TOPOLOGY-CONSISTENT TASK PLANNING OVER CELLULAR WORKFLOW COMPLEXES FOR LLM-BASED AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Task planning for LLM agents requires workflows that satisfy both user intent and complex sub-task dependencies. While existing planners work well for sequential or directed acyclic graph (DAG)-like structures, they struggle with workflow patterns such as verification-correction loops, convergent branch merging, and reusable intermediate states that arise naturally in real-world tool orchestration. We present TopoPlanner, a topology-consistent planning framework that lifts tool dependency graphs into cellular workflow complexes and uses them as topology-aware context for LLM tool planning. TopoPlanner retrieves a request-relevant closed subcomplex through cosheaf-consistent cellular retrieval, performs multi-dimensional structural reasoning over the retrieved topology, and interfaces the resulting cellular representation with the planner LLM for tool-sequence generation. Experiments on four tool-planning benchmarks with topology-guided loop, merge, and loop-merge workflows show consistent improvements over prompt-based and graph-enhanced baselines across different local LLM backbones. Code/data: `https://anonymous.4open.science/r/TopoPlaner-AE05`.

1 INTRODUCTION

Large language model (LLM) agents are becoming a practical interface for tool use, autonomous exploration, and complex decision making (Wang et al. (2024a); Bubeck et al. (2023); Jiang et al. (2025)). Their effectiveness depends critically on task planning: given a natural-language request, the agent must decompose the request into executable sub-tasks, organize their dependencies, and produce a workflow that can be grounded into tools or actions (Schick et al. (2024); Shen et al. (2024a;b)). In realistic agentic settings, this workflow must capture not only individual sub-tasks, but also the dependency structure that makes them executable as a coherent process.

This exposes a structural gap in existing planners. Most LLM planning methods are strongest when the workflow is sequential or approximately directed acyclic graph (DAG)-like, where pairwise task adjacency already captures most of the required structure (Wei et al. (2022); Yao et al. (2024); Shinn et al. (2024)). As illustrated in Fig. 1, real-world tool orchestration is often more complex: it may involve verification-correction loops, convergent branch merging, and reusable intermediate states. Although these structures can be encoded by edges, their joint regional organization remains implicit in pairwise message passing and pooled graph summaries. In these settings, the key dependency patterns arise from how multiple relations jointly form loop, merge, and reuse structures, rather than from isolated edges. This motivates *topology-consistent task planning* beyond pairwise adjacency.

Recent graph-enhanced approaches inject structural bias into agent planning by coupling LLMs with graphs or GNNs (Tang et al. (2024); Chai et al. (2023); Wu et al. (2024); Chen et al. (2025; 2026)). GTool (Chen et al. (2026)) studies graph-enhanced tool planning by encoding tool-dependency graphs with a conventional node-edge GNN and conditioning the planner on one pooled graph token. These methods improve over pure prompting, but they still reason mainly over node-edge structure and therefore leave cyclic or convergent workflow regions implicit. This is restrictive when the critical structure of a workflow lies in loop, merge, and reuse patterns rather than in isolated pairwise relations. Meanwhile, recent evidence suggests that cyclic structure is not incidental: stronger reasoning models tend to exhibit richer loop patterns in their reasoning graphs (Minegishi et al. (2025)).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

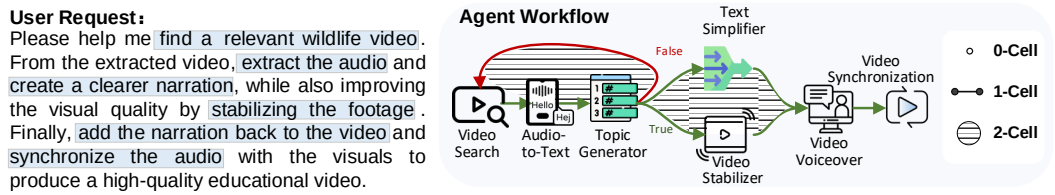

Figure 1: LLM task planning across workflow topologies.

This perspective motivates a unified algebraic-topological formulation of workflow structures. Starting from a task workflow graph, we promote independent cycles induced by non-tree edges into explicit cycle-level cells, resulting in a *cellular workflow complex*. In this complex, boundary relations organize sub-tasks, dependencies, and non-linear workflow regions within one multi-dimensional structure, making region-level dependencies explicit rather than implicit in pairwise paths.

Building on this formulation, we propose **TopoPlanner**, a topology-consistent task planning framework for LLM agents. TopoPlanner lifts the tool dependency graph into a cellular workflow complex, retrieves a request-relevant closed subcomplex through cosheaf-consistent cellular retrieval, performs multi-dimensional structural reasoning over the retrieved topology, and interfaces the resulting cellular representation with the planner LLM for tool-sequence generation. To evaluate this setting, we extend four tool-planning datasets, including the three TaskBench domains (Shen et al. (2024b)) and ToolBench (Chen et al. (2026)), with topology-guided loop, merge, and loop-merge cases that emphasize non-linear workflow structure. Experiments show that TopoPlanner consistently improves planning quality over prior baselines. Our contributions in this work are summarized as follows:

- We formulate loop, merge, and reuse dependencies as explicit regions in cellular workflow complexes, providing a unified multi-dimensional representation of non-linear workflows.
- We introduce TopoPlanner, which lifts a workflow graph into a cellular complex used for cosheaf-consistent retrieval, multi-dimensional reasoning, and topology-guided planning.
- We construct topology-guided extensions of four benchmarks with loop, merge, and loop-merge cases, and show consistent improvements over the compared baselines across different local LLM backbones.

2 PRELIMINARIES

Definition 1. (Cell Complex). Hansen & Ghrist (2019) A **regular cell complex** is a topological space X decomposed into a collection of disjoint subspaces $\{X_\sigma\}_{\sigma \in P_X}$, referred to as *cells*, satisfying the following conditions:

1. For each $x \in X$, there exists an open neighborhood intersecting only finitely many cells.
2. For any pair of cells X_σ, X_τ, the intersection $X_\tau \cap \overline{X_\sigma}$ is nonempty if and only if $X_\tau \subseteq \overline{X_\sigma}$, where $\overline{X_\sigma}$ denotes the topological closure of X_σ.
3. Each cell X_σ is homeomorphic to an open ball in $\mathbb{R}^n$ for some non-negative integer n.
4. *(Regularity)* The closure $\overline{X_\sigma}$ of every cell is homeomorphic to a closed ball in $\mathbb{R}^{n_\sigma}$, with the interior mapped homeomorphically onto X_σ itself.

Definition 2. Given a directed workflow graph $G = (V, E, T)$, let G_u denote its underlying undirected graph, with the original edge directions retained as attributes. For a connected G_u, a chosen spanning tree $\mathcal{T}$ defines a **cellular lifting map** $f_\mathcal{T}$ that constructs a regular cell complex $X = f_\mathcal{T}(G)$: its 0-cells and 1-cells correspond to V and E, and its 2-cells are attached along the fundamental cycles of G_u induced by non-tree edges. Thus, the original directed dependencies remain on the 1-skeleton while cycle-level workflow regions are made explicit.

Definition 3. (Topology-Consistent Task Planning). Let $G = (V, E, T)$ be a text-attributed task graph, and let $X = f_\mathcal{T}(G)$ be its lifted regular cell complex under a cellular lifting map $f_\mathcal{T}$ (Definition 2), where cells encode sub-tasks and their higher-order dependency structures. Given a user instruction, *topology-consistent task planning* selects a request-relevant subcomplex $\widehat{X} \subseteq X$

according to two structural criteria: (i) boundary completeness, i.e., for every cell $\sigma \in \widehat{X}$, its boundary satisfies $\partial\sigma \subseteq \widehat{X}$; and (ii) request-conditioned incidence compatibility between selected cofaces and their boundary cells, measured by the residual in Section 3.2. Retrieval favors lower incidence inconsistency rather than imposing a universal correctness threshold. These criteria define topology consistency of the retrieved structural context: they keep a selected higher-dimensional cell together with its boundary and favor incidence-compatible assignments. They do not, by themselves, constitute a correctness or executability certificate for the tool sequence generated by the planner LLM.

3 The Proposed Model

In this section, we present **TopoPlanner** as a topology-consistent task planning framework over cellular workflow complexes, organized into four modules: *Cellular Workflow Lifting*, *Cosheaf-Consistent Cellular Retrieval*, *Multi-Dimensional Structural Reasoning*, and *Topology-Guided LLM Tool Planning*. Figure 2 summarizes the overall framework. The retrieved cellular representation is interfaced with the planner LLM through a soft graph token.

3.1 Cellular Workflow Lifting

TopoPlanner views the workflow graph as the skeleton of a regular cellular complex and promotes selected cycles into explicit workflow regions. The resulting complex unifies sub-tasks, dependencies, and cycle-level structure, so all later modules operate on one multi-dimensional object.

Given a directed workflow graph $G = (V, E, T)$, we form its underlying undirected graph G_u while retaining each dependency direction as an edge attribute. We regard G_u as the 1-skeleton of a regular cell complex: each vertex is a 0-cell, and each edge is a 1-cell attached to its endpoints. For a connected graph, we fix a spanning tree $\mathcal{T} \subseteq G_u$ (Appendix C) and use the non-tree edges to identify fundamental cycles.

$$\gamma : G_u \to G_u/\mathcal{T}, \tag{1}$$

where $G_u/\mathcal{T}$ denotes the quotient space obtained by collapsing $\mathcal{T}$ into a single point. Each non-tree edge $e = (u, v) \in E \setminus \mathcal{T}$ induces a unique fundamental cycle ℓ_e in G_u, formed by e and the tree path connecting its endpoints. We attach a cycle-level cell along each such loop:

$$\varphi_e : \partial D^2 \to G_u^{(1)}, \qquad \varphi_e(\partial D^2) = \ell_e, \tag{2}$$

where $G^{(1)}$ is the 1-skeleton of the induced CW complex. This lifting step turns cycle-level workflow structure into explicit cells while keeping it attached to the same node-edge scaffold.

Formally, the collection of cycle-level cells constitutes

$$X^{(2)} = \{x_e^2 \simeq D^2 \mid e = (u, v) \in E \setminus \mathcal{T}\}, \tag{3}$$

where each disk D^2 is attached along its boundary $\partial D^2 \cong S^1$ via the loop $\ell_e \subset G_u$.

This lifting operation completes the construction of a 2-dimensional regular cell complex:

$$X = X^{(0)} \cup X^{(1)} \cup X^{(2)}, \tag{4}$$

where the added cycle-level cells complement the original node-edge scaffold by encoding workflow regions within the same cellular complex. In the following, we use $\mathcal{X}$ to denote this lifted cellular workflow complex. We encode node-cell tool descriptions with a pre-trained text encoder, initialize dependency cells from textualized relations, and derive cycle-cell features from their boundaries.

Proposition 1. *(Proof in Appendix A.1.) $G_u/\mathcal{T}$ is homotopy-equivalent to G_u, and γ induces an isomorphism on the first homology group $H_1(G_u; \mathbb{Z})$.*

3.2 Cosheaf-Consistent Cellular Retrieval

After lifting the workflow graph into a cellular complex, TopoPlanner retrieves a compact subcomplex that is relevant to the current request while remaining compatible with the cellular topology. Given a request q, we encode it as

$$\boldsymbol{h}_q = \mathrm{LM}(q) \in \mathbb{R}^d. \tag{5}$$

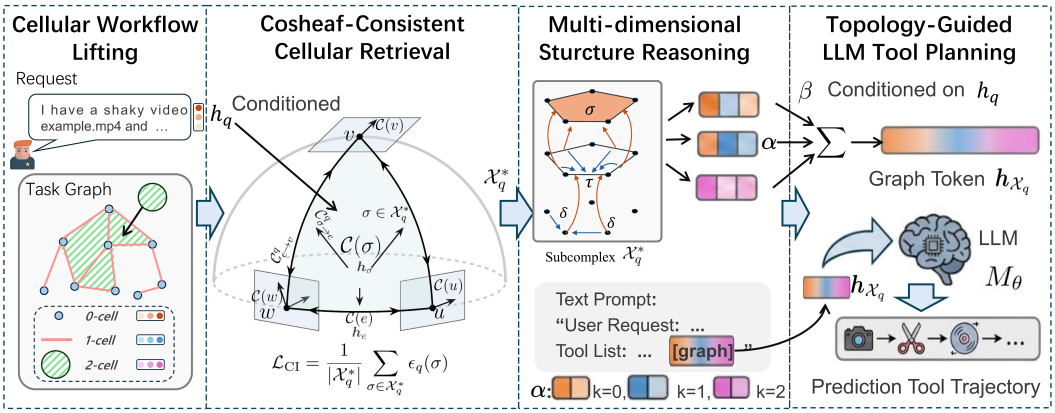

Figure 2: Overview of TopoPlanner.

Rather than appending a request node and independently ranking cells of different dimensions, we view the request as a relevance functional over the lifted cellular workflow complex. Retrieval is then defined as a closed-subcomplex selection problem, while cross-dimensional compatibility is regularized through a cellular cosheaf.

Cellular cosheaf structure. We introduce a request-conditioned cellular cosheaf $\mathcal{C}^q$ over the lifted complex $\mathcal{X}$, following recent sheaf- and cosheaf-based formulations of local consistency on structured domains (Hansen & Ghrist (2019); Barbero et al. (2022); Wei & Wei (2025)). We use the reversed face-poset convention: each cell σ is assigned a stalk vector space $\mathcal{C}(\sigma)$, and an incidence $\tau \prec \sigma$ induces a coface-to-boundary corestriction map:

$$\mathcal{C}^q_{\sigma \to \tau} : \mathcal{C}(\sigma) \to \mathcal{C}(\tau), \tag{6}$$

This follows the homological boundary direction; under the opposite face-poset convention, the same maps are covariant. We implement each map as a q-conditioned sigmoid gate over the request, coface, and boundary-cell embeddings.

Unified subcomplex retrieval. For each cell σ, its request relevance is computed by projecting its cellular embedding into the request embedding space:

$$r_q(\sigma) = \cos\left(\boldsymbol{h}_q, \psi(\boldsymbol{h}_\sigma)\right), \tag{7}$$

where $\psi(\cdot)$ maps cellular embeddings into the request embedding space. For each cell σ, we define its cosine-based cosheaf inconsistency as the average residual over its boundary cells:

$$\epsilon_q(\sigma) = \begin{cases} \dfrac{1}{|\partial\sigma|} \displaystyle\sum_{\tau \prec \sigma} \left(1 - \cos\left(\mathcal{C}^q_{\sigma \to \tau}(\boldsymbol{h}_\sigma), \boldsymbol{h}_\tau\right)\right), & |\partial\sigma| > 0, \\ 0, & |\partial\sigma| = 0, \end{cases} \tag{8}$$

which measures how consistently a selected coface agrees with its boundary in the retrieved complex. We retrieve a compact request-relevant subcomplex by solving:

$$\mathcal{X}^*_q = \underset{\substack{\emptyset \neq \mathcal{Y} \subseteq \mathcal{X}, \\ \partial\sigma \subseteq \mathcal{Y}\ \forall \sigma \in \mathcal{Y}}}{\operatorname{argmax}} \frac{1}{|\mathcal{Y}|} \sum_{\sigma \in \mathcal{Y}} \left(r_q(\sigma) - \epsilon_q(\sigma)\right), \tag{9}$$

The selected set is kept boundary-closed, i.e., for any retained cell $\sigma \in \mathcal{X}^*_q$, its boundary satisfies $\partial\sigma \subseteq \mathcal{X}^*_q$. Thus, the retrieval objective favors request-relevant cells while discouraging cosheaf-inconsistent boundary assignments. Appendix B provides further interpretations of this construction in terms of fixed-depth regional communication, incidence-consistency energy, and the role of closure in preventing consistency by omission. In practice, we cast the selection as a Prize-Collecting Steiner Tree (PCST)-style problem over cells (He et al., 2024), where $r_q(\sigma)$ is the cell prize and $\epsilon_q(\sigma)$ is the cosheaf-induced cost. We then greedily collect high-prize cells with boundary closure. The training regularizer over the retrieved subcomplex is:

$$\mathcal{L}_{\mathrm{CI}} = \frac{1}{|\mathcal{X}^*_q|} \sum_{\sigma \in \mathcal{X}^*_q} \epsilon_q(\sigma). \tag{10}$$

3.3 Multi-Dimensional Structural Reasoning

Given the retrieved request-specific subcomplex $\mathcal{X}_q^*$, TopoPlanner performs multi-dimensional reasoning over its cellular topology. This stage jointly updates representations across dimensions via boundary relations and co-facial interactions, so that downstream planning can use embeddings that reflect both local semantics and global workflow structure.

For each cell σ, the messages are defined as:

$$m_{\mathcal{B}}^{(t+1)}(\sigma) = \mathrm{AGG}_{\tau \in \mathcal{B}(\sigma)} M_{\mathcal{B}} \left(\boldsymbol{h}_\sigma^{(t)}, \boldsymbol{h}_\tau^{(t)} \right), \tag{11}$$

$$m_{\uparrow}^{(t+1)}(\sigma) = \mathrm{AGG}_{\substack{\tau \in \mathcal{N}_\uparrow(\sigma) \\ \delta \in \mathcal{C}(\sigma,\tau)}} M_{\uparrow} \left(\boldsymbol{h}_\sigma^{(t)}, \boldsymbol{h}_\tau^{(t)}, \boldsymbol{h}_\delta^{(t)} \right), \tag{12}$$

where $\mathcal{B}(\sigma) = \partial\sigma$ denotes the set of lower-dimensional cells on the boundary of cell σ, $\mathcal{N}_\uparrow(\sigma)$ denotes the set of same-dimensional cells that co-occur with σ in at least one higher-dimensional cell δ, and $\mathcal{C}(\sigma, \tau)$ denotes the set of such shared higher-dimensional cells.

Finally, both types of structure-aware messages are combined with the current representation of the cell and passed through an update function to obtain the new representation:

$$\boldsymbol{h}_\sigma^{(t+1)} = \mathcal{U} \left(\boldsymbol{h}_\sigma^{(t)},\ m_{\mathcal{B}}^{(t+1)}(\sigma),\ m_{\uparrow}^{(t+1)}(\sigma) \right), \tag{13}$$

where $\mathcal{U}(\cdot)$ is implemented as a nonlinear update function (e.g., ELU + linear). This stage lets each cell encode its local semantics and role in the multi-dimensional workflow.

3.4 Topology-Guided LLM Tool Planning

After multi-dimensional structural reasoning, the retrieved subcomplex $\mathcal{X}_q^*$ contains topology-aware representations of the selected cellular structure. For local planner LLMs, we summarize it as a continuous graph token and inject it together with the textual prompt.

Let $\boldsymbol{h}_\sigma^{(k)}$ denote the updated representation of a k-cell $\sigma \in \mathcal{X}_q^{*(k)}$. TopoPlanner summarizes the retrieved cellular structure by request-aware pooling over both cells and dimensions:

$$\boldsymbol{h}_{\mathcal{X}_q} = \sum_{k=0}^{2} \beta_k \sum_{\sigma \in \mathcal{X}_q^{*(k)}} \alpha_\sigma^{(k)} \boldsymbol{h}_\sigma^{(k)}, \tag{14}$$

where $\alpha_\sigma^{(k)}$ weighs cells per dimension and β_k balances dimensions, both conditioned on the request embedding $\boldsymbol{h}_q$. The vector $\boldsymbol{h}_{\mathcal{X}_q}$ encodes retrieved tools, edges, and higher-dimensional structure.

We then construct the planner prompt from the tool list and user request, with a special graph placeholder `[graph]...[/graph]`. As in soft graph-token conditioning (Tang et al. (2024); Perozzi et al. (2024)), the textual tokens are embedded by the LLM, while the placeholder embedding is replaced by $\boldsymbol{h}_{\mathcal{X}_q}$. The planner LLM M_θ then generates the tool trajectory:

$$\hat{\tau} = M_\theta(w_q), \tag{15}$$

where $\hat{\tau}$ is the predicted tool sequence. The output interface remains identical to standard tool-planning benchmarks: TopoPlanner generates tool names, and workflow edges are constructed and evaluated by the same downstream protocol used in the main experiments.

Training Objective. TopoPlanner is trained to generate the target tool trajectory while reducing cosheaf inconsistency over the retrieved cellular subcomplex. Given the ground-truth tool trajectory τ, the tool-planning negative log-likelihood loss is

$$\mathcal{L}_{\mathrm{TL}} = -\log p_{M_\theta}(\tau \mid w_q). \tag{16}$$

We combine this tool-planning loss with the cosheaf inconsistency loss introduced above:

$$\mathcal{L} = \mathcal{L}_{\mathrm{TL}} + \lambda_{CI} \mathcal{L}_{\mathrm{CI}}, \tag{17}$$

where λ_{CI} controls the strength of cosheaf-inconsistency regularization.

Dataset	Dataset Type					Tool Graph	
	Single	Chain	DAG	Cycle	Total	$\|V\|$	$\|E\|$
HuggingFace	3,067	3,642	837	1,817	9,363	23	225
Multimedia	2,037	2,982	565	1,419	7,003	40	449
Daily Life	1,258	2,787	275	1,098	5,418	40	1,560
ToolBench	101	85	-	412	598	245	366

Table 1: Dataset and tool-graph statistics.

4 Dataset and Data Extension

4.1 Dataset

The proposed framework is evaluated on four tool-planning datasets: HuggingFace, Multimedia, and Daily Life from TaskBench (Shen et al. (2024b)), plus ToolBench (Chen et al. (2026)); their statistics are summarized in Table 1. HuggingFace focuses on AI model tasks in NLP and computer vision, Multimedia covers tasks such as downloading and format conversion, Daily Life contains everyday service operations such as shopping and route navigation, and ToolBench features a substantially larger and more diverse API inventory. The original queries are mainly categorized as Single, Chain, and DAG. To evaluate topology-aware planning, we further introduce a Cycle category covering loop, merge, and loop-merge workflows. We use *Cycle* as an umbrella category for workflow motifs that induce a closed region in the lifted cellular representation: directed feedback loops, convergent merges formed by multiple dependency paths, and loop-merge compositions. A merge is represented as a cycle-level cell because its multiple dependency paths bound a closed region; this structural notion does not imply repeated execution. For motifs containing a directed loop, the loop denotes bounded revision, reuse, or finite transformation rather than unbounded execution; runtime repetition and termination are governed by the downstream executor or stopping policy.

4.2 Agentic Topology-Guided Data Extension

To evaluate topology-aware task planning, we extend the datasets with queries whose task graphs contain higher-dimensional workflow structures, as illustrated in Figure 1. Rather than letting a language model freely invent samples, we first fix a target topology and then generate a natural-language request under that structural constraint. This fixes the dependency structure before generation while allowing variation in request wording.

The extension uses a code-executing construction workflow inspired by code-action agents (Wang et al. (2024b)): a human designer specifies the structural constraints, executable scripts instantiate candidate workflows, and an LLM generates each request conditioned on the selected tools and fixed topology. Each accepted workflow is represented by task nodes, task links, task steps, and its generated user request. The resulting samples are validated, merged into the original dataset format, and split with grouped train-test partitioning to reduce structural leakage. All baselines and TopoPlanner are evaluated on this same extended dataset and split; implementation and validation details are provided in Appendix D, especially Sections D.1 and D.3. Appendix G.1 further reports a motif-leakage audit, held-out-topology evaluation, and results on original non-augmented workflows.

5 Experiments

5.1 Experimental Setup

Evaluation Metrics. Evaluation metrics are adopted from prior work (Shen et al. (2024b;a)): Node F1-Score (*n-F1*) and Link F1-Score (*l-F1*) to measure the accuracy of task predictions and inter-task dependencies, respectively. Additionally, Accuracy (*ACC*) evaluates the overall success rate.

LLMs. Our main experiments use open-source large language models, including CodeLlama-7B/13B-Instruct-hf (Roziere et al. (2023)) and Vicuna-7B/13B-v1.5 (Zheng et al. (2023)). We additionally evaluate TopoPlanner with API-based LLM backbones, including Qwen3-235B-A22B-Instruct (Yang et al. (2025)) and DeepSeek-V3-671B (DeepSeek-AI (2024)), in Appendix E.

LLM	Method	HuggingFace			Multimedia			Daily Life			ToolBench		
		N ↑	*L* ↑	*Acc* ↑	*N* ↑	*L* ↑	*Acc* ↑	*N* ↑	*L* ↑	*Acc* ↑	*N* ↑	*L* ↑	*Acc* ↑
Vicuna-7B	Direct	43.66	11.91	6.70	34.78	0.45	2.05	42.82	17.05	10.11	36.83	17.89	13.24
	TalkGraph	57.21	26.18	8.37	43.25	13.85	3.37	58.97	29.74	20.23	20.13	12.79	5.88
	Plan-over-Graph	57.36	11.71	9.74	43.18	0.59	3.66	60.41	16.27	22.99	36.83	21.46	13.24
	GNN4TaskPlan	57.17	27.01	8.68	46.90	17.29	4.83	58.97	29.74	20.23	24.34	18.24	11.76
	USPilot	60.67	32.31	12.79	46.45	17.20	4.83	52.10	23.91	14.71	21.09	11.18	5.88
	GTool	52.33	19.73	8.09	47.17	19.64	7.07	75.10	52.40	42.96	45.53	28.27	30.83
	TopoPlanner	**82.23**	**55.72**	**42.35**	**86.80**	**66.24**	**54.76**	**97.27**	**78.53**	**86.25**	**55.11**	**35.96**	**41.67**
CodeLlama-7B	Direct	56.19	28.88	9.12	49.69	5.27	11.23	71.58	46.09	36.19	45.63	22.60	11.67
	TalkGraph	61.85	32.74	9.12	52.58	29.19	13.80	69.38	44.32	32.99	44.75	29.46	21.67
	Plan-over-Graph	61.23	24.37	9.26	56.55	4.97	16.36	73.48	45.50	38.86	45.63	24.06	11.67
	GNN4TaskPlan	61.55	32.92	9.41	57.01	34.94	18.80	71.04	45.64	35.12	41.98	26.99	20.83
	USPilot	65.06	38.49	13.82	58.59	36.75	21.61	66.59	39.76	26.17	31.02	13.56	5.00
	GTool	68.35	39.28	23.53	69.23	46.25	30.37	79.56	58.29	52.58	55.57	38.76	36.67
	TopoPlanner	**83.43**	**59.62**	**47.35**	**89.08**	**69.59**	**62.20**	**97.43**	**77.95**	**87.29**	**59.51**	**43.88**	**40.00**
Vicuna-13B	Direct	50.94	23.16	7.00	43.10	8.98	4.76	57.93	32.68	20.28	42.31	18.42	9.65
	TalkGraph	59.47	29.31	7.88	53.28	27.43	12.29	65.56	38.00	23.95	41.53	28.94	20.18
	Plan-over-Graph	60.90	22.22	10.50	55.58	9.32	11.74	69.64	31.93	30.42	42.31	19.67	9.65
	GNN4TaskPlan	59.90	30.95	9.88	58.11	31.61	15.06	65.56	38.00	23.95	42.82	29.14	21.93
	USPilot	63.53	37.30	14.00	57.42	31.82	14.62	60.93	32.77	19.23	31.72	13.10	5.26
	GTool	68.98	41.03	26.03	64.87	39.53	21.71	81.49	62.14	59.11	45.07	28.52	22.50
	TopoPlanner	**83.49**	**58.32**	**44.26**	**89.39**	**69.20**	**61.83**	**98.09**	**78.74**	**89.00**	**49.77**	**31.06**	**27.66**
CodeLlama-13B	Direct	59.70	31.12	11.47	60.82	0.34	18.54	75.78	50.92	44.41	48.00	24.91	15.00
	TalkGraph	63.22	36.20	8.68	60.61	36.68	19.51	73.39	49.03	38.21	43.42	28.74	22.50
	Plan-over-Graph	64.12	27.87	11.62	63.76	0.42	20.12	76.17	49.21	44.23	48.00	25.59	15.00
	GNN4TaskPlan	64.51	37.88	10.59	64.20	40.56	23.41	73.39	49.03	38.21	40.91	27.31	21.67
	USPilot	67.89	42.99	15.59	64.75	41.18	23.66	68.26	42.72	30.81	32.03	13.00	8.33
	GTool	71.59	47.24	33.68	73.80	52.55	39.63	82.86	63.28	60.65	57.89	38.74	35.83
	TopoPlanner	**83.07**	**58.21**	**45.29**	**90.25**	**70.93**	**67.07**	**97.36**	**77.08**	**86.77**	**66.48**	**49.06**	**49.17**

Table 2: Results on four benchmarks with local LLM backbones; best values are bold and second-best values are underlined.

Baselines. Direct: Queries a pre-trained LLM directly to generate task plans without using any graph structure; **TalkGraph** (Fatemi et al. (2024)): Encodes graph-structured data as text for black-box LLMs, enabling reasoning over nodes and edges through prompt engineering; **GNN4TaskPlan** (Wu et al. (2024)): Leverages LLMs to interpret user requests and employs GraphSAGE with ranking-loss training to retrieve relevant sub-tasks; **Plan-over-Graph** (Zhang et al. (2025)): Leverages an LLM to extract task graphs and applies topological sorting over LLM-predicted task dependencies to generate execution plans; **USPilot** (Chen et al. (2025)): Combines a large language model with an LLM-enhanced GNN, using dynamic routing and adapters for task planning; **GTool** (Chen et al. (2026)): A strong graph-enhanced tool-planning baseline that encodes the tool-dependency graph with a conventional node-edge GNN, pools the graph into a single soft graph token for the planner LLM, and adds an auxiliary edge-existence objective.

Implementation Details. All local-model training and evaluation experiments are conducted on four NVIDIA H800-80G GPUs. We use `e5-large-v2` (Wang et al. (2022)) to initialize tool, request, and cellular features. TopoPlanner is configured with a hidden dimension of 1024. In the main local-LLM setting, the planner backbone is frozen and only the cellular modules and graph-token interface are trained with AdamW (Kingma & Ba (2014)) ($lr = 1 \times 10^{-3}$, weight decay 0.05, batch size 1) for at most 12 epochs with early stopping (patience 2). For the 7B LoRA continuation experiments, we use a smaller learning rate of 1×10^{-5}. The maximum input and generation lengths are 1024 and 128, respectively. We sweep the coefficient of $\mathcal{L}_{CI}$ by varying $\lambda_{CI} \in \{0, 0.005, 0.01, 0.05, 0.1\}$ across the three TaskBench domains.

5.2 Experimental Results

Overall Performance. Table 2 compares methods under open-source local LLMs:

TopoPlanner consistently improves both tool selection and workflow construction. It outperforms the compared prompting and graph-enhanced baselines across the evaluated datasets and local LLM backbones. More importantly, the improvement is not confined to n-F1: the gains over the strongest baseline are often more pronounced in l-F1 and ACC, which require recovering dependencies and the complete plan rather than only identifying relevant tools. For example, on Multimedia with Vicuna-7B, TopoPlanner improves over GTool from 47.17/19.64/7.07 to 86.80/66.24/54.76 in n-F1/l-F1/ACC; on the larger ToolBench graph with CodeLlama-13B, the corresponding scores increase from 57.89/38.74/35.83 to 66.48/49.06/49.17.

LLM	Method	HuggingFace			Multimedia			Daily Life		
		N ↑	*L* ↑	*Acc* ↑	*N* ↑	*L* ↑	*Acc* ↑	*N* ↑	*L* ↑	*Acc* ↑
Vicuna-7B	-w/o 2-cell	75.58	46.61	30.88	73.29	50.13	34.15	81.25	62.42	59.28
	-w/o Retrieval	76.07	47.88	32.21	77.97	55.11	35.37	82.99	65.65	64.26
	-w/o lci	80.69	53.31	38.68	81.47	58.29	40.98	93.13	71.44	69.76
	TopoPlanner	**82.23**	**55.72**	**42.35**	**86.80**	**66.24**	**54.76**	**97.27**	**78.53**	**86.25**
CodeLlama-7B	-w/o 2-cell	78.95	51.87	35.29	75.90	56.35	43.54	82.86	64.35	63.75
	-w/o Retrieval	79.75	56.96	35.59	83.89	62.92	49.27	86.85	67.05	53.66
	-w/o lci	80.8	53.83	38.82	87.12	67.61	53.54	95.41	76.2	75.95
	TopoPlanner	**83.43**	**59.62**	**47.35**	**89.08**	**69.59**	**62.20**	**97.43**	**77.95**	**87.29**

Table 3: TopoPlanner ablations on three datasets with 7B local LLMs.

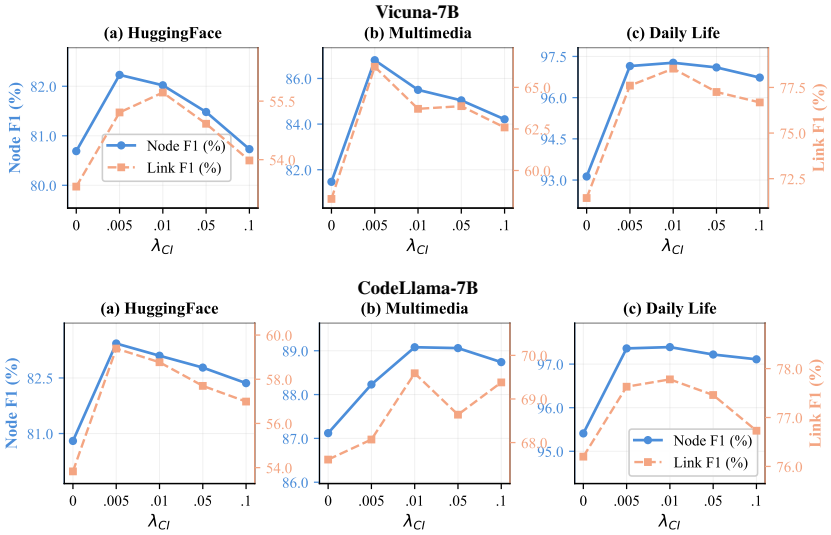

Figure 3: λ_{CI} sensitivity across datasets and 7B backbones.

These results align with the structural gap motivating TopoPlanner. The strongest comparator in most settings, GTool, already conditions the same planner backbones on a pooled node–edge graph representation. TopoPlanner instead makes loop, merge, and loop-merge regions explicit as 2-cells, retrieves their request-relevant boundary-closed subcomplex, and propagates information across dimensions before conditioning the planner. Its additional gains over GTool are therefore consistent with the value of preserving the regional structures highlighted in Figure 1, rather than flattening them into pairwise relations. The advantage persists across the 7B and 13B Vicuna and CodeLlama families, suggesting that this structural interface complements backbone capacity.

To isolate the sources of these gains, the ablations below evaluate 2-cell lifting, request-specific cellular retrieval, and its consistency objective. Appendix G.3 tests boundary closure and an alternative hypergraph representation, while Appendix G.4 directly evaluates merge/reuse recall and dependency order. Together, these diagnostics test the proposed regional representation and boundary-complete retrieval.

Ablation Study. Table 3 compares the full model with `-w/o 2-cell`, which uses the edge-only 1-skeleton; `-w/o Retrieval`, which removes request-specific closed-subcomplex retrieval; and `-w/o lci`, which removes the retrieval-consistency supervision.

- **Explicit higher-order lifting provides useful regional context.** Replacing the cellular workflow complex with its 1-skeleton causes a clear performance drop across both backbones and all three domains. Under the evaluated architectures, explicit 2-cell regions provide context for loop, merge, and loop-merge patterns beyond an edge-only model.
- **Cellular retrieval and its consistency signal are complementary.** Relative to removing $\mathcal{L}_{CI}$, removing retrieval causes a larger drop in 17 of the 18 backbone–domain–metric comparisons; the sole exception is CodeLlama-7B/HuggingFace l-F1. The full model exceeds both variants in all 18 comparisons.

Sensitivity to λ_{CI}. Figure 3 probes the consistency term in cellular retrieval, which is designed to keep the request-relevant subcomplex boundary-compatible rather than selecting cells independently. All six panels peak at a positive λ_{CI}, usually 0.005 or 0.01, whereas larger weights generally

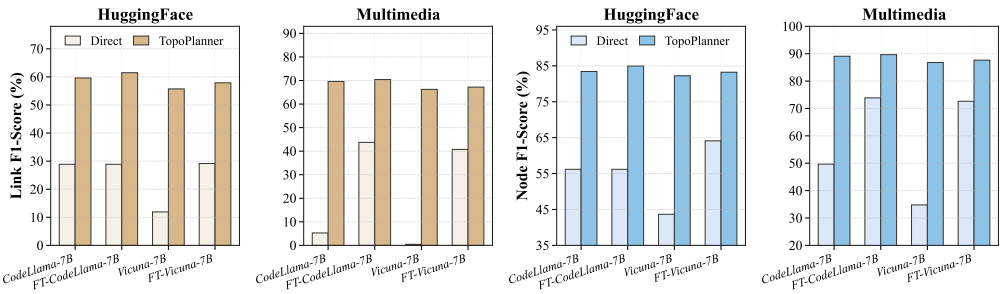

Figure 4: One-epoch LoRA continuation results.

plateau or reduce performance. This pattern supports the intended balance of TopoPlanner: regional consistency complements semantic relevance but should not dominate it.

Effect of fine-tuning. Figure 4 tests whether backbone adaptation alone closes the structural gap highlighted in Figure 1. One-epoch LoRA improves Direct, especially on Multimedia, but the corresponding TopoPlanner result remains higher before and after adaptation in all four panels. The result is consistent with the two signals being complementary: fine-tuning strengthens the planner, while cellular conditioning preserves explicit information about non-linear dependency regions. This conclusion is limited to the tested one-epoch setting. Qualitative analyses of loop, merge, and loop-merge workflows are provided in Appendix H.

6 Related Work

Task planning in LLM-based agents aims to decompose natural-language requests into executable sub-tasks. Most existing methods are prompt-based, including task decomposition (Wei et al. (2022)), multi-plan search (Yao et al. (2024); Besta et al. (2024)), symbolic planning (Liu et al. (2023)), and reflection-based refinement (Shinn et al. (2024)). These approaches have broadened the planning ability of LLMs, but they usually assume sequential, tree-like, or DAG-like workflows, which makes it difficult to capture richer dependency patterns such as loops, merges, and feedback structures.

Another line of work studies graph reasoning with LLMs. GPT4Graph and NLGraph evaluate how pretrained LLMs process graph problems in natural language (Guo et al. (2023); Wang et al. (2023)), while TalkLikeAGraph and Plan-over-Graph encode graphs as text and rely on the LLM to reason over dependencies or derive execution orders (Fatemi et al. (2024); Zhang et al. (2025)). These results suggest that LLMs can exploit graph-structured context, but text-only reasoning still lacks explicit structural bias on more complex topological relations.

To address this limitation, several hybrid LLM+GNN methods inject graph representations into LLMs. GraphGPT, GraphLLM, and GraphToken incorporate learned graph embeddings or graph tokens into the language-model input on graph-centric tasks (Tang et al. (2024); Chai et al. (2023); Perozzi et al. (2024)). In tool planning, GNN4TaskPlan (Wu et al. (2024)) uses an LLM to decompose user goals and a GNN matcher to retrieve relevant sub-tasks, while USPilot (Chen et al. (2025)) combines an LLM with an LLM-enhanced GNN, dynamic routing, and adapters. GTool (Chen et al. (2026)) encodes the tool-dependency graph with a node-edge GNN, pools it into a single graph token for the planner LLM, and adds an auxiliary edge-existence objective. Despite their strength, these methods still reason mainly over node-edge structure and do not explicitly elevate higher-order workflow regions into the planning representation. Appendix I reviews topological deep learning.

7 Conclusion

In this work, we present **TopoPlanner**, a topology-consistent task planning framework that makes loop, merge, and loop-merge regions explicit by lifting tool dependency graphs into cellular workflow complexes. The lifted structure supports request-specific retrieval, multi-dimensional reasoning, and topology-guided tool-sequence generation. We also construct topology-guided extensions of four tool-planning benchmarks. Across the evaluated datasets and local LLM backbones, TopoPlanner consistently outperforms the compared baselines; the ablations and structure-specific diagnostics further support explicit regional modeling as an effective inductive bias within the evaluated scope.

AI USE STATEMENT

Generative AI tools assisted with language editing and implementation of the topology-guided data-construction pipeline; an LLM also generated requests under fixed workflow constraints, as documented in Appendix D.1. All outputs and scientific claims were reviewed and verified by the authors.

REFERENCES

S. Barbarossa and S. Sardellitti. Topological signal processing over simplicial complexes. *IEEE Transactions on Signal Processing*, 2020.

Federico Barbero, Cristian Bodnar, Haitz Sáez de Ocáriz Borde, Michael Bronstein, Petar Veličković, and Pietro Liò. Sheaf neural networks with connection laplacians, 2022.

Claudio Battiloro, Zhiyang Wang, Hans Riess, Paolo Di Lorenzo, and Alejandro Ribeiro. Tangent bundle convolutional learning: from manifolds to cellular sheaves and back. *IEEE Transactions on Signal Processing*, 72:1892–1909, 2024.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17682–17690, 2024.

Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yuguang Wang, Pietro Liò, Guido F. Montúfar, and Michael M. Bronstein. Weisfeiler and lehman go cellular: CW networks. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 2625–2640, 2021a. URL https://proceedings.neurips.cc/paper/2021/hash/157792e4abb490f99dbd738483e0d2d4-Abstract.html.

Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F Montufar, Pietro Lio, and Michael Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks. In *International conference on machine learning*, pp. 1026–1037. PMLR, 2021b.

Cristian Bodnar, Francesco Di Giovanni, Benjamin Paul Chamberlain, Pietro Lio, and Michael M. Bronstein. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in GNNs. In *Advances in Neural Information Processing Systems*, 2022.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiao Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. GraphLLM: Boosting graph reasoning ability of large language model. *arXiv preprint arXiv:2310.05845*, 2023.

Mingcong Chen, Siqi Fan, Guanglin Cao, Yun-hui Liu, and Hongbin Liu. Uspilot: An embodied robotic assistant ultrasound system with a large language model enhanced graph planner. *IEEE Robotics Autom. Lett.*, 10(10):10027–10034, 2025. doi: 10.1109/LRA.2025.3598625. URL https://doi.org/10.1109/LRA.2025.3598625.

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

DeepSeek-AI. Deepseek-v3 technical report. *CoRR*, abs/2412.19437, 2024. doi: 10.48550/ARXIV.2412.19437. URL https://doi.org/10.48550/arXiv.2412.19437.

S. Ebli, M. Defferrard, and G. Spreemann. Simplicial neural networks. In *Advances in Neural Information Processing Systems Workshop on Topological Data Analysis and Beyond*, 2020.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models. In *The Twelfth International Conference on Learning Representations, ICLR*, 2024. URL https://openreview.net/forum?id=IuXR1CCrSi.

Lorenzo Giusti, Claudio Battiloro, Lucia Testa, Paolo Di Lorenzo, Stefania Sardellitti, and Sergio Barbarossa. Cell attention networks. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2023.

Christopher Wei Jin Goh, Cristian Bodnar, and Pietro Lio. Simplicial attention networks. In *ICLR 2022 Workshop on Geometrical and Topological Representation Learning*.

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

M. Hajij, G. Zamzmi, T. Papamarkou, N. Miolane, A. Guzmán-Sáenz, and K. N. Ramamurthy. Higher-order attention networks, 2022.

Mustafa Hajij, Kyle Istvan, and Ghada Zamzmi. Cell complex neural networks. In *Advances in Neural Information Processing Systems Workshop on TDA & Beyond*, 2020.

Mustafa Hajij, Ghada Zamzmi, Theodore Papamarkou, Nina Miolane, Aldo Guzmán-Sáenz, Karthikeyan Natesan Ramamurthy, Tolga Birdal, Tamal K. Dey, Soham Mukherjee, Shreyas N. Samaga, Neal Livesay, Robin Walters, Paul Rosen, and Michael T. Schaub. Topological deep learning: Going beyond graph data, 2023.

J. Hansen and R. Ghrist. Toward a spectral theory of cellular sheaves. *Journal of Applied and Computational Topology*, 3:315–358, 2019.

Jakob Hansen and Thomas Gebhart. Sheaf neural networks, 2020.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *Advances in Neural Information Processing Systems*, 37:132876–132907, 2024.

Jinhao Jiang, Kun Zhou, Xin Zhao, Yang Song, Chen Zhu, Hengshu Zhu, and Ji-Rong Wen. Kg-agent: An efficient autonomous agent framework for complex reasoning over knowledge graph. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025*, pp. 9505–9523, 2025.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

Gouki Minegishi, Hiroki Furuta, Takeshi Kojima, Yusuke Iwasawa, and Yutaka Matsuo. Topology of reasoning: Understanding large reasoning models through reasoning graph properties. *CoRR*, abs/2506.05744, 2025. doi: 10.48550/ARXIV.2506.05744. URL https://doi.org/10.48550/arXiv.2506.05744.

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei-ge Chen, Olga Vrousgos, Corby Rosset, Fillipe Silva, Hamed Khanpour, Yash Lara, and Ahmed Awadallah. Agentinstruct: Toward generative teaching with agentic flows. *arXiv preprint arXiv:2407.03502*, 2024.

Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. Let your graph do the talking: Encoding structured data for llms, 2024.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

S. Sardellitti, S. Barbarossa, and L. Testa. Topological signal processing over cell complexes. In *Asilomar Conference on Signals, Systems, and Computers*, 2021.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2024.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36, 2024a.

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. *Advances in Neural Information Processing Systems*, 37:4540–4574, 2024b.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

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 *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 491–500, 2024.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. Can language models solve graph problems in natural language? *Advances in Neural Information Processing Systems*, 36, 2023.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):1–26, 2024a.

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. *ArXiv*, abs/2212.03533, 2022.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232, 2024b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Xiaoqi Wei and Guo-Wei Wei. Persistent sheaf laplacians. *Foundations of data science (Springfield, Mo.)*, 7(2):446, 2025.

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 Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/098d1bd3eb6156a4c2f834563cdcf617-Abstract-Conference.html.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Maosheng Yang and Elvin Isufi. Convolutional learning on simplicial complexes, 2023.

Ruochen Yang, Frederic Sala, and Paul Bogdan. Efficient representation learning for higher-order data with simplicial complexes. In Bastian Rieck and Razvan Pascanu (eds.), *Proceedings of the First Learning on Graphs Conference*, volume 198 of *Proceedings of Machine Learning Research*, pp. 13:1–13:21. PMLR, 09–12 Dec 2022. URL https://proceedings.mlr.press/v198/yang22a.html.

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. *Advances in Neural Information Processing Systems*, 36, 2024.

Shiqi Zhang, Xinbei Ma, Zouying Cao, Zhuosheng Zhang, and Hai Zhao. Plan-over-graph: Towards parallelable llm agent schedule. *arXiv preprint arXiv:2502.14563*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

A PROOFS

A.1 PROOFS REGARDING PROPOSITION 1

Proof. **Contractibility of the spanning tree.** For the connected underlying undirected graph G_u, a spanning tree $\mathcal{T}$ is connected and acyclic, hence contractible. A contractible subspace can be continuously shrunk to a point.

Collapsing a contractible subspace. Consider the quotient map $\gamma : G_u \to G_u/\mathcal{T}$ that identifies all points of $\mathcal{T}$ with a single vertex v_0. Because $\mathcal{T}$ is a contractible subcomplex of the graph, the quotient map is a homotopy equivalence. Hence there exists a homotopy inverse $s : G_u/\mathcal{T} \to G_u$ such that $s \circ \gamma \simeq \mathrm{id}_{G_u}$ and $\gamma \circ s \simeq \mathrm{id}_{G_u/\mathcal{T}}$. Thus, $G_u \simeq G_u/\mathcal{T}$.

Induced isomorphism on first homology. Homotopy-equivalent spaces have isomorphic homology groups. Hence the induced map $\gamma_* : H_1(G_u; \mathbb{Z}) \to H_1(G_u/\mathcal{T}; \mathbb{Z})$ is an isomorphism.

Intuition in graph terms. The spanning tree $\mathcal{T}$ contains no cycles, so collapsing it does not remove or merge any cycles in G_u. Each fundamental cycle in G_u (formed by a non-tree edge and the unique tree path connecting its endpoints) becomes a loop based at the collapsed tree vertex. Thus, collapsing $\mathcal{T}$ preserves the independent cycles measured by $H_1(G_u)$. □

A.2 PROOF OF THE FUNDAMENTAL-CYCLE BASIS CLAIM

Proof. Let $G_u = (V, E)$ be a finite connected undirected graph underlying a directed workflow graph, and let $\mathcal{T} \subset G_u$ be a spanning tree. Edge directions remain attributes for planning but do not alter the cycle-space argument.

Existence and uniqueness of fundamental cycles. For each non-tree edge $e = (u, v) \in E \setminus \mathcal{T}$, there exists a unique simple path $P_{\mathcal{T}}(u, v)$ in $\mathcal{T}$ connecting u and v, by acyclicity of $\mathcal{T}$. Concatenating e with $P_{\mathcal{T}}(u, v)$ defines a unique simple cycle

$$C_e = e \cup P_{\mathcal{T}}(u, v) \subset G.$$

Under the quotient map $\gamma : G_u \to G_u/\mathcal{T}$, the tree path $P_{\mathcal{T}}(u, v)$ is mapped to that point, so C_e becomes a nontrivial loop in $G_u/\mathcal{T}$.

Spanning and independence in homology. The cyclomatic number of G_u is $\beta_1(G_u) = |E| - |V| + 1$, which equals the number of non-tree edges. Thus there are exactly $\beta_1(G)$ fundamental cycles.

Any cycle in G_u is a linear combination of these fundamental cycles: whenever traversal encounters a non-tree edge e, the corresponding C_e eliminates the associated tree segment.

These cycles are independent in $H_1(G_u)$: under γ, each fundamental cycle maps to a distinct loop in $G_u/\mathcal{T}$, and loops around different edges of a wedge of circles are homologically independent.

Hence the set

$$\{[C_e] \mid e \in E \setminus \mathcal{T}\}$$

forms a basis of $H_1(G_u)$ under the stated connected-graph assumption.

Topological summary. Different choices of spanning tree yield different fundamental-cycle edge sets, but their homology classes always span $H_1(G_u)$. These loops therefore capture all independent cycle-space relations of the underlying graph, while the retained edge directions preserve the workflow semantics used by the planner. □

B THEORETICAL ROLE OF THE CELLULAR STRUCTURE

The homotopy and cycle-basis results above justify the lifting construction. We next clarify what the lifted cells contribute to representation learning and retrieval. These statements characterize the structural inductive bias of TopoPlanner; boundary closure alone does not certify runtime correctness.

Proposition 2 (Fixed-depth regional communication)**.** *Let R be a workflow region containing two boundary elements whose graph distance exceeds L in the pairwise 1-skeleton. In an L-layer local message-passing network, the representation of either element cannot depend on the other. If R is*

represented by a shared 2-cell, every boundary element is incident to that cell, and information can be exchanged across the region through a boundary-to-coface update followed by a coface-to-boundary update, regardless of boundary length.

Proof. Local pairwise message passing expands a receptive field by at most one graph hop per layer. Hence two elements at distance greater than L cannot influence one another after L layers. After lifting R to a 2-cell, each boundary element is connected to the shared coface through an incidence relation. A boundary-to-coface update followed by a coface-to-boundary update therefore communicates across the region in constant depth. This is a representational shortcut rather than a correctness guarantee; a suitably constructed hypergraph can provide a related shortcut, while the cellular representation additionally retains graded boundary relations. □

Proposition 3 (Incidence-consistency energy)**.** *For a boundary-closed candidate $\mathcal{Y}$, let $H(\mathcal{Y})$ be its directed Hasse incidence graph and let the normalized coface-to-boundary residual be*

$$(\delta_q \boldsymbol{h})_{\sigma,\tau} = \widehat{\mathcal{C}^q_{\sigma\to\tau}(\boldsymbol{h}_\sigma)} - \widehat{\boldsymbol{h}_\tau}, \qquad \tau \prec \sigma. \tag{18}$$

Then the cosine inconsistency in Eq. (8) is a weighted quadratic incidence energy because $1 - \cos(\boldsymbol{a}, \boldsymbol{b}) = \frac{1}{2}\|\widehat{\boldsymbol{a}} - \widehat{\boldsymbol{b}}\|_2^2$. In the quadratic implementation, the same construction has the form

$$E_q(\mathcal{Y}, \boldsymbol{h}) = \|W_q^{1/2}\delta_q \boldsymbol{h}\|_2^2 = \boldsymbol{h}^\top L_q \boldsymbol{h}, \qquad L_q = \delta_q^\top W_q \delta_q. \tag{19}$$

Consequently, low energy indicates proximity to assignments that agree across the selected incidences.

Proposition 4 (Closure prevents consistency by omission)**.** *Let S be a selected set of cells, $\mathrm{cl}(S)$ its boundary closure, and $I(S)$ the set of coface–boundary incidences whose two cells are both contained in S. With non-negative incidence weights and per-incidence residuals $e_q(\sigma, \tau) \geq 0$,*

$$E_q(\mathrm{cl}(S), \boldsymbol{h}) = E_q(S, \boldsymbol{h}) + \sum_{(\sigma,\tau)\in I(\mathrm{cl}(S))\setminus I(S)} w_{\sigma\tau} e_q(\sigma, \tau) \geq E_q(S, \boldsymbol{h}). \tag{20}$$

Thus closure forces the objective to evaluate newly included incidences between retained cofaces and their required boundary cells, rather than treating the added cells as an independent subcomplex. Moreover, let $\mathcal{Y}^\star$ be a reachable closed reference workflow and $\mathcal{Y}$ a competing closed workflow. If the competitor's relevance advantage is at most ρ while its excess incidence inconsistency is at least μ, then for $J_q(\mathcal{Y}) = \bar{r}_q(\mathcal{Y}) - \bar{\epsilon}_q(\mathcal{Y})$,

$$J_q(\mathcal{Y}^\star) - J_q(\mathcal{Y}) \geq \mu - \rho. \tag{21}$$

Hence $\mu > \rho$ is a sufficient conditional margin for preferring the incidence-compatible workflow. This criterion explains consistency and closure only under its stated assumptions.

C Algorithm Details

Based on the theoretical proofs provided in Section A.2, we detail the computational procedure for lifting the task graph into a cellular complex. Algorithm 1 outlines the process of extracting the fundamental cycle basis using a spanning tree approach. This method guarantees a linear number of cycles relative to the cyclomatic number β_1, ensuring the scalability discussed in Section J.

Algorithm 1 Fundamental Cycle Basis Construction via Spanning Tree

Require: Connected underlying undirected graph $G_u = (V, E)$; optional maximum length K_{max}
Ensure: Fundamental cycle set $\mathcal{C}$ (a complete basis when K_{max} is None)
Initialize: Disjoint Set Union (DSU) structure for cycle detection.
$\mathcal{T} \leftarrow \emptyset$ {Initialize Spanning Tree edges}
$E_{non} \leftarrow \emptyset$ {Initialize Non-Tree edges}
$\mathcal{C} \leftarrow \emptyset$ {Initialize Cycle Basis}
{**Phase 1: Build Spanning Tree (Kruskal-like logic)**}
for each edge $(u, v) \in E$ **do**
  if FINDSET$(u) \neq$ FINDSET(v) **then**
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{(u, v)\}$
    UNIONSET(u, v)
  else
    $E_{non} \leftarrow E_{non} \cup \{(u, v)\}$
  end if
end for
{**Phase 2: Extract Fundamental Cycles**}
Construct adjacency list for $\mathcal{T}$ to enable path traversal.
for each non-tree edge $e = (u, v) \in E_{non}$ **do**
  {Find the unique path in $\mathcal{T}$ connecting u and v}
  $P_{\mathcal{T}}(u, v) \leftarrow \text{BFS}(start = u, target = v, graph = \mathcal{T})$
  {Form the cycle by combining the non-tree edge and the tree path}
  $C_e \leftarrow P_{\mathcal{T}}(u, v) \cup \{e\}$
  if K_{max} is None **or** $|C_e| \leq K_{max}$ **then**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_e\}$
  end if
end for
return $\mathcal{C}$

The homology and basis claims use the untruncated setting $K_{max} = \text{None}$. With a finite K_{max}, the algorithm returns only the retained length-filtered subset, which is not claimed to span the full cycle space.

D Data Construction and Prompt Details

To ensure reproducibility and facilitate future research, we describe the topology-guided data construction workflow and provide the prompt template used by TopoPlanner during inference. The data extension is not produced by a one-shot generation prompt. Instead, it is generated by executable scripts developed through an agentic code-writing workflow, following the idea of code-action agents (Wang et al. (2024b)) and agentic data construction (Mitra et al. (2024)).

D.1 Agentic Topology-Guided Construction Workflow

Given human-specified workflow-topology requirements, Codex is used as a code-executing agent to implement and iterate the data construction pipeline. Algorithm 2 summarizes the process: it mines valid motifs from the original tool graph, instantiates samples, generates requests under fixed topology constraints, validates them, and creates leakage-controlled splits.

Structure constraints. The construction code treats the original tool graph as the source of structural validity. For each dataset, it loads the tool inventory and directed dependency links, builds adjacency lists and an edge lookup table, and samples candidate motifs only from this graph. We use three concrete paradigms: *directed loop*, such as $A \rightarrow B \rightarrow C \rightarrow A$; *merge*, where two dependency paths converge to a shared downstream tool, such as $A \rightarrow C$ and $A \rightarrow B \rightarrow C$; and *loop-merge*, where a merge structure is further connected by a closing path back to an earlier tool. A candidate is accepted only if all of its workflow dependencies are present in the original edge set, all task nodes belong to the tool inventory, and the resulting task graph satisfies the intended topological pattern. Thus, topology is fixed before request generation.

Codex workflow instructions. Codex is not prompted to directly invent final samples. Instead, it receives module-level implementation instructions and iteratively edits executable construction programs. The main instructions used in the workflow are summarized below:

- **Motif mining:** "Sample valid directed-loop, merge, and loop-merge motifs from each tool graph."
- **Sample construction:** "For each valid motif, construct task nodes, task links, and task steps in the original dataset schema. Record motif statistics for later inspection."
- **Request generation:** "Generate a natural request from the fixed, hidden tool topology."
- **Validation and merging:** "Validate tools, edges, node-link consistency, identifiers, and motifs; repair or reject errors before merging leakage-controlled splits."

Algorithm 2 Agentic Topology-Guided Data Construction

Require: Dataset collection $\mathcal{D}$, target counts $\{N_d\}$, topology types $\mathcal{C}$
Ensure: Extended datasets with train/test splits

for each dataset $d \in \mathcal{D}$ **do**
  Load tool graph $G_d = (\mathcal{T}_d, \mathcal{E}_d)$ and tool descriptions
  Build adjacency lists and valid edge lookup from $\mathcal{E}_d$
  $\mathcal{S}_d \leftarrow \emptyset$
  while $|\mathcal{S}_d| < N_d$ **do**
    Sample a topology type $c \in \mathcal{C}$
    Mine a candidate motif m of type c from G_d
    if m is not structurally valid under $\mathcal{E}_d$ **then**
      continue
    end if
    Instantiate m as task nodes, task links, and task steps
    Generate a user request conditioned on tools, descriptions, and motif m
    Form candidate sample s
    if VALIDATESAMPLE$(s, \mathcal{T}_d, \mathcal{E}_d, c)$ accepts **then**
      $\mathcal{S}_d \leftarrow \mathcal{S}_d \cup \{s\}$
    end if
  end while
  Group samples by request, task nodes, and task links
  Merge $\mathcal{S}_d$ into the original dataset and assign leakage-controlled splits
end for
return Extended datasets

Validation algorithm. The final validator is deterministic and is applied after request generation. Algorithm 3 summarizes the logic.

D.2 Human Evaluation Protocol

We further conduct a human evaluation to assess the quality of the topology-guided data extension. Following the evaluation dimensions used in TaskBench (Shen et al. (2024b)), each sample is rated from 1 to 5 along three axes: **Naturalness**, which measures whether the user request is fluent and realistic; **Complexity**, which measures whether the task requires non-trivial multi-step tool use; and **Alignment**, which measures whether the request semantics match the annotated task graph.

To reduce single-perspective bias, we invite three human evaluators with different backgrounds: one task-planning researcher who focuses on structural correctness, one domain practitioner who focuses on tool semantics, and one user-facing reviewer who focuses on request naturalness. For each dataset, evaluators inspect a randomly sampled subset of the augmented Cycle examples together with their task nodes and task links. The final score is the average over evaluators and sampled examples. These values are descriptive expert ratings rather than inferential population estimates. This expert assessment is used only to rate benchmark instances; it does not involve crowdsourcing, collection of personal data, or interventions on participants. The results are reported in Table 4.

Algorithm 3 Topology-Guided Sample Validation

Require: Candidate sample s, tool set $\mathcal{T}$, valid edge set $\mathcal{E}$, target topology type c
Ensure: Accept or reject s

```
Extract task nodes V_s and task links E_s from s
if any node in V_s is not in T then
    return Reject
end if
if any link in E_s is not in E then
    return Reject
end if
if task steps, task nodes, and task links are structurally inconsistent then
    return Reject
end if
if s does not satisfy the target topological pattern c then
    return Reject
end if
if s duplicates an existing identifier or an existing workflow key then
    return Repair or reject
end if
return Accept
```

Dataset	*Naturalness* ↑	*Complexity* ↑	*Alignment* ↑	*Overall* ↑
HuggingFace	4.28	4.34	4.16	4.26
Multimedia	4.35	4.29	4.22	4.29
Daily Life	4.06	4.21	3.88	4.05

Table 4: Human ratings (1–5) for the topology-guided data extension.

D.3 Measures to Prevent Data Leakage

We strictly isolate the training and evaluation phases to prevent data contamination:

- **Offline construction:** The data extension is performed before model training and evaluation. No generation-time construction state is exposed to the evaluated planner.
- **Grouped splitting:** Augmented samples are grouped by normalized request, ordered task nodes, and dependency links before train-test assignment. This reduces the risk that near-duplicate workflow graphs appear across training and testing splits.
- **Format preservation:** The augmented samples are merged into the original dataset schema, and all evaluated methods use the same updated split.

D.4 Model Overview and API-Compatible Interface

Figure 2 in the main text summarizes the local open-source TopoPlanner pipeline. Here we clarify the API deployment mode. When the backbone is accessed only through an API, the model exposes text completions but not token embeddings, so the local graph-token interface in the main model cannot be applied directly. We therefore keep the same cellular lifting, retrieval, and multi-dimensional reasoning modules locally, use the API LLM only for textual proposal generation, and let the local topology module perform the final topology-guided selection and graph-consistent decoding.

D.5 TopoPlanner Inference Prompts

Local graph-token prompt. For local open-source LLMs, TopoPlanner follows the same output interface as the evaluated tool-planning benchmarks: the LLM produces one tool name per line, and dependency edges are reconstructed by the shared evaluation protocol. Given a request q, TopoPlanner retrieves a closed cellular subcomplex $\mathcal{X}_q^*$, encodes it into a graph token $\boldsymbol{h}_{\mathcal{X}_q}$, and injects it at the `[graph]...[/graph]` position. The textual part of the prompt is:

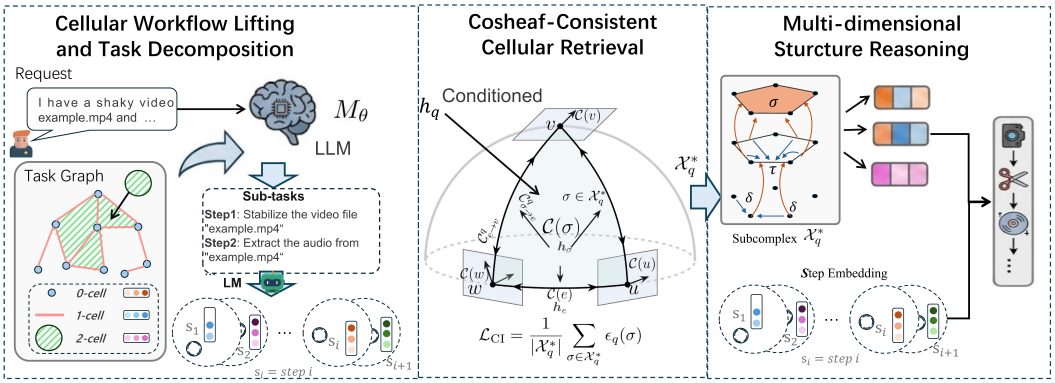

Figure 5: API-compatible TopoPlanner with local topology-guided proposal selection.

Local Planning Prompt:
GOAL #: Please use the provided tools to solve the problem.

GRAPH CONTEXT #: `[graph]` $\boldsymbol{h}_{\mathcal{X}_q}$ `[/graph]`

TASK LIST #: {tool_list}
USER REQUEST #: {user_request}

OUTPUT FORMAT #: Output only one exact tool name per line:
Tool1: tool_name
Tool2: tool_name

API-accessible LLMs. For Qwen3-235B and DeepSeek-V3, the interface is text-only and does not expose token embeddings or support replacing a placeholder with $\boldsymbol{h}_{\mathcal{X}_q}$. We therefore do not claim graph-token injection for these models. Instead, the API LLM predicts textual task steps, candidate tools, and coarse task links from the plain-text tool list and user request, while the local lifted cellular encoder performs topology-guided tool scoring and graph-consistent decoding on top of these proposals, as illustrated in Figure 5. We also do not include GTool as an API baseline, because its graph-enhanced formulation likewise assumes direct model-side graph/embedding injection, which is unavailable under the same black-box API constraint.

E Additional Results on Large LLMs

Table 5 reports the main results on API-accessible large LLMs, including Qwen3-235B and DeepSeek-V3-671B, using the API interface described above. These numbers should therefore be interpreted as gains from topology-guided proposal selection and decoding over black-box textual outputs, rather than as hidden-state graph-token injection into the API models themselves.

F Limitations, Broader Impacts, and Asset Use

Limitations. Our empirical evaluation covers four benchmark-style tool-planning datasets; unconstrained real-world deployments remain unevaluated. The lifting mechanism targets loop, merge, and loop-merge workflow regions, and λ_{CI} is selected per dataset. In the black-box API setting, TopoPlanner uses topology-guided proposal selection and decoding because graph-token injection is unavailable.

Broader impacts. Topology-consistent planning can improve dependency preservation in workflows with feedback, verification, or branch merging. When connected to external APIs or action tools, the same capability can increase execution risk; the released artifacts therefore comprise benchmark data and research code rather than a turnkey autonomous agent.

LLM	Method	HuggingFace			Multimedia			Daily Life		
		n-F1 ↑	*l-F1* ↑	*Acc* ↑	*n-F1* ↑	*l-F1* ↑	*Acc* ↑	*n-F1* ↑	*l-F1* ↑	*Acc* ↑
Qwen3-235B	Direct	73.63	47.77	23.82	77.10	57.49	48.29	83.96	62.88	67.36
	TalkGraph	60.03	28.59	10.00	62.06	39.30	22.32	78.10	57.44	50.48
	Plan-over-Graph	62.54	31.59	9.85	66.17	42.03	22.93	80.73	59.22	58.60
	GNN4TaskPlan	61.99	31.25	11.91	68.45	47.04	30.12	78.10	57.44	50.48
	USPilot	70.48	44.64	21.03	68.28	46.33	31.22	73.74	50.78	41.08
	TopoPlanner	**81.40**	**60.56**	**44.74**	**87.65**	**71.00**	**61.33**	**97.95**	**83.16**	**89.65**
DeepSeek-V3-671B	Direct	76.42	53.82	27.65	77.61	56.09	47.80	82.76	47.15	67.01
	TalkGraph	63.50	35.80	10.88	64.43	40.39	24.76	77.02	55.82	49.66
	Plan-over-Graph	67.33	39.84	13.82	68.89	44.72	25.98	79.36	43.44	58.25
	GNN4TaskPlan	65.32	37.34	12.65	70.06	47.47	31.83	77.02	55.82	49.66
	USPilot	72.01	47.89	23.53	69.07	45.51	30.37	73.31	49.93	41.41
	TopoPlanner	**82.21**	**59.86**	**45.79**	**88.08**	**69.19**	**61.17**	**98.33**	**82.95**	**91.43**

Table 5: API-backbone results with the API-compatible topology interface.

Asset attribution and licensing. The paper builds on existing benchmarks, backbone models, and baseline implementations, all of which are credited in the main text and bibliography. The released repository preserves attribution and provides access notes for the licenses and terms governing TaskBench, ToolBench, and the backbone models or APIs.

G Additional Evaluation Diagnostics

This section complements the main experiments with diagnostics of benchmark validity, cycle semantics, component effects, structural accuracy, transfer, and robustness. Unless stated otherwise, the analyses use the same fixed test examples and evaluation definitions as the main experiments; they characterize TopoPlanner in this controlled setting rather than establish runtime correctness in an unconstrained open-world environment.

G.1 Benchmark Validity Beyond Topology-Guided Augmentation

Motif-leakage audit. We group augmented examples by normalized request and exact ordered node–link structure before splitting. Table 6 reports no repeated normalized construction key between train and test. A few graph-only overlaps remain in HuggingFace and Multimedia because different requests can instantiate the same valid motif; none occur in Daily Life.

Domain	Aug. test	Repeated key	Graph-only overlap
HuggingFace	300	0	21
Multimedia	220	0	13
Daily Life	160	0	0

Table 6: Motif-leakage audit on augmented TaskBench domains.

Held-out topology. We evaluate held-out structure in one controlled setting: a merge topology absent from the corresponding training split. Table 7 reports TopoPlanner and its paired improvement over GTool on the same HuggingFace/Vicuna-7B examples, with confidence intervals obtained by paired bootstrap over test examples. This evidence is specific to the tested setting and does not establish generalization to every unseen workflow topology.

Metric	TopoPlanner	Improvement over GTool (95% CI)
N-F1	0.787	$+0.266\ [0.240, 0.292]$
L-F1	0.491	$+0.306\ [0.273, 0.339]$
Exact match	0.436	$+0.352\ [0.306, 0.400]$
Precedence accuracy	0.591	$+0.358\ [0.321, 0.397]$

Table 7: Held-out merge-topology results on HuggingFace/Vicuna-7B.

Original non-augmented workflows. To test whether the observed gains are confined to topology-guided evaluation examples, Table 8 evaluates original DAG workflows and the subset whose undirected workflow graph has positive cycle rank. The latter contains 47 of 837 HuggingFace DAGs and 12 of 565 Multimedia DAGs. These examples come from the source benchmarks rather than the

augmented loop/merge construction. This diagnostic evaluates performance on original examples; by itself, it does not isolate the effect of using augmented examples during training.

Domain	Backbone	Original DAG (N-F1/L-F1) GTool	Original DAG (N-F1/L-F1) TopoPlanner	Original topology-bearing subset (N-F1/L-F1) GTool	Original topology-bearing subset (N-F1/L-F1) TopoPlanner
HuggingFace	Vicuna-7B	48.80/10.78	84.38/48.00	53.76/17.13	79.60/49.50
HuggingFace	CodeLlama-7B	66.81/25.89	82.49/47.11	73.06/41.10	82.43/54.75
Multimedia	Vicuna-7B	57.79/27.29	82.31/49.01	48.28/15.11	70.22/40.22
Multimedia	CodeLlama-7B	76.85/43.52	84.45/50.00	72.28/35.44	83.23/51.64

Table 8: Results on original workflows (%).

G.2 Executable Interpretation of Workflow Cycles

The *Cycle* category groups loop, merge, and loop-merge motifs because each induces a closed cycle-level region in the lifted cellular representation. A merge is represented as a cycle-level cell because its multiple dependency paths bound a closed region; this structural notion does not imply repeated execution. For motifs containing a directed loop, the loop represents bounded feedback, revision, reuse, or finite transformation at the workflow-graph level, rather than mutually blocking preconditions or an instruction to execute indefinitely. We audit all 2,266 loop-bearing requests using non-exclusive semantic cue categories. Requests unmatched by the cue lexicon are manually assigned to finite transformation only if they specify a finite tool-call sequence.

Subset	#Req.	Condition/stop	Verify/check	Refine/correct	Finite cyclic	Bounded/finite
All loop-bearing	2,266	27.89	52.07	81.16	1.99 (45)	100.00
Directed loop	1,134	12.43	43.03	81.66	3.00 (34)	100.00
Loop-merge	1,132	43.37	61.13	80.65	0.97 (11)	100.00

Table 9: Bounded/finite semantics audit for loop-bearing requests (%; cues may overlap).

We also inspect raw generated tool sequences with a diagnostic guard of $\max(8, 2 \times \text{gold tool count})$. "Repeat" flags any repeated tool call, "over-step" flags a sequence exceeding the guard, and "safe stop" denotes that neither trigger occurs. This offline diagnostic does not replace executor-level stopping.

Method	Safe stop	Repeat	Over-step	C-M safe	Cycle safe
GTool	90.15	6.08	8.43	78.0	82.0
TopoPlanner	99.11	0.79	0.10	96.0	99.0

Table 10: Offline guarded-stop audit (%; repeat/over-step ↓).

G.3 Component Diagnostics

Table 11 extends the main ablation with inference and representation diagnostics. The hypergraph variant tests whether arbitrary higher-order grouping alone explains the gain, while the no-consistency and no-closure variants isolate the two components of the retrieval criterion. The former rows alter the trained architecture or objective, whereas "no Eq. 9 consistency" and "no boundary closure" are inference-time interventions; the two groups should therefore not be interpreted as directly interchangeable ablations. Across both backbones and datasets, the full model exceeds every tested variant on N-F1, L-F1, and Accuracy, although the effect sizes vary by setting.

G.4 Structure-Specific Evaluation

Merge and reuse dependencies. Merge edges point to nodes with in-degree greater than one, reuse edges leave nodes with out-degree greater than one, and M/R recall is computed on their union. The four-domain 7B subgroup contains 804 examples and 2,325 gold merge/reuse edges for each backbone. Merge and reuse are not independent cell types; this diagnostic tests regional dependency preservation.

Backbone	Variant	HuggingFace N/L/Acc	Multimedia N/L/Acc
Vicuna-7B	w/o 2-cell	75.58/46.61/30.88	73.29/50.13/34.15
	w/o retrieval	76.07/47.88/32.21	77.97/55.11/35.37
	w/o $\mathcal{L}_{CI}$	80.69/53.31/38.68	81.47/58.29/40.98
	no Eq. 9 consistency	77.39/49.09/33.68	80.95/57.03/41.71
	no boundary closure	75.66/47.07/32.50	81.41/57.71/43.17
	hypergraph variant	79.47/51.66/37.65	82.38/59.73/44.63
	TopoPlanner	**82.23/55.72/42.35**	**86.80/66.24/54.76**
CodeLlama-7B	w/o 2-cell	78.95/51.87/35.29	75.90/56.35/43.54
	w/o retrieval	79.75/56.96/35.59	83.89/62.92/49.27
	w/o $\mathcal{L}_{CI}$	80.80/53.83/38.82	87.12/67.61/53.54
	no Eq. 9 consistency	79.44/53.19/38.24	86.24/64.34/53.41
	no boundary closure	78.74/52.00/35.59	85.59/66.14/55.37
	hypergraph variant	77.35/50.24/31.32	84.20/62.79/49.15
	TopoPlanner	**83.43/59.62/47.35**	**89.08/69.59/62.20**

Table 11: Extended component diagnostics (%); training and inference variants are not directly comparable.

Method	Backbone	n	N-F1	L-F1	M/R Rec.
GTool	Vicuna-7B	804	51.18	22.06	19.04
TopoPlanner	Vicuna-7B	804	76.16	46.92	39.72
GTool	CodeLlama-7B	804	64.56	35.27	29.06
TopoPlanner	CodeLlama-7B	804	78.94	49.91	41.01

Table 12: Merge/reuse subgroup results (%).

Execution order. For each predicted tool sequence, we check each gold dependency $u \to v$. Pair measures macro-averaged correctness over gold dependency pairs, with missing endpoints counted as errors. Dep. is an example-level all-or-nothing score requiring every gold dependency's endpoints to be present and ordered correctly.

		HuggingFace		Multimedia		Daily Life	
Method	Backbone	Dep.	Pair	Dep.	Pair	Dep.	Pair
GTool	Vicuna-7B	9.71	24.18	8.29	23.69	39.86	57.32
TopoPlanner	Vicuna-7B	27.94	51.74	40.37	64.29	60.14	79.50
GTool	CodeLlama-7B	25.74	44.91	32.56	49.73	48.11	62.30
TopoPlanner	CodeLlama-7B	33.24	58.64	48.78	71.80	63.57	83.28

Table 13: Execution-order results (%).

G.5 Paired Statistical Significance

We pair TopoPlanner and GTool predictions on the same fixed test instances. N-F1, L-F1, and Accuracy differences are evaluated by example-level paired bootstrap with 10,000 resamples. Table 14 reports representative paired differences and 95% confidence intervals. All listed two-sided paired-bootstrap tests have $p < 0.0002$. These intervals quantify example-level uncertainty for fixed checkpoints, not variation across training runs.

Domain	Backbone	N-F1 difference	L-F1 difference	Accuracy difference
HuggingFace	Vicuna-7B	$+0.298\ [0.276, 0.319]$	$+0.359\ [0.331, 0.387]$	$+0.349\ [0.312, 0.387]$
HuggingFace	CodeLlama-7B	$+0.149\ [0.129, 0.169]$	$+0.201\ [0.172, 0.229]$	$+0.228\ [0.188, 0.269]$
Multimedia	Vicuna-7B	$+0.392\ [0.373, 0.411]$	$+0.466\ [0.440, 0.492]$	$+0.482\ [0.446, 0.517]$
Multimedia	CodeLlama-13B	$+0.162\ [0.144, 0.182]$	$+0.185\ [0.162, 0.208]$	$+0.283\ [0.250, 0.317]$
Daily Life	Vicuna-13B	$+0.166\ [0.144, 0.189]$	$+0.166\ [0.142, 0.189]$	$+0.301\ [0.261, 0.340]$
Daily Life	CodeLlama-13B	$+0.144\ [0.124, 0.165]$	$+0.139\ [0.116, 0.162]$	$+0.261\ [0.222, 0.301]$

Table 14: Paired TopoPlanner–GTool differences with 95% bootstrap CIs (scores in $[0, 1]$).

G.6 Cross-Domain Transfer and Incomplete-Graph Robustness

Graph-grounded transfer. The source-domain checkpoint is evaluated in a different target domain without target-domain parameter updates. Target-domain tool descriptions and the target dependency graph remain available at inference. This is graph-grounded transfer to unseen-but-described tools and target graphs, not unconstrained discovery of unregistered tools.

Backbone	Source	Target	Direct			GTool			TopoPlanner		
			N	L	Acc	N	L	Acc	N	L	Acc
Vicuna-7B	Daily Life	Multimedia	.2897	.0038	.0171	.2288	.0106	.0122	.5120	.1820	.0646
Vicuna-7B	HuggingFace	Daily Life	.3200	.1274	.0756	.3195	.1251	.0739	.7436	.4012	.2474
Vicuna-7B	Multimedia	HuggingFace	.4218	.1151	.0647	.3576	.0766	.0294	.4571	.1736	.0676
CodeLlama-7B	Daily Life	Multimedia	.4963	.0527	.1122	.3893	.1266	.0268	.5717	.2905	.1305
CodeLlama-7B	HuggingFace	Daily Life	.6844	.4362	.3265	.6759	.4214	.3247	.8955	.6407	.6065
CodeLlama-7B	Multimedia	HuggingFace	.5619	.2888	.0912	.5821	.2445	.1059	.6722	.3567	.1868

Table 15: Cross-domain transfer without target-domain updates (scores in $[0, 1]$).

Incomplete dependency graphs. We keep the gold workflows unchanged, remove a controlled fraction of dependency edges from the input graph before training and inference, and recompute edge and 2-cell features from the remaining graph. Table 16 shows that node selection is comparatively stable, whereas link recovery degrades more strongly because it depends directly on the available structural evidence. This diagnostic covers controlled missing edges, not general corruption.

Dataset	Missing edges	N-F1	L-F1	Acc
HuggingFace	0%	83.43	59.62	47.35
HuggingFace	10%	82.01	53.50	43.68
HuggingFace	20%	80.77	44.17	40.00
HuggingFace	50%	80.49	31.61	38.09
Multimedia	0%	89.08	69.59	62.20
Multimedia	10%	87.52	60.97	58.78
Multimedia	20%	87.79	58.96	59.63
Multimedia	50%	87.69	43.42	58.78

Table 16: Robustness to missing dependency edges (%).

G.7 Retrieval Scale and Workflow Length

Retrieval scores cells in the available tool graph rather than enumerating every path or subgraph. All tool nodes and 2-cells are retained; only the dense Daily Life edge set uses a deterministic 512-edge prefilter. Table 17 reports timing without LLM generation.

Dataset	Pool (nodes/edges/2-cells)	Edge candidates	CPU ms
HuggingFace	23/225/54	225	23.65
Multimedia	40/449/85	449	33.11
Daily Life	40/1560/146	512	39.45
ToolBench	245/366/19	366	22.89

Table 17: Retrieval-pool size and mean CPU time over 50 fixed queries.

Table 18 groups examples by gold tool-call length and compares TopoPlanner with GTool on the same examples. The gains generally increase for medium and long workflows, while the lower length MAE indicates that the improvement is not explained only by producing longer sequences.

Dataset	Gold length	#Ex.	ΔN-F1	ΔL-F1	ΔAcc	Length MAE (GTool → Ours)
HuggingFace	1–2	31	+5.64	+11.83	+12.10	0.32 → 0.19
	3–4	484	+14.94	+18.59	+22.37	0.67 → 0.27
	≥ 5	165	+24.62	+26.03	+18.34	1.52 → 0.96
Multimedia	1–2	45	+6.08	+13.33	+15.55	0.20 → 0.07
	3–4	513	+23.90	+27.98	+38.84	0.94 → 0.31
	≥ 5	262	+27.50	+30.88	+33.31	1.68 → 0.65
Daily Life	1–2	25	+1.38	+2.22	+4.00	0.12 → 0.09
	3–4	416	+15.44	+16.87	+32.63	0.71 → 0.07
	≥ 5	141	+27.14	+28.07	+40.07	1.99 → 0.17

Table 18: TopoPlanner–GTool differences by gold tool-call length (percentage points).

H Workflow Examples and Structural Interpretation

The following examples illustrate how loop, merge, and loop-merge workflows are represented in the lifted complex. They explain the structural information available to TopoPlanner; because per-example baseline predictions are not shown, they are not presented as qualitative evidence of baseline failure.

H.1 Case Study 1: HuggingFace

Example ID: `paper_cycle_huggingface_0005`.
Request summary: extract information from a historical document, highlight key entities and dates, convert the result to speech, enhance the audio, transcribe it back, and then revisit the original document question answering step using the refined transcript.

Topology. This workflow forms a loop-merge pattern: `Document QA` → `Token Classification` → `Text-to-Speech` → `Audio-to-Audio` → `ASR` → `Document QA`, with an additional direct edge `Document QA` → `Text-to-Speech`. The direct edge matters because the speech synthesis stage must inherit both the initial extraction and the entity-focused refinement, while the ASR output closes a feedback loop for iterative correction.

Structural interpretation. The lifted representation places the direct merge path and the feedback path on the boundary of the same cycle-level region. Boundary-closed retrieval therefore exposes both relations to the planner rather than representing only a single linear path.

H.2 Case Study 2: Multimedia

Example ID: `paper_cycle_multimedia_0002`.
Request summary: transcribe a raw video interview, summarize it, generate relevant topics, rewrite the narration, denoise the audio, synchronize the new voiceover with the video, and refine the narration again if synchronization is unsatisfactory.

Topology. The workflow is: `Video-to-Text` → `Text Summarizer` → `Topic Generator` → `Voice Changer` → `Audio Noise Reduction` → `Video Synchronization` → `Video-to-Text`, with an additional direct edge `Video-to-Text` → `Topic Generator`. This creates a multi-stage feedback loop in which the synchronization result may trigger another transcription-and-revision pass.

Structural interpretation. The region contains both the shortcut into `Topic Generator` and the closing edge back to `Video-to-Text`, making the dual-source topic dependency and synchronization feedback jointly available during cellular reasoning.

H.3 Case Study 3: Daily Life

Example ID: `paper_cycle_dailylife_0005`.
Request summary: share a conference poster, print it for a manager, book a flight, pay the registration bill, search for attendee reviews, and if necessary share a cancellation notice and rebook a later flight.

Topology. The workflow is a loop-merge pattern: `share_by_social_network` → `print_document` → `book_flight` → `daily_bill_payment` → `search_by_engine` → `share_by_social_network`, with an additional direct edge `share_by_social_network` → `book_flight`. The request therefore couples communication, logistics, payment, and post-hoc revision in a closed rather than one-way workflow.

Structural interpretation. Boundary closure keeps the direct booking dependency together with the late feedback edge that reopens communication and trip planning, representing the request as a closed region rather than a one-way chain.

H.4 Case Study 4: ToolBench

Example ID: `toolbench_circle_0005`.
Request summary: generate a face image, consult a zodiac-sign prediction, produce a flirty line conditioned on that prediction, and then adjust the generated face based on the tone of the final line.

Topology. This ToolBench example forms a compact loop-merge structure: `generate a face` → `zodiac prediction` → `generate flirty line` → `generate a face`, with an additional direct edge `generate a face` → `generate flirty line`. The direct edge expresses that the final line depends not only on zodiac information but also on the face-generation context, while the feedback edge turns the line into a refinement signal for the image generator.

Structural interpretation. Here the cycle-level cell couples the direct face-to-line dependency with the feedback edge from the line to image generation. This makes the merge and refinement relations jointly available even in ToolBench's larger tool inventory.

I Additional Related Work

Topological Deep Learning. Topological deep learning (TDL) builds upon early advances in Topological Signal Processing (TSP) (Barbarossa & Sardellitti (2020); Sardellitti et al. (2021)), which emphasized that many real-world systems exhibit interactions that go beyond pairwise relations and are more naturally described by high-dimensional topological structures. This insight has motivated the generalization of classical graph-theoretic tools from pairwise graphs to richer topological domains, such as simplicial complexes (SCs), which model interactions among fixed-size groups, and cell complexes (CWs), which further generalize this representation to arbitrary polyhedral cells.

Along this line, extensions of the Weisfeiler–Lehman (WL) test to SCs and CWs (Bodnar et al. (2021b;a)) have provided theoretical foundations for distinguishing and propagating high-dimensional structural information, thereby enabling principled message passing beyond nodes and edges. Building on these ideas, convolutional architectures operate on SCs and CWs (Ebli et al. (2020); Yang et al. (2022); Hajij et al. (2020); Yang & Isufi (2023); Hajij et al. (2022)), while attentional variants model higher-dimensional neighborhoods (Goh et al.; Giusti et al. (2023)).

To unify these heterogeneous developments, the combinatorial complex (CC) framework (Hajij et al. (2023)) was introduced as a general abstraction that subsumes SCs, CWs, and hypergraphs within a single message-passing formalism, allowing a systematic treatment of multi-dimensional relational structures. In parallel, Sheaf Neural Networks (SNNs) (Hansen & Ghrist (2019); Hansen & Gebhart (2020); Bodnar et al. (2022); Battiloro et al. (2024); Barbero et al. (2022); Wei & Wei (2025)) have explored a complementary perspective by modeling local consistency constraints via sheaf theory, demonstrating strong performance in settings with heterophily and structured local interactions.

Recently, topological perspectives have been applied to analyze reasoning in large language models. Minegishi et al. (2025) introduce *reasoning graphs* by clustering hidden-state representations across

reasoning steps and analyze their structural properties—such as cyclicity, diameter, and small-world characteristics—on mathematical reasoning benchmarks. These developments suggest that topological structure is increasingly relevant to LLM reasoning, but direct topology-aware planning over tool workflows remains underexplored, which is the gap our work addresses.

J Computational Complexity Analysis

This section characterizes the computational cost of TopoPlanner. The total computational overhead can be decomposed into four stages aligned with the model design: cellular workflow lifting, cosheaf-consistent cellular retrieval, multi-dimensional structural reasoning, and LLM tool planning.

J.1 Theoretical Analysis

1. Topological Lifting. The construction extracts a fundamental cycle basis from the underlying undirected graph $G_u = (V, E)$. For a connected graph, we first construct a spanning tree $\mathcal{T}$ in $O(|V| + |E|)$ time. Each of the $\beta_1 = |E| - |V| + 1$ non-tree edges induces one fundamental cycle; standard cycle-basis algorithms compute the basis in $O(|V| \cdot |E|)$ time.

- **Efficiency:** This lifting can be precomputed for each dataset/tool graph and cached together with the graph features. During training and inference, TopoPlanner reads the precomputed cellular structure and only applies request-dependent retrieval and encoding.

2. Cosheaf-Consistent Cellular Retrieval. For each request, TopoPlanner scores cells by request relevance and cosheaf-induced boundary compatibility. The practical implementation first applies a cheap edge prefilter, then performs closure-preserving greedy selection. Its cost is linear in the number of retained candidate cells after prefiltering; Table 17 reports the measured retrieval time separately from LLM decoding.

3. Multi-Dimensional Structural Reasoning. Our encoder employs a message-passing neural network over the retrieved cell complex. Let $\mathcal{X}_q^*$ denote the retrieved subcomplex and $\mathcal{E}_{adj}$ denote the boundary and coface adjacency connections in the lifted complex.

- One reasoning layer costs $O(|\mathcal{X}_q^*| \cdot d^2 + |\mathcal{E}_{adj}| \cdot d)$ for hidden dimension d.
- **Linear scaling in retained incidences:** Unlike global Transformer attention, which scales as $O(|\mathcal{X}_q^*|^2)$, message passing is confined to boundary and coface relations. Its cost is therefore linear in the retained incidence connections; for the dense Daily Life edge pool, retrieval first applies the 512-edge prefilter in Table 17.

4. LLM Tool Planning Overhead. The local open-source variant decodes autoregressively with a frozen planner LLM and one injected graph token. The API variant uses topology to rerank proposals.

- LLM latency dominates, with approximate per-token complexity $O(T \cdot d_{model}^2)$.
- **Measured retrieval cost:** Table 17 reports mean CPU times of 22.89–39.45 ms over 50 fixed queries. These measurements exclude LLM generation and therefore do not establish end-to-end latency overhead.

J.2 Scalability Discussion

Enumerating all simple cycles can be exponential, especially in dense graphs. TopoPlanner instead constructs one fundamental cycle per non-tree edge, so the untruncated number of 2-cells equals the cycle rank $\beta_1 = |E| - |V| + 1$ for a connected graph. This remains linear in graph size even for the dense Daily Life graph; its request-time retrieval additionally applies the deterministic 512-edge prefilter reported in Table 17.