Appendix D.3 details the integration with GNN4Plan and GTool.

4 EXPERIMENTS AND RESULTS

In this section, we detail the experimental settings we followed to test the performance of MOSAIC, and then present and discuss the empirical results referring to four key research questions (RQs).

4.1 EXPERIMENTAL SETUP

Datasets. We evaluate MOSAIC on the HuggingFace, Multimedia, and Daily Life domains of TaskBench (Shen et al., 2024), following the experimental setting of GNN4Plan (Wu et al., 2024). For each domain, we use the provided dependency graph and construct the historical tool graph exclusively from ground-truth tool transitions in the training set. We use 3,000/500/500 queries for training/validation/test, retaining the same test set as GNN4Plan. Appendix D.1 provides additional dataset and graph statistics.

Metrics. Following TaskBench and GNN4Plan, we report Node F1-Score (n-F1), Link F1-Score (l-F1), and Accuracy (Acc) to capture tool selection, dependency prediction, and exact tool-set matching, respectively. Appendix D.2 provides further details about their calculation.

Backbones. We evaluate MOSAIC when plugged into two graph-based task planners, GNN4Plan (Wu et al., 2024) and GTool (Chen et al., 2026), which exploit graph structure through different planning architectures. Across our main experiments, we consider three LLM backbones: Qwen3-14B (Team, 2025), CodeLlama-13B (Rozière et al., 2023), and Mistral-7B (Jiang et al., 2023). Appendices D.3 and D.4 provide details on the planners, their MOSAIC instantiations, and the LLM backbones.

Implementation details. We follow the original architecture and training configuration of each backbone planner, using GraphSAGE (Hamilton et al., 2017) for GNN4Plan and TransformerConv (Shi et al., 2021) for GTool. For MOSAIC, separate GNN encoders model the dependency and historical graph views according to the multiplex formulation in Section 3.1.3. Appendix D.5 reports complete implementation, training, and hyperparameter-selection details.

Research questions. With our experiments, we aim to answer the following research questions:

- **RQ1)** Can MOSAIC improve the prediction performance of graph-based task planning methodologies when mounted on top of them?
- **RQ2)** What is the contribution of each building component of MOSAIC?
- **RQ3)** Is MOSAIC robust to targeted perturbations on both tool graph structures (i.e., dependency and historical)?
- **RQ4)** Does MOSAIC affect the efficiency of the graph-based task planner backbones it is integrated with?

4.2 OVERALL RESULTS (RQ1)

Table 1 compares the original GNN4Plan and GTool backbones with their MOSAIC-enhanced variants across the three LLMs and TaskBench domains. We report Node F1-Score, Link F1-Score, and Accuracy, together with the relative improvement Δ (%) of each +MOSAIC variant over its corresponding baseline. Across all configurations, MOSAIC improves the corresponding baseline on all three metrics. This consistency across GNN4Plan and GTool indicates that the improvements are not specific to a single backbone architecture.

The largest gains are observed with Mistral-7B, the smallest LLM considered in the main experiments. In particular, GTool+MOSAIC improves Accuracy by 22.25% on HuggingFace and 16.50% on Multimedia, together with Link F1-Score gains of 10.89% and 7.58%, respectively. Substantial gains are also observed with CodeLlama-13B: Accuracy improves by 9.55% for GNN4Plan on Multimedia and by 9.25% and 8.26% for GTool on HuggingFace and Multimedia, respectively. Improvements with Qwen3-14B are generally more moderate, while remaining positive across all metrics and domains. Additional results are reported in Appendix E, while Appendix F provides qualitative case studies analyzing both correct and incorrect tool selections and the instance-dependent effect of MOSAIC.

4.3 ABLATION STUDY (RQ2)

Table 2 assesses the contribution of the main components of MOSAIC on Multimedia with Mistral-7B, considering both GNN4Plan and GTool as backbone planners. In *w/o* $\mathcal{L}^*_{\text{intra}}$, we disable intra-level graph contrastive learning on both tool graphs, while *w/o* $\mathcal{L}^*_{\text{intra}} + \mathcal{L}_{\text{cross}}$ additionally disables cross-level graph contrastive learning. In *w/o Confidence Weighting*, we perform unweighted message passing, disable graph sparsification, and replace confidence-aware edge dropout with uniform dropout when generating the contrastive views. In *w/o Graph Sparsification*, we disable top-k sparsification while retaining the weighted tool graphs. Finally, *w/o Multimodal Fusion* retains only the dependency tool representation, whereas *w/o Identity Embedding* fuses the dependency and historical tool representations without the identity tool embeddings. All ablated variants underperform the complete MOSAIC configuration across both backbone planners and all metrics. For GNN4Plan, *w/o Confidence Weighting* yields the largest decrease in n-F1, from 78.76 to 76.76. For GTool, *w/o* $\mathcal{L}^*_{\text{intra}}$ reduces Accuracy from 69.20 to 67.20, with a further decrease to 65.20 when $\mathcal{L}_{\text{cross}}$ is also removed. Notably, all ablated variants remain above their corresponding baselines across all metrics, indicating that multiple components contribute to the task-planning gains of MOSAIC.

4.4 ROBUSTNESS TO TOOL GRAPH NOISE (RQ3)

We further assess robustness to structural noise by injecting random edges into the tool graphs at increasing noise ratios, from 10% to 90%. For perturbations of the dependency tool graph, newly injected edges are weighted according to the dependency weighting defined in Section 3.1.1; for the historical tool graph, their weights are sampled within the range of the observed historical edge

Table 1: Task planning performance across three datasets (HuggingFace, Multimedia, Daily Life), three backbone LLMs (i.e., Qwen3-14B, CodeLlama-13B, Mistral-7B), and two graph-based task planners (GNN4Plan, GTool), with and without the proposed MOSAIC framework. $\Delta\%$ denotes the percentage change of the +MOSAIC variant with respect to its corresponding base planner.

LLM	Model	HuggingFace			Multimedia			Daily Life		
		n-F1 ↑	l-F1 ↑	Acc ↑	n-F1 ↑	l-F1 ↑	Acc ↑	n-F1 ↑	l-F1 ↑	Acc ↑
Qwen3-14B	GNN4Plan	79.87	61.02	45.60	82.33	68.21	54.60	96.15	84.10	81.40
	GNN4Plan+MOSAIC	80.56	62.49	46.40	85.29	71.33	56.80	96.29	84.30	81.60
	Δ (%)	*+0.86*	*+2.40*	*+1.75*	*+3.59*	*+4.57*	*+4.03*	*+0.15*	*+0.24*	*+0.25*
	GTool	81.43	59.86	46.40	86.33	69.49	58.00	97.46	87.36	88.40
	GTool+MOSAIC	83.10	62.80	47.40	86.56	70.01	58.60	97.73	88.56	88.60
	Δ (%)	*+2.05*	*+4.91*	*+2.16*	*+0.27*	*+0.75*	*+1.03*	*+0.28*	*+1.37*	*+0.23*
CodeLlama-13B	GNN4Plan	72.16	49.36	30.26	77.60	58.42	40.70	93.40	79.98	71.84
	GNN4Plan+MOSAIC	73.25	49.97	31.46	79.47	60.82	44.58	94.14	81.69	74.49
	Δ (%)	*+1.51*	*+1.24*	*+3.97*	*+2.41*	*+4.10*	*+9.55*	*+0.79*	*+2.14*	*+3.69*
	GTool	81.71	61.46	45.40	87.85	72.67	58.01	97.04	86.40	82.60
	GTool+MOSAIC	82.17	63.46	49.60	89.18	74.64	62.80	97.55	88.48	86.20
	Δ (%)	*+0.56*	*+3.25*	*+9.25*	*+1.51*	*+2.71*	*+8.26*	*+0.53*	*+2.41*	*+4.36*
Mistral-7B	GNN4Plan	67.97	45.67	30.27	75.48	56.66	40.00	89.76	68.82	59.26
	GNN4Plan+MOSAIC	69.75	47.38	30.88	78.76	58.78	43.03	90.66	70.81	61.32
	Δ (%)	*+2.62*	*+3.74*	*+2.02*	*+4.34*	*+3.74*	*+7.58*	*+1.00*	*+2.90*	*+3.47*
	GTool	79.67	56.31	39.10	88.86	71.49	59.40	96.24	84.39	80.40
	GTool+MOSAIC	82.16	62.44	47.80	90.36	76.91	69.20	97.92	88.64	87.60
	Δ (%)	*+3.13*	*+10.89*	*+22.25*	*+1.69*	*+7.58*	*+16.50*	*+1.75*	*+5.04*	*+8.96*

Table 2: Ablation study isolating the contribution of each component of MOSAIC. Results are computed on Multimedia for both GNN4Plan and GTool backbones with Mistral-7B.

Setting	GNN4Plan			GTool		
	n-F1 ↑	l-F1 ↑	Acc ↑	n-F1 ↑	l-F1 ↑	Acc ↑
w/o $\mathcal{L}^{*}_{\text{intra}}$	78.19	57.81	40.81	90.01	76.81	67.20
w/o $\mathcal{L}^{*}_{\text{intra}} + \mathcal{L}_{\text{cross}}$	77.97	57.25	41.41	90.01	76.22	65.20
w/o *Confidence Weighting*	76.76	56.27	41.21	89.58	76.54	65.60
w/o *Graph Sparsification*	77.55	57.40	41.21	89.76	75.39	65.10
w/o *Multimodal Fusion*	77.13	56.51	41.01	89.75	75.74	65.80
w/o *Identity Embedding*	76.90	56.85	41.21	90.03	75.59	65.80
Baseline	75.48	56.66	40.00	88.86	71.49	59.40
Baseline+MOSAIC	**78.76**	**58.78**	**43.03**	**90.36**	**76.91**	**69.20**

weights. The perturbed graphs are used during both training and inference. We compare the unperturbed baseline with the same planner trained on a perturbed dependency tool graph, and with MOSAIC when noise is independently injected into either the dependency (MOSAIC (Dep.)) or historical (MOSAIC (Hist.)) tool graph. Figure 2 reports Accuracy on HuggingFace with CodeLlama-13B for both backbone planners.

For GNN4Plan, the perturbed baseline progressively degrades from 30.26% Accuracy without noise to 28.45% at 90% noise, whereas both MOSAIC variants remain above the clean baseline throughout the evaluated range. For GTool, the effect of perturbations is less monotonic, but MOSAIC remains more robust overall: under dependency-graph perturbations, it outperforms the perturbed baseline at every noise level, while perturbing the historical graph leads to larger fluctuations. These results indicate that jointly exploiting the two tool graph representations can mitigate the effect of noise affecting either relation, rather than making planning dependent on a single perturbed graph.

4.5 EFFICIENCY STUDY (RQ4)

Finally, we assess the computational overhead that MOSAIC introduces on both task planner backbones. Specifically, we measure (i) the average training time per epoch, (ii) the average inference time per query, and (iii) the average number of tokens consumed per query, and compare each backbone against its MOSAIC-augmented counterpart. Table 3 reports the results on Multimedia and Daily Life, with Qwen3-14B as the underlying LLM. We omit token consumption for GNN4Plan

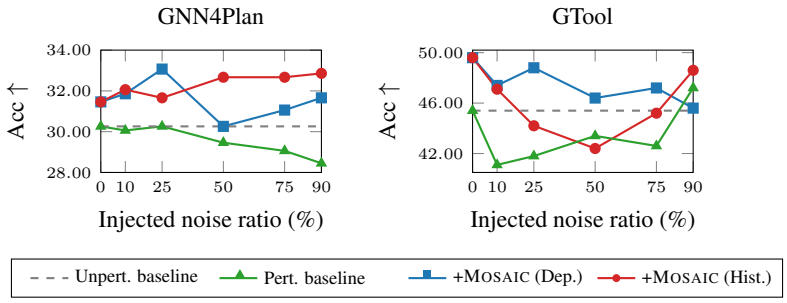


Figure 2: Accuracy performance under structural noise injection of random edges into the dependency and historical tool graphs, for HuggingFace and CodeLlama-13B. The dashed gray line indicates the unperturbed baseline, while the green, blue, and red lines stand for the perturbed baseline, and MOSAIC when perturbing either the dependency (Dep.) and historical (Hist.) tool graphs.

Table 3: Average training time per epoch and average inference time per query (expressed in seconds), alongside average tokens consumption per query on Multimedia and Daily Life, using Qwen3-14B.

Dataset	Model	Time / epoch (s)	Time / query (s)	Tokens / query
Multimedia	GNN4Plan	12.13	2e-5	–
	GNN4Plan+MOSAIC	20.16	3e-5	–
	GTool	2612.97	1.81	60.24
	GTool+MOSAIC	2588.33	1.90	64.00
Daily Life	GNN4Plan	18.34	2e-5	–
	GNN4Plan+MOSAIC	21.75	2e-5	–
	GTool	3681.38	1.38	40.33
	GTool+MOSAIC	3781.97	1.24	35.29

and GNN4Plan+MOSAIC, as neither uses LLM calls during training and both share the same LLM-generated query decomposition at inference. Thus, MOSAIC introduces no additional token consumption. Overall, MOSAIC adds only a marginal, and expected, overhead to both backbones. The most noticeable increase occurs at training time, where the average epoch duration grows slightly; we attribute this mainly to the graph contrastive learning module within MOSAIC. At inference time, the impact on both latency and token consumption remains limited. These findings indicate that MOSAIC achieves a favorable trade-off between planning performance and computational cost.

5 Conclusion and Future Work

In this work, we presented MOSAIC, a model-agnostic plug-in that strengthens the tool representation learning of existing planners without modifying their planning architecture. MOSAIC combines the historical and dependency graphs into a confidence-weighted multiplex tool graph, learns relation-specific tool representations through a confidence-aware graph contrastive objective that perturbs less reliable edges more aggressively and aligns each tool across the two relations, and fuses them with learnable identity embeddings that are agnostic to the tool graph. Mounted on GNN4Plan and GTool, MOSAIC improves every metric across three TaskBench domains and three LLMs, with relative accuracy gains of up to 22.25%, mitigates the effect of structural noise injected into the tool graphs, and adds only a marginal overhead, mostly at training time. Future work will first evaluate MOSAIC on newer planners, larger tool catalogs, and additional LLMs, since graph-based task planning is still an emerging research area. We also plan to replace the transductive identity embeddings with inductive ones, so as to represent tools unseen during training, and to condition edge confidences on the incoming query, capturing which connections matter for each request. Finally, we will reduce the training overhead through lighter contrastive objectives and explore adaptive strategies that select the most informative relations within the multiplex tool graph.

AI USE STATEMENT

Large language models were used exclusively as writing assistants, to refine and polish the language of the manuscript. They were not involved in the design of the methodology, the implementation, the execution of the experiments, or the analysis of the results. All content was carefully reviewed, verified, and edited by the authors, who take full responsibility for the entire work.

ETHICS STATEMENT

This work does not involve human subjects, user studies, or the collection of personal data. All experiments are conducted on publicly available benchmarks (TaskBench) and open-source models, used in accordance with their respective licenses. MOSAIC improves how existing task planners represent and select tools, and does not introduce new capabilities beyond those of the underlying planners and language models. We are not aware of any ethical concern arising from this work.

REPRODUCIBILITY STATEMENT

We made every effort to ensure the reproducibility of our results. The complete training procedure of MOSAIC is reported in Algorithm 1 (Appendix C). Details on the datasets and the construction of the tool graphs are provided in Appendix D.1, the evaluation metrics in Appendix D.2, and the integration of MOSAIC with each backbone planner in Appendix D.3. The LLMs used in our experiments are listed in Appendix D.4. The training configuration and the hyperparameter search spaces, including the model selection procedure, are described in Appendix D.5. To facilitate the replication of our experiments, we release the source code of MOSAIC, together with the scripts to reproduce all results, at an anonymous repository: `https://anonymous.4open.science/r/MOSAIC-86BA`.

REFERENCES

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*, pp. 17682–17690. AAAI Press, 2024.

Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiao Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. Graphllm: Boosting graph reasoning ability of large language model. *IEEE Trans. Big Data*, 12 (2):475–483, 2026.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1597–1607. PMLR, 2020.

Wenjie Chen, Di Yao, Wenbin Li, Xuying Meng, Chang Gong, and Jingping Bi. Gtool: Graph enhanced tool planning with large language model. In *International Conference on Learning Representations*, volume 2026, pp. 69247–69269, 2026.

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.

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.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li,

Hao Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, Tao Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nat.*, 645(8081):633–638, 2025.

Jiayan Guo, Lun Du, and Hengyu Liu. Gpt4graph: Can large language models understand graph structured data ? an empirical evaluation and benchmarking. *CoRR*, abs/2305.15066, 2023.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 1024–1034, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html.

Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4116–4126. PMLR, 2020.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V. Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. In *NeurIPS*, 2024.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *CoRR*, abs/2310.06825, 2023.

Bowen Jin, Gang Liu, Chi Han, Meng Jiang, Heng Ji, and Jiawei Han. Large language models on graphs: A comprehensive survey. *IEEE Trans. Knowl. Data Eng.*, 36(12):8622–8642, 2024.

Chen Kang and Kunyan Li. A lifecycle-oriented survey of llm-based agents: Workflow, graph augmentation, and future directions. *Neurocomputing*, 704:134838, 2026. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2026.134838. URL https://www.sciencedirect.com/science/article/pii/S0925231226022368.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Thomas N. Kipf and Max Welling. Variational graph auto-encoders. *CoRR*, abs/1611.07308, 2016.

Yuhan Li, Zhixun Li, Peisong Wang, Jia Li, Xiangguo Sun, Hong Cheng, and Jeffrey Xu Yu. A survey of graph meets large language model: Progress and future directions. In *IJCAI*, pp. 8123–8131. ijcai.org, 2024.

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*, pp. 30108–30134. PMLR / OpenReview.net, 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.

Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip S. Yu. Graph self-supervised learning: A survey. *IEEE Trans. Knowl. Data Eng.*, 35(6):5879–5900, 2023.

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*, pp. 89–105. Springer, 2024b.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Zihan Luo, Xiran Song, Hong Huang, Jianxun Lian, Chenhao Zhang, Jinqi Jiang, and Xing Xie. Graphinstruct: Empowering large language models with graph understanding and reasoning capability. *CoRR*, abs/2403.04483, 2024.

Chanyoung Park, Donghyun Kim, Jiawei Han, and Hwanjo Yu. Unsupervised attributed multiplex network embedding. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, pp. 5371–5378. AAAI Press, 2020.

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

Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Seyed Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. Let your graph do the talking: Encoding structured data for llms. *CoRR*, abs/2402.05862, 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.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. In Jeff A. Bilmes and Andrew Y. Ng (eds.), *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*, pp. 452–461. AUAI Press, 2009. URL https://www.auai.org/uai2009/papers/UAI2009_0139_48141db02b9f0b02bc7158819ebfa2c7.pdf.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net, 2020.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950, 2023.

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.

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 Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/085185ea97db31ae6dcac7497616fd3e-Abstract-Datasets_and_Benchmarks_Track.html.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. In Zhi-Hua Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pp. 1548–1554. ijcai.org, 2021. doi: 10.24963/IJCAI.2021/214. URL https://doi.org/10.24963/ijcai.2021/214.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Heung-Yeung Shum, and Jian Guo. Think-on-graph: Deep and responsible reasoning of large language model with knowledge graph. *CoRR*, abs/2307.07697, 2023.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. Graphgpt: Graph instruction tuning for large language models. In *SIGIR*, pp. 491–500. ACM, 2024.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748, 2018.

Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep graph infomax. In *7th International Conference on Learning Representations, ICLR 2019*. OpenReview.net, 2019.

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.

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. doi: 10.48550/ARXIV.2212.03533. URL https://doi.org/10.48550/arXiv.2212.03533.

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.

Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9929–9939. PMLR, 2020.

Xiao Wang, Nian Liu, Hui Han, and Chuan Shi. Self-supervised heterogeneous graph neural network with co-contrastive learning. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1726–1736. ACM, 2021.

Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. Self-supervised graph learning for recommendation. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 726–735. ACM, 2021.

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.

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.

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*, 2023a.

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, 2023b.

Ruosong Ye, Caiqi Zhang, Runhui Wang, Shuyuan Xu, and Yongfeng Zhang. Natural language is all a graph needs. *CoRR*, abs/2308.07134, 2023.

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.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*, 2020.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *NeurIPS*, pp. 5171–5181, 2018.

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

Xikun Zhang, Antoine Bosselut, Michihiro Yasunaga, Hongyu Ren, Percy Liang, Christopher D. Manning, and Jure Leskovec. Greaselm: Graph reasoning enhanced language models. In *ICLR*. OpenReview.net, 2022.

Zhongjian Zhang, Xiao Wang, Mengmei Zhang, Jiarui Tan, and Chuan Shi. Toward graph-tokenizing large language models with reconstructive graph instruction tuning. In *WWW*, pp. 430–441. ACM, 2026.

Jing Zhu, Xiang Song, Vassilis N. Ioannidis, Danai Koutra, and Christos Faloutsos. Touchup-g: Improving feature representation through graph-centric finetuning. In *SIGIR*, pp. 2662–2666. ACM, 2024.

Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep graph contrastive representation learning. *CoRR*, abs/2006.04131, 2020.

Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In *WWW '21: The Web Conference 2021*, pp. 2069–2080. ACM / IW3C2, 2021.

APPENDIX

TABLE OF CONTENTS

A Related Work **15**

B Background on Graph Contrastive Learning **16**

C Additional Methodological Details of MOSAIC **18**

D Experimental Settings **18**

D.1 Datasets 18

D.2 Metrics 19

D.3 Baseline implementations 21

D.4 Large language models 23

D.5 Training and evaluation configuration 23

E Additional Experimental Results **25**

E.1 Additional large language models 25

E.2 Hyperparameter sensitivity 25

E.3 Pre-training and fine-tuning on other datasets 25

F Case Studies **27**

G Computational Resources and Codebase **27**

H Limitations **28**

A RELATED WORK

LLM-based task planning. Task planning asks an LLM agent to turn a natural language request into an ordered set of tool invocations. Early solutions prompt the LLM to decompose the request into subtasks and assign a model or API to each of them (Shen et al., 2023; 2024), possibly with tree or graph structured reasoning (Yao et al., 2023a; Besta et al., 2024), while later ones fine-tune LLMs on large tool corpora (Qin et al., 2024; Yang et al., 2023), map each tool to a dedicated token (Wang et al., 2025), or refine the plan through execution feedback at the price of repeated LLM calls (Yao et al., 2023b). A recurring observation is that tools are not independent, since the output of a tool often feeds another one, and that neglecting these dependencies yields invalid plans (Chen et al., 2026; Gao et al., 2026; Kang & Li, 2026). Tool graphs make such dependencies explicit and, following Kang & Li (2026), are either *trajectory derived*, with edges taken from consecutive invocations in historical traces (Liu et al., 2024a), or *schema derived*, with edges linking tools whose input and output types are compatible (Liu et al., 2024b). These two families correspond to the historical and dependency tool graphs combined by MOSAIC. Graph-based planners use this structure to search the tool space (Liu et al., 2024a;b), to reason over graph shaped plans (Lin et al., 2024; Zhang et al., 2025), to retrieve dependent tools (Gao et al., 2025; Patel et al., 2026), or to plan over function level workflows (You et al., 2026).

Being very recent, several of these contributions are still available only as preprints (Liu et al., 2024a; Zhang et al., 2025; Gao et al., 2025; 2026; You et al., 2026), so their implementations and protocols may still change. We therefore focus on two planners that subsequent work consistently adopts as reference points. GNN4Plan (Wu et al., 2024), published at NeurIPS 2024, is the established one: it shows that attention biases and the autoregressive loss hinder LLMs when deciding over graphs, and delegates tool selection to a GNN. GTool (Chen et al., 2026), published at ICLR 2026, is the most recent peer reviewed one: it builds a request specific tool graph and feeds a GNN derived graph token to the LLM. GNN4Plan is a baseline for GTool, which is in turn a baseline for GRAFT and ToolLIFT (Gao et al., 2026; You et al., 2026), so the pair covers established and recent practice at once. These works also agree that the tool graph is far from reliable, and our motivation draws directly on this point. GNN4Plan lists the manual construction of the graph among its limitations (Wu et al., 2024), GTool states that graphs built from historical trajectories unavoidably contain erroneous and missing edges (Chen et al., 2026), DTDR finds that static dependency graphs bias the agent toward repeated calls (Patel et al., 2026), GRAFT argues that graph matching depends heavily on the quality of the graph (Gao et al., 2026), and ToolLIFT observes that rarely used tools remain sparsely connected (You et al., 2026). Kang & Li (2026) likewise list incorrect or incomplete edges among the open challenges of tool graphs.

LLMs over graphs. How LLMs should consume graph structured data has been studied well beyond planning (Jin et al., 2024; Li et al., 2024). A first strategy serializes the graph into text: GPT4Graph (Guo et al., 2023) and NLGraph (Wang et al., 2023) show that LLMs have basic graph reasoning abilities but are sensitive to the input format and to spurious correlations, and instruction tuning on graph problems (Luo et al., 2024; Ye et al., 2023) only partially closes the gap. GNN4Plan traces these failures to the lack of invariance to node permutations and to frequency based correlations induced by the autoregressive loss (Wu et al., 2024), while serialization also lengthens the prompt and loses structural information (Chen et al., 2026; Gao et al., 2026). A second strategy lets a graph encoder produce tokens or soft prompts for the LLM (Tang et al., 2024; Chai et al., 2026; Perozzi et al., 2024; He et al., 2024), and further alternatives encode text and graph jointly (Zhang et al., 2022), fine-tune on graph centric objectives (Zhu et al., 2024), or reason along paths of a knowledge graph (Sun et al., 2023). Graph-based planners inherit these choices: GNN4Plan moves tool retrieval to a GNN, GTool injects a single request level graph token, and GRAFT assigns a token to each tool and trains the LLM to reconstruct successor edges from its hidden states (Wu et al., 2024; Chen et al., 2026; Gao et al., 2026), in the spirit of reconstructive graph instruction tuning (Zhang et al., 2026). All of them, however, take the tool graph as given, and the only component acting on its edges is the missing dependency prediction of GTool, rooted in link prediction on graphs (Kipf & Welling, 2016; Zhang & Chen, 2018). MOSAIC is complementary: it leaves the backbone planner unchanged and makes the tool representations it receives robust to unreliable edges, building on the graph contrastive learning principles recalled in Appendix B.

B BACKGROUND ON GRAPH CONTRASTIVE LEARNING

This section recalls the general formalism of graph contrastive learning (GCL), on which the confidence-aware contrastive module of MOSAIC (Section 3.2) is built. We keep the notation of the main paper wherever possible. For a broader overview of self-supervised learning on graphs, we refer the reader to Liu et al. (2023).

Setting. Let $\mathcal{G} = \{\mathbf{X}, \mathcal{E}\}$ be a graph over a node set $\mathcal{V}$, where $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d_0}$ contains the input node features and $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ is the adjacency matrix induced by $\mathcal{E}$, possibly weighted. A graph neural network with learnable parameters θ maps the graph to node representations $\mathbf{Z} = \mathrm{GNN}_\theta(\mathbf{X}, \mathbf{A}) \in \mathbb{R}^{|\mathcal{V}| \times d}$, where $\mathbf{z}[i]$ denotes the representation of node i. GCL trains θ without labels, by asking the encoder to produce representations that remain stable under perturbations of the input graph which are not expected to change the meaning of each node (Velickovic et al., 2019; Zhu et al., 2020; You et al., 2020).

Graph augmentations and contrastive views. Let $\mathcal{Q}$ be a family of stochastic augmentation functions defined on graphs. At each training step, two functions $q^{(1)}, q^{(2)} \sim \mathcal{Q}$ are sampled and applied to $\mathcal{G}$, producing two views $\mathcal{G}^{(v)} = q^{(v)}(\mathcal{G}) = \{\mathbf{X}^{(v)}, \mathbf{A}^{(v)}\}$ for $v \in \{1, 2\}$. Typical choices

include node dropping, subgraph sampling, feature masking and edge perturbation (You et al., 2020). The augmentation most relevant to our work is edge dropping, originally studied as a regularizer for deep GNNs (Rong et al., 2020) and later adopted as a structural augmentation in GCL (Zhu et al., 2020). Given a binary mask $\mathbf{M}^{(v)}$, the perturbed adjacency matrix is:

$$\mathbf{A}^{(v)} = \mathbf{A} \odot \mathbf{M}^{(v)}, \qquad M^{(v)}[i,j] \sim \textsc{Bernoulli}(1 - p_{ij}), \tag{13}$$

where p_{ij} is the probability of removing the edge between nodes i and j. In the uniform case, $p_{ij} = p$ for every edge. Feature masking is defined in a similar way, by sampling a vector $\mathbf{m}^{(v)} \in \{0,1\}^{d_0}$ with entries drawn from $\textsc{Bernoulli}(1 - p_f)$ and setting $\mathbf{x}^{(v)}[i] = \mathbf{x}[i] \odot \mathbf{m}^{(v)}$ for all nodes. MOSAIC only perturbs the graph structure and shares the node features $\mathbf{X}^T$ across views, so only Equation 13 applies in our case.

Both views are processed by the same encoder, $\mathbf{Z}^{(v)} = \text{GNN}_\theta(\mathbf{X}^{(v)}, \mathbf{A}^{(v)})$. Several methods add a projection head g_ϕ, usually a small MLP, and compute the loss on $g_\phi(\mathbf{z}^{(v)}[i])$ rather than on $\mathbf{z}^{(v)}[i]$ (Chen et al., 2020; Zhu et al., 2020). For readability, we omit it below, which corresponds to taking g_ϕ as the identity.

Contrastive objective. For an anchor node i, the representation of the same node in the other view forms the positive pair, while the representations of all other nodes act as negatives. The standard objective is InfoNCE (van den Oord et al., 2018), also known as NT-Xent in the formulation of Chen et al. (2020):

$$\ell\Big(\mathbf{z}^{(1)}[i], \mathbf{z}^{(2)}[i]\Big) = -\log \frac{\exp\big(\cos(\mathbf{z}^{(1)}[i], \mathbf{z}^{(2)}[i])/\tau\big)}{\sum_{j=1}^{|\mathcal{V}|} \exp\big(\cos(\mathbf{z}^{(1)}[i], \mathbf{z}^{(2)}[j])/\tau\big)}, \tag{14}$$

where $\tau > 0$ is a temperature. This is the form adopted by MOSAIC. Zhu et al. (2020) also add intra-view negatives, namely the terms $\exp(\cos(\mathbf{z}^{(1)}[i], \mathbf{z}^{(1)}[j])/\tau)$ for $j \neq i$, to the denominator. Since neither view is privileged, the loss is usually averaged over both directions:

$$\mathcal{L}_{\text{cl}} = \frac{1}{2|\mathcal{V}|} \sum_{i=1}^{|\mathcal{V}|} \Big[\ell\Big(\mathbf{z}^{(1)}[i], \mathbf{z}^{(2)}[i]\Big) + \ell\Big(\mathbf{z}^{(2)}[i], \mathbf{z}^{(1)}[i]\Big)\Big]. \tag{15}$$

Minimizing Equation 14 maximizes a lower bound on the mutual information between the representations of the two views, which becomes tighter as the number of negatives grows (van den Oord et al., 2018). An alternative reading is given by Wang & Isola (2020): the numerator drives the *alignment* of positive pairs, while the denominator encourages the *uniformity* of representations on the unit hypersphere, which prevents collapse to a trivial solution. The temperature τ controls how strongly the loss penalizes negatives that lie close to the anchor. Earlier approaches such as Deep Graph Infomax (Velickovic et al., 2019) contrast node representations with a global graph summary through a discriminator; MOSAIC instead follows the node to node formulation above.

Adaptive augmentations. Uniform edge dropping gives every edge the same chance of being removed, so structurally important connections may be deleted as often as spurious ones. Zhu et al. (2021) address this by assigning each edge an importance score κ_{ij}, derived from the centrality of its endpoints, and by setting the removal probability as:

$$p_{ij} = \min\left(p_\tau \cdot \frac{\kappa_{\max} - \kappa_{ij}}{\kappa_{\max} - \mu_\kappa},\ p_{\max}\right), \tag{16}$$

where $\kappa_{\max}$ and μ_κ are the maximum and the average of the scores, p_τ controls the overall removal rate and $p_{\max}$ is a cut off that prevents any edge from being removed with certainty. Less important edges are thus removed more often, and the encoder learns to be invariant mainly to perturbations of weakly informative structure. The confidence-aware augmentation of MOSAIC (Equations 6 and 7) follows the same principle, with two differences: the importance of each edge is not derived from topological centrality but from the confidence weights $\tilde{w}^*_{ij}$ obtained at graph construction time (Equations 1 and 2), normalized into uncertainty scores γ^*_{ij}, and it is computed separately for the dependency and the historical tool graphs.

Multi-view and cross-view contrast. The two views need not come from random perturbations of one graph. Hassani & Khasahmadi (2020) contrast representations computed on the adjacency matrix with those computed on a diffusion matrix of the same graph. In multiplex networks, where the same nodes are linked by several relation types, Park et al. (2020) train one encoder per relation and combine them through a consensus objective, while Wang et al. (2021) contrast node representations obtained from two structurally different views of a heterogeneous graph. The cross-level loss of MOSAIC (Equation 10) belongs to this family: the dependency and the historical tool graphs share the node set $\mathcal{T}$ and the features $\mathbf{X}^T$, so they provide two natural views of each tool, and positive pairs are given by the data rather than by a sampled augmentation.

Contrastive learning as an auxiliary objective. GCL was first used for unsupervised pre-training, followed by a separate downstream stage. A common alternative in recommendation is to optimize the contrastive loss jointly with the supervised one, $\mathcal{L} = \mathcal{L}_{\text{main}} + \lambda \mathcal{L}_{\text{cl}}$, so that the contrastive term acts as a regularizer on the representations used by the main task (Wu et al., 2021). MOSAIC adopts this scheme (Equation 12), with one intra-level term for each tool graph and one cross-level term, each weighted by its own coefficient.

C ADDITIONAL METHODOLOGICAL DETAILS OF MOSAIC

Algorithm 1 depicts the formalism behind the main pipeline stages of MOSAIC, as described in Section 3 from the main paper.

Algorithm 1 MOSAIC training procedure

Require: Tools $\mathcal{T}$ with description embeddings $\mathbf{X}^T$ and input-output types; trajectories $\mathcal{P}$; training queries q with ground-truth plans π_q; graph-based planner $\text{plan}(\cdot)$

Multiplex tool graph construction (once, before training)
Build $\hat{\mathbf{A}}^T_{\text{dep}}, \hat{\mathbf{A}}^T_{\text{hist}}$ with confidence weights ▷ Eq. 1, 2
Sparsify into $\tilde{\mathbf{A}}^T_{\text{dep}}, \tilde{\mathbf{A}}^T_{\text{hist}}$ via top-k selection ▷ Eq. 3
Compute edge dropout probabilities p^*_{ij} for $* \in \{\text{dep}, \text{hist}\}$ ▷ Eq. 6, 7
for each batch of training queries **do**
    # Multiplex message passing
    $\tilde{\mathbf{Z}}_* \leftarrow \text{GNN}_*(\mathbf{X}^T, \tilde{\mathbf{A}}^T_*)$ for $* \in \{\text{dep}, \text{hist}\}$ ▷ Eq. 4
    # Confidence-aware graph contrastive learning
    for $* \in \{\text{dep}, \text{hist}\}$ **do**
        Sample masks $\mathbf{M}^{(1)}_*, \mathbf{M}^{(2)}_*$ and views $\mathbf{A}^{(v)}_* = \tilde{\mathbf{A}}^T_* \odot \mathbf{M}^{(v)}_*$ ▷ Eq. 8
        $\tilde{\mathbf{Z}}^{(v)}_* \leftarrow \text{GNN}_*(\mathbf{X}^T, \mathbf{A}^{(v)}_*)$ for $v \in \{1, 2\}$
        Compute $\mathcal{L}^*_{\text{intra}}$ on $\tilde{\mathbf{Z}}^{(1)}_*, \tilde{\mathbf{Z}}^{(2)}_*$ ▷ Eq. 9
    end for
    Compute $\mathcal{L}_{\text{cross}}$ on $\tilde{\mathbf{Z}}_{\text{dep}}, \tilde{\mathbf{Z}}_{\text{hist}}$ ▷ Eq. 10
    # Multimodal fusion and task planning
    $\tilde{\mathbf{T}} \leftarrow \text{fuse}_\psi(\tilde{\mathbf{Z}}_{\text{dep}}, \tilde{\mathbf{Z}}_{\text{hist}}, \mathbf{U})$ ▷ Eq. 11
    $\mathcal{L}_{\text{task}} \leftarrow \mathcal{L}_{\text{plan}}\big(\text{plan}(\tilde{\mathbf{T}}, q), \pi_q\big)$, averaged over the batch
    $\mathcal{L} \leftarrow \mathcal{L}_{\text{task}} + \lambda_{\text{dep}}\mathcal{L}^{\text{dep}}_{\text{intra}} + \lambda_{\text{hist}}\mathcal{L}^{\text{hist}}_{\text{intra}} + \lambda_{\text{cross}}\mathcal{L}_{\text{cross}}$ ▷ Eq. 12
    Update $\theta_{\text{dep}}, \theta_{\text{hist}}, \mathbf{U}, \psi$ and the trainable parameters of $\text{plan}(\cdot)$ by backpropagation on $\mathcal{L}$
end for
return trained encoders, identity embeddings, fusion module, and planner

D EXPERIMENTAL SETTINGS

D.1 DATASETS

Table 4 reports the statistics of the dependency and historical graphs used in our experiments. The dependency-graph topology is inherited from TaskBench (Shen et al., 2024), whereas the histori-

Table 4: Statistics of the three TaskBench domains and their dependency and historical graphs. Graph density is computed as $|E|/(|T|(|T|-1))$.

Dataset	#Tools	#Dep.	Dep. dens.	#Hist.	Hist. dens.	Train	Val	Test
HuggingFace	23	225	0.445	244	0.482	3,000	500	500
Multimedia	40	449	0.288	456	0.292	3,000	500	500
Daily Life	40	1,560	1.000	1,529	0.980	3,000	500	500

cal graph is constructed exclusively from the ground-truth task links observed in the training split; validation and test instances are never used for its construction.

The three domains exhibit markedly different graph structures. HuggingFace and Multimedia contain dependency graphs with densities of 0.45 and 0.29, respectively, whereas the Daily Life dependency graph is fully connected, containing all $40 \times 39 = 1{,}560$ possible directed non-self edges. The historical graphs have densities of 0.48, 0.29, and 0.98 for HuggingFace, Multimedia, and Daily Life, respectively. For their construction, repeated occurrences of the same directed task link are aggregated over the training split and weighted according to the transition-frequency formulation of Section 3.1.2.

Table 5 complements these aggregate statistics by reporting the top-3 and bottom-3 edges according to their weights for both graph types. For the dependency graph, the examples provide a qualitative indication that the weighting scheme captures meaningful differences among connections admitted by the original TaskBench topology. Across all three domains, top-ranked edges tend to connect tools with closely related functionalities, whereas bottom-ranked edges exhibit substantially weaker semantic or functional relationships. For example, `Text-to-Speech` → `Automatic Speech Recognition` in HuggingFace and `URL Extractor` → `Text Downloader` in Multimedia receive substantially higher weights than `Object Detection` → `Sentence Similarity` and `Image-to-Text` → `Article Spinner`, respectively. The same pattern is visible in the fully connected Daily Life dependency graph, where `book_car` → `book_flight` is ranked considerably higher than `do_tax_return` → `organize_meeting_online`. Thus, the edge weights provide a finer-grained signal than edge existence alone, particularly when input–output compatibility induces many possible connections.

The historical graph provides a complementary view based on transitions observed in the training plans. In HuggingFace, `Audio-to-Audio` → `Automatic Speech Recognition` is observed 74 times and receives a weight of 4.26, compared with 0.06 for the bottom-ranked, single-occurrence transitions. A similar pattern emerges in Multimedia, where `Image Search (by Image)` → `Image-to-Text` occurs 40 times and receives a weight of 3.80, while single-occurrence transitions receive 0.10. In Daily Life, the corresponding values are 10 occurrences and 2.58 for the top-ranked transitions, compared with one occurrence and 0.26 for the bottom-ranked ones. These examples show how historical weighting assigns higher confidence to recurrent tool transitions and lower confidence to transitions that are only sporadically observed in the training plans.

D.2 Metrics

We evaluate tool-planning performance using Node F1 (n-F1), Link F1 (l-F1), and Accuracy (Acc). We compute the metrics independently for each test instance and then macro-average them over the full evaluation set.

Let $\mathcal{V}$ and $\hat{\mathcal{V}}$ denote the sets of ground-truth and predicted tool nodes, respectively, and let $\mathcal{E}$ and $\hat{\mathcal{E}}$ denote the corresponding sets of ground-truth and predicted links.

Node F1 (n-F1). Node F1 evaluates whether the model identifies the correct set of tools needed to solve a request, regardless of order. For a single instance, node precision and recall are defined as:

$$P_{\text{node}} = \frac{|\hat{\mathcal{V}} \cap \mathcal{V}|}{|\hat{\mathcal{V}}|}, \qquad R_{\text{node}} = \frac{|\hat{\mathcal{V}} \cap \mathcal{V}|}{|\mathcal{V}|}. \tag{17}$$

Table 5: Representative high- and low-weight edges from the dependency (Dep.) and historical (Hist.) graphs. Historical counts report the number of occurrences of each transition in the training plans.

Dataset	Graph	Weight ranking	Source → Target	Count	Weight
HuggingFace	Dep.	Top-3	`Audio-to-Audio` → `Audio Classification`	–	3.15
			`Text-to-Speech` → `Automatic Speech Recognition`	–	3.09
			`Text-to-Speech` → `Audio-to-Audio`	–	2.81
		Bottom-3	`Object Detection` → `Sentence Similarity`	–	0.14
			`Image Classification` → `Text-to-Speech`	–	0.14
			`Audio Classification` → `Translation`	–	0.14
	Hist.	Top-3	`Audio-to-Audio` → `Automatic Speech Recognition`	74	4.26
			`Text-to-Speech` → `Audio-to-Audio`	73	4.21
			`Audio-to-Audio` → `Audio Classification`	65	3.74
		Bottom-3	`Sentence Similarity` → `Question Answering`	1	0.06
			`Image Editing` → `Sentence Similarity`	1	0.06
			`Text-to-Speech` → `Translation`	1	0.06
Multimedia	Dep.	Top-3	`URL Extractor` → `Text Downloader`	–	3.87
			`URL Extractor` → `Image Downloader`	–	3.49
			`URL Extractor` → `Video Downloader`	–	3.41
		Bottom-3	`Image-to-Text` → `Article Spinner`	–	0.20
			`Text Grammar Checker` → `URL Extractor`	–	0.21
			`Text Sentiment Analysis` → `Article Spinner`	–	0.21
	Hist.	Top-3	`Image Search (by Image)` → `Image-to-Text`	40	3.80
			`URL Extractor` → `Audio Downloader`	38	3.61
			`URL Extractor` → `Image Downloader`	38	3.61
		Bottom-3	`Text Summarizer` → `Audio Effects`	1	0.10
			`Video-to-Text` → `Voice Changer`	1	0.10
			`Image-to-Text` → `Video Voiceover`	1	0.10
Daily Life	Dep.	Top-3	`book_car` → `book_flight`	–	3.20
			`online_shopping` → `sell_item_online`	–	3.17
			`sell_item_online` → `online_shopping`	–	3.17
		Bottom-3	`do_tax_return` → `organize_meeting_online`	–	1.38
			`apply_for_passport` → `get_news_for_topic`	–	1.40
			`apply_for_passport` → `organize_meeting_online`	–	1.41
	Hist.	Top-3	`consult_lawyer_online` → `sell_item_online`	10	2.58
			`pay_for_credit_card` → `get_news_for_topic`	10	2.58
			`stock_operation` → `enroll_in_course`	10	2.58
		Bottom-3	`make_video_call` → `play_music_by_title`	1	0.26
			`online_shopping` → `set_alarm`	1	0.26
			`buy_insurance` → `set_alarm`	1	0.26

Node F1 is their harmonic mean:

$$\text{n-F1} = \frac{2P_{\text{node}}R_{\text{node}}}{P_{\text{node}} + R_{\text{node}}}. \tag{18}$$

Link F1 (l-F1). Link F1 evaluates whether the model recovers the correct tool dependencies. Each directed link is treated as an element of a set, and a predicted link is considered correct only when it exactly matches a ground-truth link. Link precision and recall are:

$$P_{\text{link}} = \frac{|\hat{\mathcal{E}} \cap \mathcal{E}|}{|\hat{\mathcal{E}}|}, \qquad R_{\text{link}} = \frac{|\hat{\mathcal{E}} \cap \mathcal{E}|}{|\mathcal{E}|}, \tag{19}$$

and Link F1 is computed as:

$$\text{l-F1} = \frac{2P_{\text{link}}R_{\text{link}}}{P_{\text{link}} + R_{\text{link}}}. \tag{20}$$

Accuracy (Acc). We additionally report an instance-level accuracy derived from Node F1. A prediction is counted as correct when its n-F1 score is at least 0.99:

$$\text{Acc}_i = \mathbb{I}\left[\text{n-F1}_i \geq 0.99\right], \tag{21}$$

where $\mathbb{I}[\cdot]$ is the indicator function. Consequently, the reported accuracy corresponds to the proportion of test instances for which the predicted tool set achieves an n-F1 of at least 0.99. This metric depends only on node selection and does not require the predicted links to match the ground-truth links.

D.3 BASELINE IMPLEMENTATIONS

GNN4Plan. GNN4Plan formulates task planning as a sequential tool-retrieval problem over a tool graph. Given a query q, an LLM first decomposes it into atomic subtasks $\{s_1, \ldots, s_m\}$; then, using the notation of Section 2, let $\mathcal{G}^T = (\mathbf{X}^T, \mathcal{E}^T)$ denote the tool graph over the available tool set $\mathcal{T} = \{t_1, \ldots, t_{|\mathcal{T}|}\}$. GNN4Plan retrieves, for each subtask s_i, a corresponding tool t_i, yielding a tool sequence $\pi_q = \langle t_1, \ldots, t_m \rangle$ forming a valid path, or more generally a subgraph, in $\mathcal{G}^T$.

Each training instance provides the subtasks $\{s_1, \ldots, s_m\}$ associated with query q, together with the corresponding ground-truth tool sequence $\pi_q = \langle t_1, \ldots, t_m \rangle$. Since each subtask is associated with one ground-truth tool, GNN4Plan trains on the resulting subtask–tool pairs $\{(s_i, t_i)\}_{i=1}^m$.

Each subtask s_i is encoded using the pretrained language model e5-335M (Wang et al., 2022) into the representation $\mathbf{x}_i^S$. The same language model is used to encode the textual description of each tool t_j, yielding its initial representation $\mathbf{x}_j^T$. A GraphSAGE encoder (Hamilton et al., 2017) is then applied to the tool graph to obtain a graph-aware representation $\mathbf{t}_j$ for each tool t_j, i.e., the j-th row of the tool representations $\mathbf{T}$ of Definition 2.2.

To train the GNN, GNN4Plan casts tool retrieval as a binary ranking problem and adopts the Bayesian Personalized Ranking (BPR) loss (Rendle et al., 2009). For each positive subtask–tool pair (s_i, t_i), negative tools t_j are sampled among tools whose textual descriptions are similar to that of t_i. Following the original implementation, two negative tools are sampled for each positive pair. For a sampled negative tool t_j, the BPR objective is:

$$\mathcal{L}_{\mathrm{BPR}} = -\log \sigma \left(\langle \mathbf{t}_i, \mathbf{x}_i^S \rangle - \langle \mathbf{t}_j, \mathbf{x}_i^S \rangle \right), \tag{22}$$

which encourages the ground-truth tool t_i to receive a higher matching score with subtask s_i than a sampled negative tool.

At inference time, each subtask s_i is encoded into $\mathbf{x}_i^S$, and the corresponding tool is retrieved using the representations produced by the trained GraphSAGE encoder. The first tool $\tilde{t}_1$ is selected from the entire tool catalog, while each subsequent tool is selected among the neighbors of the previously retrieved tool:

$$\tilde{t}_i = \arg \max_{t_j \in \mathcal{N}_{\mathrm{dep}}(\tilde{t}_{i-1})} \left\langle \mathbf{t}_j, \mathbf{x}_i^S \right\rangle, \tag{23}$$

where $\mathcal{N}_{\mathrm{dep}}(\tilde{t}_{i-1})$ denotes the candidate tools connected to the previously selected tool according to the dependency tool graph. The resulting sequence $\tilde{\pi}_q = \langle \tilde{t}_1, \ldots, \tilde{t}_m \rangle$ constitutes the predicted plan.

GNN4Plan + MOSAIC. We integrate MOSAIC into GNN4Plan by replacing its original single-graph tool representation learning module with the multiplex tool representation learning pipeline introduced in Section 3.1.3, while preserving GNN4Plan's original subtask–tool scoring, BPR objective, and sequential retrieval procedure.

Following GNN4Plan, we use GraphSAGE (Hamilton et al., 2017) as the GNN backbone. To support the confidence-weighted tool graphs introduced by MOSAIC, we implement a weighted GraphSAGE variant in which each neighboring representation is scaled by the corresponding retained edge weight before aggregation:

$$\mathbf{m}_{*,i}^{(l)} = \mathrm{MEAN}_{t_j \in \mathcal{N}_*(t_i)} \left(\tilde{w}_{ji}^* \, \mathbf{h}_{*,j}^{(l-1)} \right), \qquad * \in \{\mathrm{dep}, \mathrm{hist}\}, \tag{24}$$

where $\mathcal{N}_*(t_i)$ denotes the neighbors of t_i in the retained edge set $\tilde{\mathcal{E}}_*^T$ and $\mathbf{h}_{*,j}^{(l)}$ the representation of t_j at layer l. We instantiate this encoder independently on the historical and dependency tool graphs, consistently with the multiplex message-passing formulation of Section 3.1.3. The two encoders yield the graph-aware tool representations $\tilde{\mathbf{Z}}_{\mathrm{hist}}$ and $\tilde{\mathbf{Z}}_{\mathrm{dep}}$, respectively. These representations are processed by the confidence-aware graph contrastive learning module described in Section 3.2 and subsequently fused with the identity tool embeddings $\mathbf{U}$ through the multimodal fusion module of Section 3.3, yielding:

$$\tilde{\mathbf{T}} = \mathrm{fuse}_\psi \left(\tilde{\mathbf{Z}}_{\mathrm{dep}}, \tilde{\mathbf{Z}}_{\mathrm{hist}}, \mathbf{U} \right), \tag{25}$$

as defined in Equation 11.

For GNN4Plan+MOSAIC, the j-th fused tool representation $\tilde{\mathbf{t}}_j$ replaces the baseline GraphSAGE representation $\mathbf{t}_j$ in GNN4Plan's inner-product scoring function. GNN4Plan then applies its original BPR objective using the fused tool representations; accordingly, the planner-specific loss in the overall MOSAIC objective is instantiated as:

$$\mathcal{L}_{\text{task}} = \mathcal{L}_{\text{BPR}}. \tag{26}$$

The complete model is therefore optimized jointly according to Equation 12, together with the intra- and cross-level contrastive objectives introduced by MOSAIC.

At inference time, GNN4Plan retains its original sequential retrieval procedure, with candidate tools scored using the fused MOSAIC representations:

$$\tilde{t}_i = \arg \max_{t_j \in \mathcal{N}_{\text{dep}}(\tilde{t}_{i-1})} \left\langle \tilde{\mathbf{t}}_j, \mathbf{x}_i^S \right\rangle. \tag{27}$$

As in the original GNN4Plan inference procedure, the dependency-graph topology is retained to constrain the candidate set for each retrieval step.

GTool. For the GTool baseline (Chen et al., 2026), we use the authors' original implementation and preserve its graph construction, training procedure, and inference pipeline. Let $\mathcal{T} = \{t_1, \ldots, t_{|\mathcal{T}|}\}$ denote the available tool set, and let $\mathbf{X}^T$ denote the initial tool representations obtained by encoding the textual descriptions of the tools, following the notation introduced in Section 2.

GTool constructs a directed graph from the tool-use trajectories observed in the training data. Each node represents a tool, and an edge from t_i to t_j is added when t_j is invoked immediately after t_i in an observed trajectory. Thus, the graph GTool uses is a historical tool graph based on previously observed tool transitions.

For each request q, GTool adds a request-specific node v_q to the historical tool graph. The request text is encoded as $\mathbf{x}_q$ and appended to the initial tool representations $\mathbf{X}^T$. The resulting node-feature matrix and the request-specific graph encoding are defined as

$$\begin{aligned} \mathbf{X}_{\text{GTool}}(q) &= \begin{bmatrix} \mathbf{X}^T \\ \mathbf{x}_q \end{bmatrix}, \\ \begin{bmatrix} \mathbf{H}^T \\ \mathbf{h}_q \end{bmatrix} &= \text{GNN}_{\text{GTool}}\left(\mathbf{X}_{\text{GTool}}(q), \mathbf{A}_{\text{GTool}}(q)\right), \end{aligned} \tag{28}$$

where $\mathbf{A}_{\text{GTool}}(q)$ denotes the adjacency matrix of the request-specific graph, including both the historical tool–tool connections and the connections between the tool nodes and the request node. As shown in Equation 28, the GNN jointly updates the representations of the tool nodes and the request node, producing $\mathbf{H}^T$ and $\mathbf{h}_q$, respectively. Through message passing, the request node receives information from the tool graph, allowing $\mathbf{h}_q$ to capture both the semantics of the input request and information from the available tools and their historical relations. GTool therefore uses $\mathbf{h}_q$ as the request-specific graph representation and injects it into the frozen language model to generate the corresponding tool sequence.

This graph representation is injected into the language model's input embedding sequence, along with the tool names and the original user request. The language model then autoregressively generates the predicted sequence of tool calls.

The language-model parameters remain frozen during training, while the graph-related parameters are optimized. In addition to the tool-planning loss $\mathcal{L}_{\text{plan}}$, GTool uses an auxiliary missing-dependency prediction loss $\mathcal{L}_{\text{MDP}}$. To compute $\mathcal{L}_{\text{MDP}}$, a subset of the graph edges is masked and the model is trained to distinguish existing tool dependencies from non-existing ones. The complete GTool training objective is

$$\mathcal{L}_{\text{GTool}} = \mathcal{L}_{\text{plan}} + \lambda_{\text{MDP}} \mathcal{L}_{\text{MDP}}. \tag{29}$$

GTool + MOSAIC. We integrate MOSAIC into GTool by modifying the way tool representations are learned, while keeping the remaining GTool planning pipeline unchanged. In particular, instead of relying only on the historical tool graph used by the original GTool model, MOSAIC represents the tool catalog using both the historical and dependency graphs introduced in Section 3.

We keep the two graphs separate during message passing. Starting from the same description-based tool features $\mathbf{X}^T$, two independent GNN encoders produce the corresponding tool representations:

$$\begin{aligned}\tilde{\mathbf{Z}}_{\text{dep}} &= \text{GNN}_{\text{dep}}(\mathbf{X}^T, \tilde{\mathbf{A}}^T_{\text{dep}}),\\ \tilde{\mathbf{Z}}_{\text{hist}} &= \text{GNN}_{\text{hist}}(\mathbf{X}^T, \tilde{\mathbf{A}}^T_{\text{hist}}).\end{aligned} \tag{30}$$

Here, $\tilde{\mathbf{A}}^T_{\text{dep}}$ and $\tilde{\mathbf{A}}^T_{\text{hist}}$ are the sparsified dependency and historical adjacency matrices defined in Section 3.1.3. Keeping the two channels separate allows the model to preserve the different information captured by input–output dependencies and historical tool transitions.

Following Equation 11, the two graph-based representations are then combined with the learned identity embeddings $\mathbf{U}$:

$$\tilde{\mathbf{T}} = \text{fuse}_\psi\left(\tilde{\mathbf{Z}}_{\text{dep}}, \tilde{\mathbf{Z}}_{\text{hist}}, \mathbf{U}\right). \tag{31}$$

The resulting representation $\tilde{\mathbf{T}}$ is used as the MOSAIC-enhanced representation of the tool catalog. It replaces the original tool representation used by GTool before its request-specific processing. We keep the remaining components of GTool unchanged, including the request node, graph-to-LLM interface, and frozen language-model planner. Therefore, MOSAIC changes how the tool representations are learned rather than introducing a new downstream planning architecture. During training, we retain the original GTool losses and add the contrastive objectives introduced by MOSAIC. The resulting objective is

$$\begin{aligned}\mathcal{L}_{\text{GTool+MOSAIC}} = \mathcal{L}_{\text{plan}} &+ \lambda_{\text{MDP}}\mathcal{L}_{\text{MDP}}\\ &+ \lambda_{\text{dep}}\mathcal{L}^{\text{dep}}_{\text{intra}} + \lambda_{\text{hist}}\mathcal{L}^{\text{hist}}_{\text{intra}} + \lambda_{\text{cross}}\mathcal{L}_{\text{cross}}\end{aligned} \tag{32}$$

The first two terms correspond to the original GTool objective, while the remaining terms are the dependency-level, historical-level, and cross-graph contrastive losses introduced in Equation 12. At inference time, these auxiliary objectives are not required; the learned MOSAIC representations are directly used by the original GTool inference pipeline.

D.4 LARGE LANGUAGE MODELS

We conduct our experiments using three open-source LLM backbones for the results described in the main paper: **Qwen3-14B** (Team, 2025)[1], **CodeLlama-13B** (Rozière et al., 2023)[2], and **Mistral-7B** (Jiang et al., 2023)[3]. Additionally, we use **DeepSeek-R1** (Guo et al., 2025)[4]) to compute some of the additional results reported in Appendix E. These models cover different model families and parameter scales, allowing us to evaluate the considered graph-based task-planning approaches across different LLM backbones.

D.5 TRAINING AND EVALUATION CONFIGURATION

We follow the original training protocol of each backbone planner unless otherwise specified. GNN4Plan is trained for up to 20 epochs using Adam with a batch size of 512 and early stopping with a patience of 5 epochs. Given its relatively low training cost, we explore learning rates in {1e-3, 1e-4} for both GNN4Plan and GNN4Plan+MOSAIC. For GTool, we retain the learning rate 1e-5 from the original configuration and train for 10 epochs. Following preliminary experiments with different multimodal fusion strategies, we use concatenation followed by an MLP throughout the main experiments.

Hyperparameter selection is performed through grid search exclusively on the validation set. To account for the different evaluation criteria of the two backbone planners, we define a validation SCORE used to rank the candidate configurations:

[1] `Qwen/Qwen3-14B`.
[2] `codellama/CodeLlama-13b-hf`.
[3] `mistralai/Mistral-7B-v0.3`.
[4] `deepseek-ai/DeepSeek-R1`.

$$\text{SCORE}_{\text{GNN4Plan}} = \frac{\text{n-F1} + \text{l-F1} + \text{Acc}}{3}.$$

$$\text{SCORE}_{\text{GTool}} = \frac{\text{n-F1} + \text{l-F1}}{2}.$$

For each experimental setting, we select the configuration achieving the highest validation SCORE and report its performance on the held-out test set. All experiments are run with a single random seed. Table 6 summarizes the fixed settings and hyperparameter search spaces used in our experiments.

Table 6: Training configuration and hyperparameter search space. Values in braces denote the grid-search space, with model selection performed on the validation set; single values denote fixed settings.

Setting	**Search space**
GNN4Plan configuration	
Text encoder	`e5-335M`
Embedding dimension	1024
GNN hidden dimension	1024
GNN layers	1
Baseline GNN	GraphSAGE
+MOSAIC GNN	Weighted GraphSAGE
Learning rate	{1e-3, 1e-4}
Optimizer	Adam Optimizer (Kingma & Ba, 2015)
Batch size	512
Training epochs	20
Early-stopping patience	5
Negative samples per positive	2
GTool configuration	
Text encoder	`all-roberta-large-v1`
Embedding dimension	1024
GNN	TransformerConv
GNN hidden dimension	1024
GNN layers	3
GNN heads	4
Learning rate	1e-5
Optimizer	AdamW (Loshchilov & Hutter, 2019)
Batch size	4
Grad steps	2
Training epochs	10
MOSAIC *configuration*	
Multimodal fusion	{Concatenation + MLP, Sum, Sum + MLP}
InfoNCE temperature τ	{0.1, 0.2}
λ_{dep}	{10, 1, 1e-1, 1e-2, 1e-4, 1e-6}
λ_{hist}	{10, 1, 1e-1, 1e-2, 1e-4, 1e-6}
λ_{cross}	{10, 1, 1e-1, 1e-2, 1e-4, 1e-6}
Target average dropout p_τ	{0.05, 0.1, 0.2, 0.4, 0.6}
Maximum dropout probability $p_{\max}$	{0.4, 0.6}
k	{5 ,10, 15}
Contrastive views	2

We additionally report a sensitivity analysis over the considered values of p_τ and top-k, together with alternative multimodal fusion strategies, in Appendix E.2.

E Additional Experimental Results

E.1 Additional large language models

To further assess the generalization of MOSAIC, we evaluate it with an additional open-source LLM, DeepSeek-R1. Table 7 reports the results, which complement the experimental setting of Section 4.2. Consistent with the findings in the main paper, integrating MOSAIC on top of GNN4Plan and GTool improves their performance on all datasets, with the largest gains on HuggingFace and Multimedia.

Table 7: Task planning performance across three benchmark datasets (HuggingFace, Multimedia, Daily Life), with DeepSeek-R1 backbone, and two graph-based task planners (GNN4Plan, GTool), with and without the proposed MOSAIC framework. $\Delta\%$ denotes the relative percentage change of the +MOSAIC variant with respect to its corresponding base planner, computed independently for each metric.

Model	HuggingFace			Multimedia			Daily Life		
	n-F1 ↑	l-F1 ↑	Acc ↑	n-F1 ↑	l-F1 ↑	Acc ↑	n-F1 ↑	l-F1 ↑	Acc ↑
GNN4Plan	73.37	52.84	34.38	78.63	61.67	45.06	93.77	79.59	72.84
GNN4Plan+MOSAIC	73.86	53.91	35.96	80.59	64.86	50.80	94.03	79.60	73.64
Δ (%)	*+0.67*	*+2.02*	*+4.58*	*+2.50*	*+5.16*	*+12.76*	*+0.28*	*+0.02*	*+1.10*
GTool	78.23	54.74	36.40	84.42	66.07	52.80	96.36	85.10	83.80
GTool+MOSAIC	79.92	58.03	40.60	89.14	74.04	60.80	98.06	88.87	88.80
Δ (%)	*+2.16*	*+6.01*	*+11.54*	*+5.59*	*+12.06*	*+15.15*	*+1.76*	*+4.43*	*+5.97*

E.2 Hyperparameter sensitivity

We further analyze the sensitivity of MOSAIC to the target average dropout rate p_τ, the graph sparsification parameter k, and the multimodal fusion strategy. For each backbone planner and LLM, we start from the configuration selected on the validation set and vary one component at a time while keeping all remaining settings fixed. We report the analysis on Daily Life for Mistral-7B and DeepSeek-R1.

Target average dropout rate. Figures 3a and 3b report the effect of p_τ with Mistral-7B, while Figures 4a and 4b report the corresponding results with DeepSeek-R1. For Mistral-7B, both planners achieve their strongest performance around $p_\tau = 0.1$, with only limited variation over the remaining values. With DeepSeek-R1, GNN4Plan remains stable for p_τ values between 0.05 and 0.4, with a slight performance decrease only at 0.6, whereas GTool shows greater sensitivity to the dropout rate and achieves its best performance at $p_\tau = 0.4$.

Graph sparsification. Figures 3c and 3d vary the top-k sparsification parameter with Mistral-7B, while Figures 4c and 4d show the same analysis with DeepSeek-R1. GNN4Plan consistently favors $k = 10$ across both LLMs. GTool exhibits only small differences across the considered values, with the preferred k varying between configurations and metrics.

Multimodal fusion. Figures 5a and 5b compare element-wise summation, summation followed by an MLP, and concatenation followed by an MLP for Mistral-7B and DeepSeek-R1, respectively. For GNN4Plan, concatenation followed by an MLP achieves the highest Accuracy in both settings, although the differences among fusion strategies are small. For GTool, the preferred fusion operator depends on the LLM: summation followed by an MLP performs best with Mistral-7B, whereas element-wise summation performs best with DeepSeek-R1.

E.3 Pre-training and fine-tuning on other datasets

We further assess how well MOSAIC transfers across datasets. Specifically, we pre-train both backbones equipped with MOSAIC on HuggingFace and Multimedia, and then fine-tune them on each of

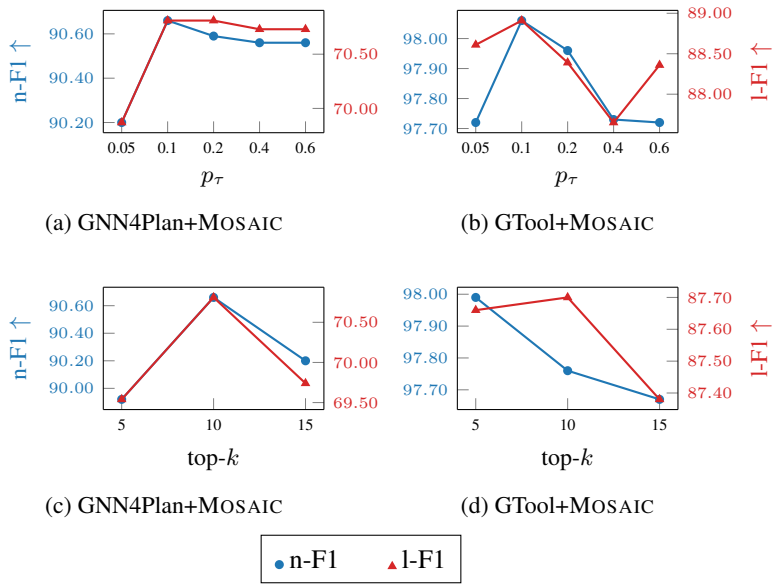


Figure 3: Hyperparameter sensitivity analysis for $p_\tau \in \{0.05, 0.1, 0.2, 0.4, 0.6\}$ and top-$k \in \{5, 10, 15\}$ on the n-F1 (left axis) and l-F1 (right axis) metrics, for GNN4Plan+MOSAIC (left column) and GTool+MOSAIC (right column). The considered setting is Mistral-7B on Daily Life.

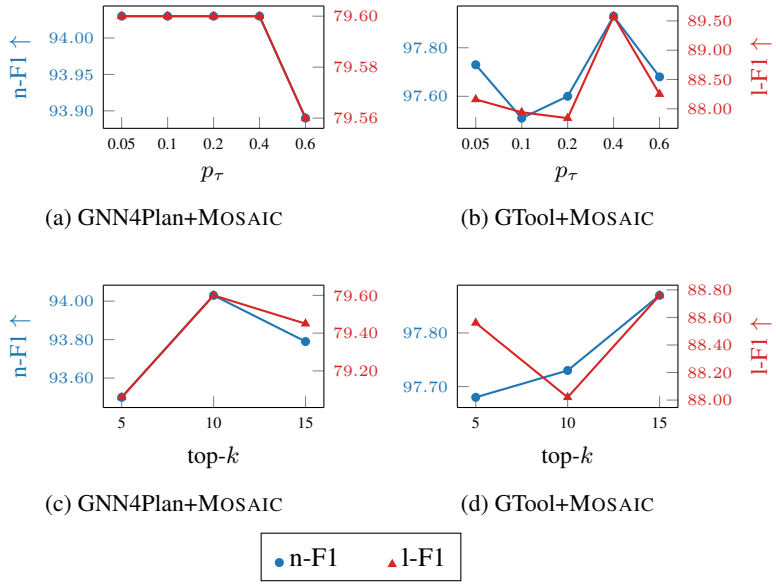
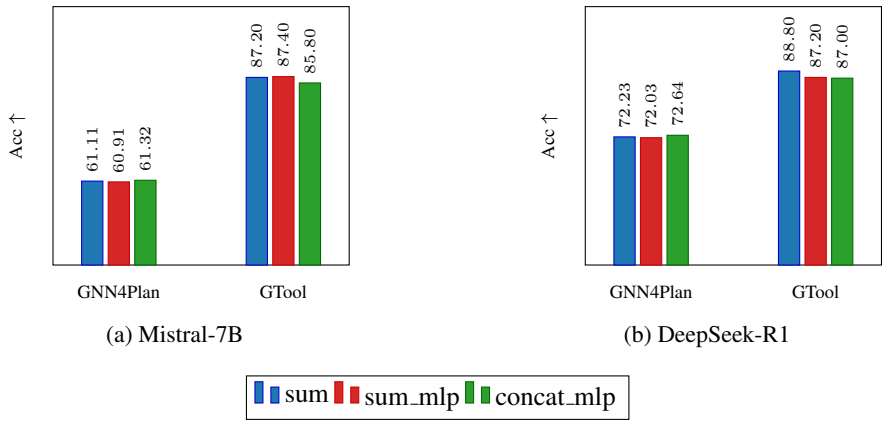


Figure 4: Hyperparameter sensitivity analysis for $p_\tau \in \{0.05, 0.1, 0.2, 0.4, 0.6\}$ and top-$k \in \{5, 10, 15\}$ on the n-F1 (left axis) and l-F1 (right axis) metrics, for GNN4Plan+MOSAIC (left column) and GTool+MOSAIC (right column). The considered setting is DeepSeek-R1 on Daily Life.

the remaining datasets, initializing the model with the pre-trained weights. The only exception is the identity embeddings $\mathbf{U}$: since they are transductive by design (i.e., their size is tied to the tool set of the training dataset), they cannot be reused and are randomly re-initialized, as is common practice. Table 8 reports the results, using Qwen3-14B as the LLM. Overall, performance remains close to that obtained under the standard training setting, suggesting that the representations learned with MOSAIC transfer effectively across datasets. Moreover, GTool+MOSAIC consistently outperforms GNN4Plan+MOSAIC, with the gap being most pronounced when pre-training on HuggingFace. We



(a) Mistral-7B (b) DeepSeek-R1

Figure 5: Hyperparameter sensitivity analysis when exploring different fusion strategies (i.e., sum, sum_mlp, concat_mlp) on Daily Life for Mistral-7B (a) and DeepSeek-R1 (b).

attribute this to the greater diversity of tools in HuggingFace compared to Multimedia, whose tools are more domain-specific.

Table 8: Prediction results when pre-training both baselines+MOSAIC on HuggingFace and Multimedia, and fine-tuning them on different datasets. For these experiments, Qwen3-14B is adopted as the LLM.

Model	Dataset	Pretrained on HuggingFace			Pretrained on Multimedia		
		n-F1 ↑	l-F1 ↑	Acc ↑	n-F1 ↑	l-F1 ↑	Acc ↑
GNN4Plan+MOSAIC	HuggingFace	–	–	–	79.99	61.42	45.40
	Daily Life	96.22	84.04	81.60	96.06	83.49	80.80
	Multimedia	85.21	71.10	56.99	–	–	–
GTool+MOSAIC	HuggingFace	–	–	–	80.15	59.31	45.10
	Daily Life	98.18	88.53	90.10	98.06	88.51	89.60
	Multimedia	88.83	74.64	65.10	–	–	–

F CASE STUDIES

Table 9 provides qualitative examples of the predicted tool sequences on HuggingFace with CodeLlama-13B. Cases (a) and (c) illustrate instances in which MOSAIC corrects errors made by the corresponding backbone planner. In (a), GNN4Plan predicts `Text-to-Image` followed by `Image Editing` instead of the ground-truth `Text Generation` followed by `Text-to-Image`, whereas in (c), GTool selects `Image Classification` instead of `Image-to-Text`. In both cases, the corresponding +MOSAIC variant recovers the ground-truth tool sequence.

Cases (b) and (d) show the converse behavior. In (b), GNN4Plan already predicts the ground-truth tool sequence, while GNN4Plan+MOSAIC selects `Text-to-Image` instead of `Image Editing`. Similarly, in (d), GTool produces the ground-truth sequence, whereas GTool+MOSAIC introduces an additional `Image Segmentation` step.

G COMPUTATIONAL RESOURCES AND CODEBASE

All our experiments were run on a single Linux machine equipped with one 32-core Intel Xeon Platinum 8358 CPU, a RAM with 512 GB, and 4 custom A100 GPU with 64 GB of memory each. For the sake of reproducibility, we make our code available at the following anonymous link: `https://anonymous.4open.science/r/MOSAIC-86BA`.

Table 9: Qualitative comparison of predicted tool sequences on the HuggingFace test split with CodeLlama-13B. Bold tool names highlight deviations from the ground-truth sequence. Cases (a)–(b) compare GNN4Plan with GNN4Plan+MOSAIC, while cases (c)–(d) compare GTool with GTool+MOSAIC.

Method	Tool sequence	Exact
(a) *Classify an image, describe it in a sentence, generate a new image from the sentence, and estimate its depth.*		
Ground Truth	Image Classification → Text Generation → Text-to-Image → Depth Estimation	–
GNN4Plan	Image Classification → **Text-to-Image** → **Image Editing** → Depth Estimation	×
GNN4Plan + MOSAIC	Image Classification → Text Generation → Text-to-Image → Depth Estimation	✓
(b) *Change a red car to blue in an existing image, then segment the edited image into objects.*		
Ground Truth	Image Editing → Image Segmentation	–
GNN4Plan	Image Editing → Image Segmentation	✓
GNN4Plan + MOSAIC	**Text-to-Image** → Image Segmentation	×
(c) *Generate a creative description of an input image and then create a video based on that description.*		
Ground Truth	Image-to-Text → Text Generation → Text-to-Video	–
GTool	**Image Classification** → Text Generation → Text-to-Video	×
GTool + MOSAIC	Image-to-Text → Text Generation → Text-to-Video	✓
(d) *Edit an image according to a text description, identify the objects in the edited image, and generate speech describing the detected objects.*		
Ground Truth	Image Editing → Object Detection → Text-to-Speech	–
GTool	Image Editing → Object Detection → Text-to-Speech	✓
GTool + MOSAIC	Image Editing → **Image Segmentation** → Object Detection → Text-to-Speech	×

H LIMITATIONS

While MOSAIC consistently improves the tested planners, some limitations remain. First, our evaluation covers two graph-based task planners, three benchmark datasets, and a limited set of open-source LLMs; since graph-based task planning is still an emerging area, broader validation on newer planners, larger tool ecosystems, and proprietary LLMs is needed to fully establish its generality. Second, MOSAIC introduces a moderate training overhead, mainly due to the graph contrastive learning module, which may become more relevant as the number of tools grows. Third, the learnable identity embeddings are transductive by design: they must be re-initialized when transferring to a new dataset and cannot directly represent tools unseen during training. Relatedly, the historical tool graph relies on past usage data, so its benefits may be reduced for newly added or rarely used tools. We will devote future directions of the current work to deepen into the outlined limitations and further improve the methodology and its applicability.