EvoFormat: Discovering Effective Graph-to-Text Formats via Rule-Based Evolutionary Search

Kazuya Kakizaki, Taiki Miyagawa, Hideyuki Oiso, Ryo Furukawa
NEC Corporation
{kazuya1210, miyagawataik, oiso-hideyuki, rfurukawa}@nec.com

Abstract

Encoding graphs as text prompts enables LLM-based graph reasoning. However, effective graph-to-text encodings depend on the downstream task, graph structure, and target LLM, making manual format design time-consuming and brittle. We propose EvoFormat, a black-box evolutionary search framework that automatically discovers interpretable graph-to-text formats represented as human-readable *formatting rules* over node naming, relation expression, and global arrangement. Using only task-level feedback from LLM outputs, EvoFormat iteratively mutates and selects rules without gradients or internal model access. Across synthetic GraphQA and IMDB-MULTI-derived structural QA tasks over unattributed graphs, EvoFormat outperforms strong hand-crafted encodings in most settings.

1 Introduction

Large language models (LLMs) have demonstrated strong capabilities across a wide range of reasoning and generation tasks, motivating their extension beyond unstructured text to structured data. Among such data, graphs are fundamental in many domains (e.g., knowledge graphs, molecular structures, and social networks) and have attracted growing interest in LLM-based graph reasoning (Ren et al., 2024). While adapting LLMs to graphs via instruction tuning and preference alignment can improve performance (Chen et al., 2024a; Wang et al., 2024; Ye et al., 2024), such adaptation typically involves data curation, computational resources for training, and engineering effort, which may introduce non-trivial practical overhead. These constraints have spurred an alternative line of work that keeps the LLM frozen and represents graphs as text embedded in prompts, enabling graph tasks without modifying the underlying LLM (Fatemi et al., 2024; Guo et al., 2023; Zhao et al., 2023).

However, the effectiveness of this paradigm critically depends on how graphs are represented as text. In practice, even for the same graph, LLM performance can vary substantially with seemingly minor choices, such as node identifiers, edge serialization, and the ordering of edges in the text encoding (Fatemi et al., 2024; Xypolopoulos et al., 2024). Moreover, what constitutes an effective encoding often depends on the downstream task and graph structure, and can also vary with the target LLM configuration (e.g., model capacity). Consequently, manually designing effective graph-to-text formats is time-consuming and often brittle.

To address this challenge, prior work has proposed training-based methods that adapt how graphs are provided to the target LLM. A representative direction learns continuous graph representations and injects them into the LLM input space (Perozzi et al., 2024; Kim et al., 2025; He et al., 2024; Tian et al., 2024; Chen et al., 2024b). While effective, such continuous encodings are often less interpretable than explicit textual formats, offering limited transparency into how graph information is represented. Moreover, they typically require white-box access to the target LLM (i.e., access to internal computations for gradient-based optimization), and thus do not directly apply when the target LLM is available only through black-box interfaces (e.g., an API). This motivates automated methods for discovering effective, interpretable graph-to-text formats using only black-box feedback from the target LLM.

To overcome these limitations, we propose **EvoFormat**, a *black-box* evolutionary search framework that *automatically* discovers effective and *interpretable* graph-to-text formats for LLM prompting. EvoFormat represents each format as a *formatting rule* for unattributed graphs, composed of three components: how nodes are named, how relations are expressed, and how the entire graph description is arranged (structured and ordered) as

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 39404–39415
October 24-29, 2026 ©2026 Association for Computational Linguistics

a single input text. Given a candidate rule, the induced encoder deterministically converts any input graph into text, which is then evaluated by downstream task performance of the target LLM. EvoFormat searches over the space of formatting rules by iteratively mutating rule components and selecting improved variants, using an LLM to propose rule candidates. Because this search relies only on task-level feedback from LLM outputs, EvoFormat requires neither gradients nor access to internal model states, making it applicable under API-only, black-box access to the target LLM.

Our contributions are threefold: (1) We introduce an interpretable *formatting rule* parameterization for unattributed graphs that decomposes graph-to-text formats into node naming, edge expression, and global arrangement, enabling systematic search over a rich format space; (2) Building on this rule space, we propose EvoFormat, a gradient-free evolutionary framework that searches over *formatting rules* using only black-box feedback from the target LLM; (3) Through experiments on six synthetic GraphQA tasks and TUDataset-derived IMDB-MULTI structural QA tasks over real unattributed graphs across multiple LLMs, we show that the proposed rule space and LLM-based initialization already produce strong candidates, while evolutionary variation provides additional, setting-dependent refinement when task headroom remains. Overall, EvoFormat achieves a higher reported mean accuracy than the best hand-crafted format in all GraphQA settings and provides a competitive automatic alternative to manual format selection on IMDB-MULTI.

2 Related Work

Automated Prompt Optimization (APO). APO aims to improve the downstream performance of an LLM f by optimizing an auxiliary prompt p (e.g., instructions, demonstrations, or templates) while treating the task input x as fixed, typically under black-box access to the target LLM (Ramnath et al., 2025; Li et al., 2025). Recent black-box approaches can be broadly grouped into feedback-driven LLM-in-the-loop search (Pryzant et al., 2023; Yang et al., 2024b), evolutionary prompt optimization (Fernando et al., 2024; Guo et al., 2024; Xu et al., 2022), and reinforcement-learning-based optimization of discrete prompts (Deng et al., 2022; Kwon et al., 2024), following the optimization perspective of Li et al. (2025). While effective, most APO methods optimize p rather than the input representation x, and therefore do not directly optimize how an input graph is encoded as text.

Some studies go beyond optimizing p and instead improve performance by rewriting the task input x. For example, they rewrite natural-language questions for Text-to-SQL using database content to reduce ambiguity (Mao et al., 2024), or rewrite source sentences to improve machine translation (Ki and Carpuat, 2025). However, these methods typically assume unconstrained natural-language inputs. In contrast, graph-to-text encodings must satisfy strict structural validity constraints, including consistent node references and edge realizations, making it non-trivial to apply generic rewriting methods. This motivates dedicated search methods for structured, human-interpretable *formatting rules* tailored to graph encodings.

Graph Reasoning with Fixed LLMs. A growing line of work enables graph reasoning with frozen LLMs by encoding graphs as explicit text prompts. Fatemi et al. (2024) propose nine human-designed formats and show large performance variations across formats and task settings. Zhao et al. (2023) further argue for reasoning in the text space and propose constructing a graph-syntax tree and serializing it into a structured text sequence for LLM prompting, while Xypolopoulos et al. (2024) study graph linearization into token sequences using centrality- and degeneracy-based orderings combined with node relabeling. Similar sensitivity to input representation has also been observed beyond graphs; for instance, a Text-to-SQL study shows that different prompt representations of database schemas affect LLM performance (Gao et al., 2024). In LLM-based node classification, Wu et al. (2025) show that the usefulness of structural context can depend on the task setting, graph properties, model choice, and prompt design. Overall, these studies highlight that effective use of graph information by LLMs is highly representation-dependent, yet graph-to-text formats are typically specified manually, without automatic search or learning.

Training-based methods instead learn continuous interfaces for injecting graph information into LLMs. For example, Perozzi et al. (2024) train graph encoders that map the given input graph into a continuous prompt representation for a frozen LLM, while Chen et al. (2024b) reorganize graph nodes into structure-aware embedding sequences

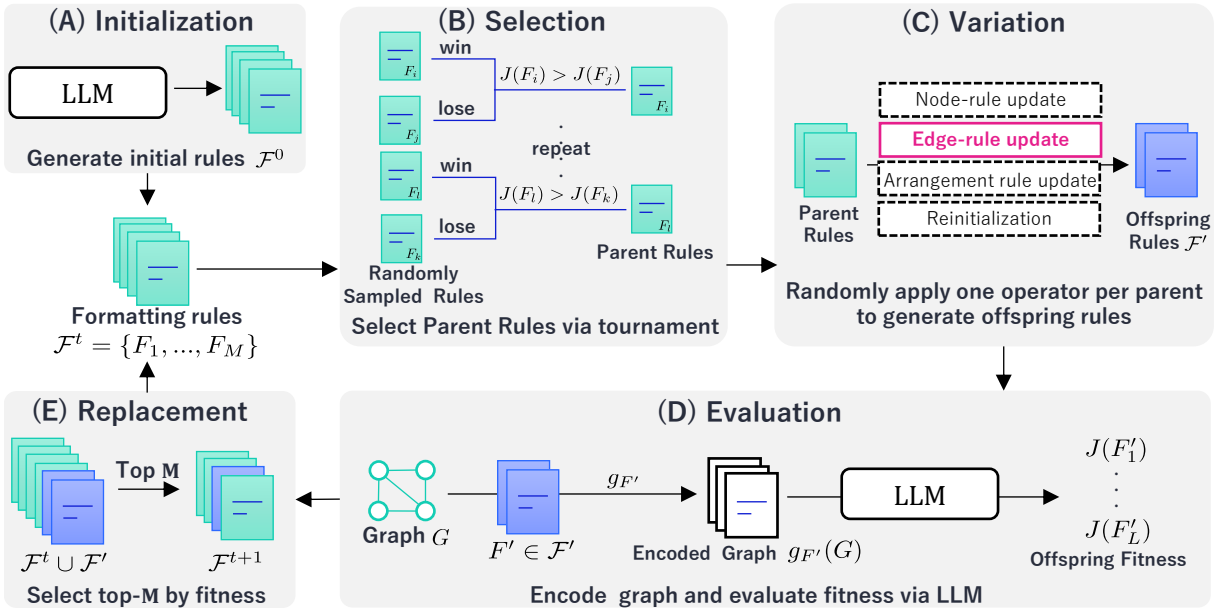


Figure 1: Overview of EvoFormat evolutionary search. **(A)** An initial population of formatting rules is generated by an LLM, and iteratively updated through **(B)** selection, **(C)** mutation, **(D)** evaluation, and **(E)** replacement.

and learn a projector into the LLM token space. Other methods make prompting query-dependent, e.g., by learning query-aware graph prompt embeddings (Kim et al., 2025; Tian et al., 2024) or by retrieving query-relevant graph evidence and using it to augment generation (He et al., 2024). Related graph foundation model approaches further learn language-based graph vocabularies for text-attributed graphs (Zhu et al., 2025). While effective, these learned approaches typically require white-box access for gradient-based optimization and are often less interpretable than explicit textual formats, motivating black-box methods that automatically discover interpretable graph-to-text formatting rules.

More recently, Ranaldi et al. (2025) provide behavioral evidence suggesting that lexical, hierarchical, and topological KG knowledge internalized during pretraining is reused by LLMs in downstream Text-to-SPARQL tasks. Their work thus probes the model's existing KG knowledge, whereas EvoFormat addresses the complementary question of how an explicit input graph should be represented at inference time.

3 Problem Formulation

Consider a graph-to-text encoder $g : \mathcal{G} \to \mathcal{T}$, where $\mathcal{G}$ denotes the graph space and $\mathcal{T}$ the text space. An LLM f takes as input the encoded graph $g(G) \in \mathcal{T}$, with $G \in \mathcal{G}$, together with a query $q \in \mathcal{T}$, and outputs an answer $f(g(G), q) \in \mathcal{Y}$, where $\mathcal{Y}$ denotes the answer space. Our objective is to optimize the graph-to-text encoder g so as to maximize performance on a downstream task (e.g., cycle check).

Following (Fatemi et al., 2024), we assume a supervised setting for selecting the encoder g. The training dataset $\mathcal{D}$ consists of graph–query–answer triples: $\mathcal{D} = \{(G_i, q_i, y_i) \mid G_i \in \mathcal{G},\ q_i \in \mathcal{T},\ y_i \in \mathcal{Y}\}$, where i indexes the samples. The optimization objective is then formally written as

$$g^* = \text{argmax}_g \ \text{Score}\big(f(g(G_i), q_i),\ y_i;\ \mathcal{D}\big) \quad (1)$$

for a task-dependent scoring function. Unlike (Fatemi et al., 2024), we optimize g while keeping the query q fixed.

Importantly, we assume that the LLM f is a black box, as in commercial API-based models; i.e., its internal parameters and gradients are inaccessible, and only input-output pairs can be observed. Under this assumption, directly optimizing Eq. (1) with gradient-based methods is not available, which motivates gradient-free optimization strategies.

4 EvoFormat

We propose **EvoFormat**, a gradient-free evolutionary framework for optimizing graph-to-text encoders under black-box LLM access. Instead of parameterizing the encoder with model parameters, EvoFormat represents it using a human-readable

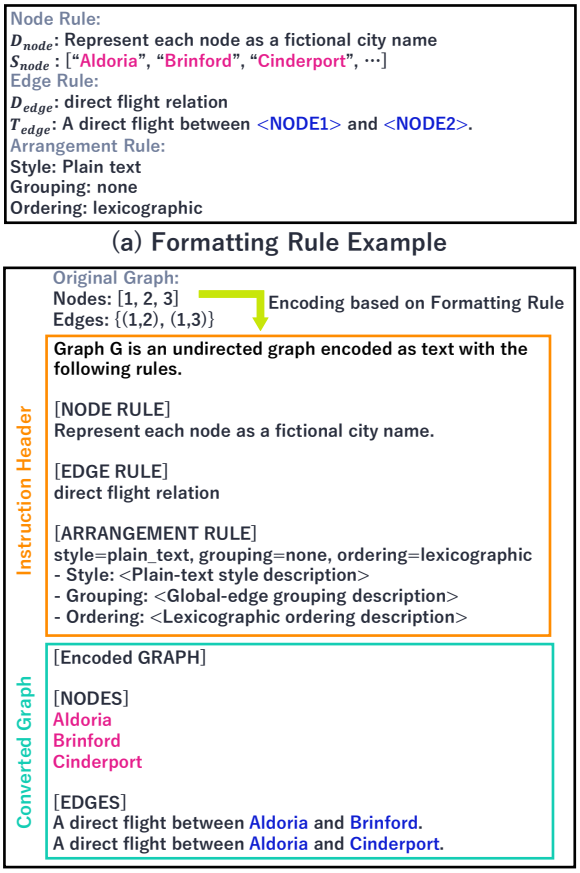


Figure 2: Example formatting rule and its induced graph-to-text encoding.

formatting rule $F \in \mathcal{F}$ over unattributed graphs, where $\mathcal{F}$ denotes the space of formatting rules. F specifies how a graph is converted into text and deterministically induces a graph-to-text encoder. This formulation casts the optimization problem as a search over formatting rules. To solve it, EvoFormat uses an evolutionary search consisting of five stages: (A) initialization, (B) selection, (C) variation, (D) evaluation, and (E) replacement of formatting rules, with all decisions driven solely by observed task performance (Figure 1). In the following, we first describe the structure of formatting rules and the rule-induced encoders, and then detail each stage of the evolutionary search procedure corresponding to steps (A)–(E).

4.1 Definition of Formatting Rules

A formatting rule $F \in \mathcal{F}$ is defined as a structured textual specification that determines how the elements of an unattributed graph are represented in text. In this work, we consider formatting rules composed of multiple interpretable components, each responsible for a different aspect of the graph-to-text conversion. Specifically, a rule F consists of (i) a *node rule*, (ii) an *edge rule*, and (iii) an *arrangement rule*, as illustrated in Figure 2 (a).

Node Rule. The node rule specifies how graph nodes are encoded and is defined by a textual description D_{node} and a corresponding node set S_{node}. D_{node} determines the semantic or stylistic form of node expressions and is included in the encoded prompt to explain the meaning of the node representations to the downstream LLM. It also serves as a high-level specification that guides rule updates in EvoFormat (see Section 4.3). S_{node} provides concrete textual realizations that serve as node-name candidates during encoding.

We consider two types of node rules: *Natural Language Rules*, which use human-readable but fictional names following a specified style, and *Symbolic Identifier Rules*, which represent nodes using abstract identifiers such as `N_i`. These rule types allow EvoFormat to explore node representations ranging from semantically expressive to purely symbolic.

Edge Rule. The edge rule specifies how relations between nodes are encoded and is defined by a textual description D_{edge} and an edge template T_{edge} containing the placeholders `<NODE1>` and `<NODE2>`. D_{edge} determines the semantic or stylistic form of edge expressions and is included in the encoded prompt to explain the meaning of the edge representations to the downstream LLM. It also serves as a high-level specification that guides rule updates in EvoFormat. T_{edge} provides a concrete textual pattern for instantiating each edge by replacing the placeholders with encoded node names.

We consider two types of edge rules: *Natural Language Rules*, which express relations using descriptive phrases, and *Symbolic Identifier Rules*, which encode relations using abstract symbolic patterns. This design enables EvoFormat to explore relational representations ranging from semantically rich natural language expressions to abstract symbolic patterns.

Arrangement Rule. The arrangement rule specifies how encoded nodes and edges are organized into a textual representation. It is parameterized by three choices: an output style, a grouping option, and an ordering strategy. The output style determines the overall serialization format, chosen from plain text, Markdown, JSON-like, XML-like,

and Python-like representations. The grouping option determines whether each edge is listed once in a global edge section or grouped under its incident nodes. The ordering strategy determines whether nodes and edges are sorted lexicographically by their encoded node names or by node degree. Together, these components define a structured arrangement space for graph information in text, enabling EVOFORMAT to explore diverse formatting variations.

4.2 Rule-Induced Graph-to-Text Encoders

Given a formatting rule $F \in \mathcal{F}$, the induced encoder g_F deterministically maps a graph $G = (V, E)$ to text by applying the components of F. It first assigns each node $v \in V$ a unique textual realization from the node set S_{node}. If the number of usable generated node names differs from $|V|$, the list is deterministically truncated when too long or extended when too short, either by continuing an identifiable numbering pattern or by appending deterministic suffixes. Each edge $(u, v) \in E$ is then rendered by instantiating the edge template T_{edge} with the corresponding node names. Finally, the arrangement rule serializes the node and edge strings according to its output style, grouping option, and ordering strategy.

The resulting encoded graph text consists of an instruction header describing the formatting rule and the encoded graph representation produced by g_F. This text is concatenated with the downstream task query and provided to the LLM. Thus, each formatting rule induces a deterministic graph-to-text encoder $g_F : \mathcal{G} \to \mathcal{T}$, as illustrated in Figure 2(b).

4.3 Evolutionary Search Procedure

Under the restriction to rule-induced encoders g_F, the optimization objective in Eq. (1) can be rewritten as a search over formatting rules:

$$F^* = \arg\max_{F \in \mathcal{F}} \text{Score}\big(f(g_F(G_i), q_i),\, y_i;\, \mathcal{D}\big). \tag{2}$$

EVOFORMAT optimizes this objective with an evolutionary search over formatting rules (Figure 1). It initializes a population of candidate rules and repeatedly applies parent selection, rule variation, fitness evaluation, and replacement. Fitness is measured by downstream task performance, and the best-performing rules are retained across generations. By iterating these stages, EVOFORMAT searches for effective graph-to-text formats under black-box LLM access.

4.3.1 Population Initialization

EVOFORMAT begins by constructing an initial population $\mathcal{F}^0 \subseteq \mathcal{F}$ of M formatting rules using an LLM (Figure 1(A)). To encourage diversity, the initial population is evenly split between Natural Language Rules and Symbolic Identifier Rules. For each rule, the LLM generates the node-rule description D_{node}, node set S_{node}, edge-rule description D_{edge}, and edge template T_{edge} according to a predefined initialization configuration. The arrangement rule is initialized separately by randomly sampling its three components: output style, grouping option, and ordering strategy. After initialization, each rule $F \in \mathcal{F}^0$ is evaluated on the downstream task using its induced encoder g_F, and its fitness score $J(F)$ is computed via the task-dependent score in Eq. (2).

4.3.2 Parent Selection

In the selection step, EVOFORMAT chooses parent rules from the current population to guide the search toward promising regions (Figure 1(B)). Specifically, we use tournament selection with a tournament size of two. To generate an offspring set of size K, the tournament is repeated K times: two rules are sampled uniformly at random, and the one with the higher fitness is selected as the parent for applying a variation operator. This strategy promotes exploitation of high-performing rules while retaining exploration through stochastic sampling.

4.3.3 Rule Variation

In the variation step, EVOFORMAT explores the formatting-rule space by applying stochastic variation operators to parent rules (Figure 1(C)). At each generation, it produces an offspring set $\mathcal{F}' \subseteq \mathcal{F}$ of size K from parents selected from the current population. For each offspring, EVOFORMAT samples one operator uniformly from four operators: reinitialization, node-rule update, edge-rule update, and arrangement-rule update. Among these, reinitialization, node-rule update, and edge-rule update use the LLM, whereas arrangement-rule update is performed programmatically.

History-aware operator prompts. To encourage exploration beyond small surface edits, the LLM-based operators append a compact *history-avoid context* to the prompt. This context lists node-rule descriptions, edge-rule descriptions, edge templates, and frequently used edge templates from same-type rules in the current population and previously generated candidates. The LLM is instructed

to treat these items as negative examples and to avoid copying, paraphrasing, or making small surface edits of them. We exclude node sets from the context to avoid inducing the LLM to copy long node-name lists. The history is constructed programmatically and used only as a prompt-level soft constraint; it does not add LLM calls.

Concretely, EVOFORMAT employs the following variation operators:

1. **Reinitialization.** This operator discards the parent rule and generates a fresh rule of the same rule type, i.e., Natural Language or Symbolic Identifier. The LLM generates D_{node}, S_{node}, D_{edge}, and T_{edge}, while the arrangement rule is randomly initialized. This operator provides a global exploratory move.

2. **Node-rule update.** This operator rewrites the node-rule description D_{node} and regenerates the node set S_{node} using the LLM. The update is guided to remain compatible with the current edge rule and edge template, and can substantially change how nodes are represented.

3. **Edge-rule update.** This operator rewrites the edge-rule description D_{edge} together with the edge template T_{edge} using the LLM. The update is constrained to remain compatible with the current node rule and node-name candidates, and can substantially alter how edges are expressed.

4. **Arrangement-rule update.** This operator modifies the arrangement rule by resampling one of its three components: output style, grouping option, or ordering strategy. The selected component is guaranteed to differ from its current value, while the remaining components are kept unchanged.

4.3.4 Fitness Evaluation

In the evaluation step, EVOFORMAT assesses the quality of candidate rules by measuring their task performance (Figure 1(D)). After variation, each offspring formatting rule $F' \in \mathcal{F}'$ is evaluated by applying its induced encoder $g_{F'}$ to the training graphs. The resulting LLM predictions are compared against the ground-truth answers to compute the fitness $J(F')$.

4.3.5 Replacement and Population Update

In the replacement step, EVOFORMAT updates the population by retaining high-performing rules for the next generation (Figure 1(E)). Let $\mathcal{F}^t \subseteq \mathcal{F}$ denote the population at generation t. Given fitness scores for rules in $\mathcal{F}^t$ and the offspring set $\mathcal{F}'$, EVOFORMAT forms the next population $\mathcal{F}^{t+1}$ by selecting the top M rules from $\mathcal{F}^t \cup \mathcal{F}'$, i.e., $\mathcal{F}^{t+1} \subseteq \mathcal{F}^t \cup \mathcal{F}'$. This is a $(\mu + \lambda)$ strategy, which preserves high-performing rules while allowing improved offspring to enter the population.

4.4 Computational Cost

We quantify the computational cost of EVOFORMAT by the number of LLM calls. For simplicity, we report nominal expected call counts, assuming that each rule proposal succeeds without retry. Let $|\mathcal{D}|$ be the number of evolution examples, M the population size, K the offspring size, and T the number of evolution steps. Evaluating the initial population and all offspring rules requires $|\mathcal{D}|(M + KT)$ task-inference calls. Rule generation adds $M + (3/4)KT$ expected calls, because initialization requires M calls and three of the four uniformly sampled variation operators use the LLM. Thus, the total nominal expected number of LLM calls is $|\mathcal{D}|(M + KT) + M + (3/4)KT$.

5 Experiments

5.1 Experimental Setup

Dataset. We evaluate EVOFORMAT on both synthetic and real graph topologies. For synthetic graphs, we use GraphQA (Fatemi et al., 2024), which comprises six graph-reasoning tasks. Using the authors' public implementation,[1] we generate 1,000 graph–query–answer triples for each task and split them evenly into 500 examples for evolution and 500 for evaluation. The synthetic graphs are sampled from the Erdős–Rényi (ER) model (Erdős and Rényi, 1959). For real graphs, we use IMDB-MULTI (Yanardag and Vishwanathan, 2015), a social-network benchmark included in TUDataset (Morris et al., 2020). From these graphs, we construct GraphQA-style structural QA examples for two tasks, edge count and node degree, with answers computed directly from the graph structure. Using a fixed stratified random split, we sample 500 graphs for evolution and 500 graphs for evaluation. All downstream queries are

[1] https://github.com/google-research/google-research/tree/master/graphqa

Dataset	Model	Task	Hand-crafted Best	Hand-crafted 2nd	EvoFormat-RandInit	EvoFormat
GraphQA	Llama3.1-8B	NC	0.960	0.880	0.996 ± 0.007	0.998 ± 0.002
		EC	0.324	0.254	0.379 ± 0.006	0.418 ± 0.035
		ND	0.692	0.422	0.788 ± 0.034	0.797 ± 0.018
		EE	0.836	0.818	0.895 ± 0.026	0.923 ± 0.015
		CN	0.510	0.282	0.602 ± 0.031	0.715 ± 0.001
		CyC	0.950	0.916	0.972 ± 0.005	0.968 ± 0.007
	Mistral-7B	NC	0.722	0.718	1.000 ± 0.000	0.999 ± 0.001
		EC	0.258	0.208	0.352 ± 0.002	0.367 ± 0.025
		ND	0.410	0.326	0.600 ± 0.026	0.616 ± 0.029
		EE	0.798	0.766	0.771 ± 0.010	0.807 ± 0.009
		CN	0.314	0.182	0.501 ± 0.031	0.550 ± 0.035
		CyC	0.870	0.836	0.914 ± 0.016	0.921 ± 0.010
	Qwen2.5-7B	NC	0.966	0.964	0.998 ± 0.003	0.997 ± 0.003
		EC	0.444	0.330	0.467 ± 0.005	0.473 ± 0.022
		ND	0.796	0.450	0.866 ± 0.005	0.872 ± 0.016
		EE	0.886	0.882	0.905 ± 0.001	0.896 ± 0.002
		CN	0.638	0.436	0.634 ± 0.018	0.641 ± 0.033
		CyC	0.954	0.942	0.970 ± 0.005	0.971 ± 0.001
IMDB-MULTI	Llama3.1-8B	EC	0.238	0.200	0.410 ± 0.080	0.457 ± 0.005
		ND	0.696	0.428	0.831 ± 0.008	0.844 ± 0.007
	Mistral-7B	EC	0.434	0.404	0.466 ± 0.012	0.471 ± 0.021
		ND	0.388	0.380	0.565 ± 0.044	0.631 ± 0.062
	Qwen2.5-7B	EC	0.524	0.492	0.488 ± 0.025	0.496 ± 0.023
		ND	0.862	0.562	0.923 ± 0.001	0.923 ± 0.002
	GPT-4o-mini	EC	0.650	0.574	0.645 ± 0.012	0.661 ± 0.007
		ND	0.966	0.634	0.949 ± 0.008	0.953 ± 0.005

Table 1: Main results on GraphQA and IMDB-MULTI-derived QA tasks. Learned methods report mean $\pm$ standard deviation over three seeds. Hand-crafted Best/2nd denote the top two predefined encodings. NC/EC/ND/EE/CN/CyC denote node count, edge count, node degree, edge existence, connected nodes, and cycle check.

evaluated zero-shot without task-specific prompt tuning.

LLMs. We evaluate EvoFormat on both open-weight and closed-source LLMs. For the open-weight models, we use the instruction-tuned checkpoints Llama3.1-8B (Grattafiori et al., 2024) (`meta-llama/Llama-3.1-8B-Instruct`), Qwen2.5-7B (Yang et al., 2024a) (`Qwen/Qwen2.5-7B-Instruct`), and Mistral-7B (Jiang et al., 2023) (`mistralai/Mistral-7B-Instruct-v0.3`), loaded directly from their Hugging Face repositories without additional fine-tuning. We also evaluate the closed-source GPT-4o-mini model through the OpenAI API, using the model identifier `gpt-4o-mini-2024-07-18`.[2] All models are evaluated in a zero-shot setting without task-specific fine-tuning.

Baselines. We compare EvoFormat against predefined hand-crafted encodings and a budget-matched random-initialization variant. For the hand-crafted baseline, we evaluate the nine predefined graph-to-text encoding configurations considered in GraphQA (Fatemi et al., 2024). For GraphQA, we use the authors' released implementation to generate the serialized inputs. For IMDB-MULTI, we retain the same serialization templates while extending the original fixed-size node-naming schemes to accommodate larger, variable-sized graphs: the adjacency and incident encodings continue to use numeric identifiers generated according to each graph's size, whereas the fixed name dictionaries used by the remaining encodings are replaced with dynamically generated generic indexed identifiers (e.g., `Person_001` and `Character_001`). For each model–task setting, we report the highest and second-highest evaluation accuracies among these encodings as Hand-crafted Best and Hand-crafted 2nd, respectively, providing oracle references over predefined manual formats.

We also include EvoFormat-RandInit, which uses the same population size, offspring budget, number of steps, and hence candidate-evaluation budget as EvoFormat. However, every offspring

[2] https://platform.openai.com/docs/models/gpt-4o-mini

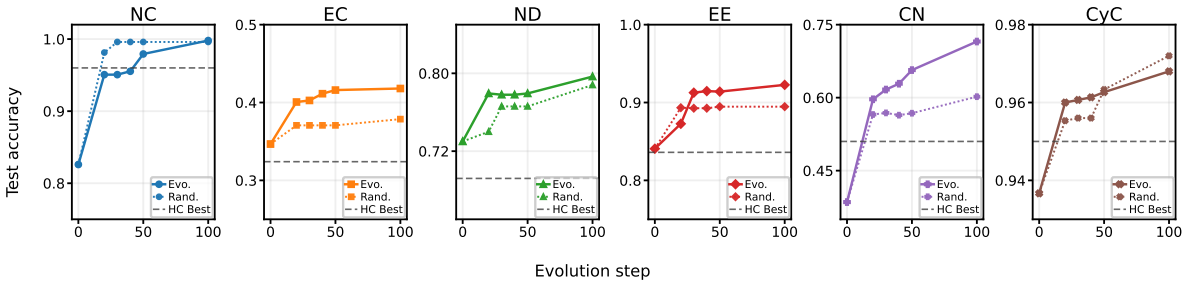


Figure 3: Effect of the number of evolution steps on GraphQA test accuracy with Llama3.1-8B. Evo. = EvoFormat, Rand. = EvoFormat-RandInit, and HC Best = best hand-crafted encoding. Points show means over three seeds.

is generated through fresh LLM-based rule initialization, independently of selected parents and previously generated rules; it uses neither parent-conditioned variation operators nor history-aware prompts. This baseline separates the contribution of the proposed rule space and LLM-based initialization from the additional contribution of evolutionary variation.

EvoFormat Configuration. Unless otherwise specified, EvoFormat is run with population size $M = 6$, offspring size $K = 2$, and $T = 100$ evolution steps. To account for stochasticity, each learned setting is repeated with three random seeds, and mean performance with standard deviation is reported.

Generation settings. We use temperature-zero decoding for downstream task prediction and nucleus sampling (temperature $= 0.8$, top-$p = 0.95$) for rule initialization and LLM-based variation. The maximum number of generated tokens is 4,096. If an operator attempt fails, EvoFormat resamples an operator, with at most three operator attempts per offspring; if all three attempts fail, the parent is retained unchanged. Each initialization or reinitialization allows at most 20 attempts in total if the LLM call fails, the response cannot be parsed as the required JSON object, or the resulting rule does not satisfy the required structural constraints. Each LLM-based rule operation uses three in-context exemplars to stabilize JSON-formatted outputs, with the NodeSet length in each exemplar set to the maximum number of nodes in the corresponding dataset.

5.2 Main Results

Comparison with hand-crafted encodings. Table 1 summarizes results on synthetic GraphQA and real-graph IMDB-MULTI. Across the 26 model–task settings, EvoFormat outperforms Hand-crafted Best in 24 settings and outperforms Hand-crafted 2nd in all settings. The large gaps between Hand-crafted Best and Hand-crafted 2nd in several tasks indicate that manual encoding selection is sensitive to both the task and the target model, making it difficult to choose a robust format a priori. These results suggest that EvoFormat can automate format design while remaining competitive with strong manually selected encodings.

Effect of evolutionary search. The results clarify the respective roles of the proposed rule space, LLM-based initialization, and evolutionary variation. EvoFormat-RandInit achieves a higher reported mean than the best hand-crafted baseline in 21 of the 26 model–task settings, indicating that the rule space and LLM-based initialization already produce strong candidates. Full EvoFormat achieves a higher reported mean than EvoFormat-RandInit in 21 of the 26 settings. However, the gains over EvoFormat-RandInit are not uniformly large, and EvoFormat does not achieve a higher reported mean in every setting. We therefore view evolutionary search not as uniformly superior to repeated initialization, but as a setting-dependent refinement process that can further improve strong candidates produced by initialization when task headroom remains.

Search dynamics. Figure 3 compares the performance of EvoFormat and EvoFormat-RandInit over search steps on GraphQA with Llama3.1-8B. On tasks with more remaining headroom, especially connected nodes, EvoFormat rises above EvoFormat-RandInit early and continues to improve with additional search. On near-saturated tasks such as node count and cycle check, both methods perform strongly from the early steps, leaving limited room for evolutionary variation to

Operator	# Updates	Share (%)	Mean ΔBest
Reinitialization	11	10.3	0.0582
Node-rule update	42	39.3	0.0154
Edge-rule update	31	29.0	0.0157
Arrangement update	23	21.5	0.0277

Table 2: Operator contribution to best-so-far improvements on GraphQA with Llama3.1-8B. Aggregated over six tasks and three seeds; Share is the percentage of best-update events and ΔBest denotes the fitness gain.

Task	Rule Type	Arrangement
NC	Sym. (3/3)	JSON, none, lex. (3/3)
EC	Sym. (2/3)	JSON, none, lex. (2/3)
ND	Nat. (2/3)	No majority; by node, lex.
EE	Sym. (3/3)	Markdown, by node, degree (2/3)
CN	Sym. (3/3)	JSON, by node, lex. (2/3)
CyC	Nat. (2/3)	JSON, none, lex. (2/3)

Table 3: Consistency of final learned rules on GraphQA with Llama3.1-8B. Each cell reports the dominant value and its agreement over three random seeds; Sym./Nat. denote symbolic/natural-language rules.

provide further gains. These trends suggest that evolutionary variation can provide additional refinement when repeated initialization does not rapidly saturate performance, while offering limited benefit when strong candidates are found early.

5.3 Analysis

Operator Contribution. We trace each best-so-far fitness improvement to the operator that generated it. Table 2 shows that improvements are distributed across operators rather than dominated by one. Node-rule and edge-rule updates frequently refine the format, while reinitialization yields fewer but larger gains on average. Arrangement updates also contribute, indicating that layout changes affect performance. Overall, these results suggest that EVOFORMAT's variation operators play complementary roles during search.

Learned Rule Characteristics. We inspect the final rules found by EVOFORMAT on GraphQA with Llama3.1-8B. Table 3 shows interpretable task-dependent patterns: NC, EC, EE, and CN often prefer symbolic identifiers, whereas ND and CyC more often use natural-language node descriptions. Lexicographic ordering dominates, except for EE's degree ordering. For example, a final CN rule assigns nodes simple identifiers such as `S_01`, `S_02`, ..., represents edges with `UNION(<NODE1> ∪ <NODE2>)`, and uses by-node grouping with lexicographic ordering. These results empirically support the motivation of EVOFORMAT: it discovers effective graph-to-text formats as explicit and interpretable rules, whose node naming, relation expression, and arrangement choices remain inspectable.

Task	Full Head	Constraint Only	Δ
NC	0.998 ± 0.002	0.917 ± 0.020	-0.081
EC	0.418 ± 0.035	0.371 ± 0.064	-0.047
ND	0.797 ± 0.018	0.725 ± 0.048	-0.072
EE	0.923 ± 0.015	0.959 ± 0.008	$+0.037$
CN	0.715 ± 0.001	0.673 ± 0.031	-0.043
CyC	0.968 ± 0.007	0.740 ± 0.232	-0.228

Table 4: Prompt-head robustness of learned EVOFORMAT rules on GraphQA with Llama3.1-8B. Results are mean ± standard deviation over three seeds; Δ is constraint-only minus full head.

Prompt-head Sensitivity. We further examine whether the learned graph encoder remains effective when the non-graph prompt component q is modified at inference time. Keeping the learned encoder g_F fixed, we replace the original prompt head, which contains both a general graph-analysis instruction and a task-specific answer constraint, with a shortened prompt head that retains only the task-specific answer constraint. Table 4 shows that performance generally decreases, confirming that q also affects downstream reasoning. Nevertheless, under the modified prompt, EVOFORMAT still exceeds the best hand-crafted result in Table 1 on four of six GraphQA tasks, and edge existence even improves. This suggests that the automatically discovered graph formats capture useful task-dependent structure, although final performance depends on both the graph encoder and the surrounding prompt.

6 Conclusion

We introduced EVOFORMAT, a black-box method for automatically optimizing graph-to-text formats for LLM-based graph reasoning. It represents formats as interpretable node, edge, and arrangement rules and searches the resulting rule space through LLM-based initialization and evolutionary variation, using downstream task performance as fitness. Across synthetic GraphQA and IMDB-MULTI-derived structural QA tasks, EVOFORMAT discovers diverse formats that generally outperform hand-crafted encodings across multiple LLMs. The rule space and LLM initialization yield strong candidates, with evolutionary search offering setting-dependent refinement when headroom remains.

Limitations

Scope of graph types and faithfulness. EvoFormat currently targets unattributed graph topology and does not yet handle richer graph settings, such as graphs with node or edge attributes, knowledge graphs with typed relations, or ontology-centric tasks involving schema constraints and symbolic ontological reasoning (Zhang et al., 2025). Extending the rule space to such settings would require explicit parameterization of how attributes and relation schemas are represented in text. Node rules would need to specify which attributes to include and how different attribute types, such as categorical, numerical, and free-text attributes, are represented or compressed. Edge rules would need to specify how edge attributes, relation directions, and relation types are represented, distinguish multiple relation types between the same entity pair, and encode schema information such as relation labels and constraints on the permissible source and target entity types. Arrangement rules would additionally need to determine how attribute descriptions and relation-specific neighborhoods are ordered, grouped, or omitted under context-length constraints.

These extensions would also produce longer and semantically denser graph-to-text inputs. Even if the serialization itself is deterministic, the downstream LLM may still confuse identifiers, mix relation or attribute evidence, or produce answers that are not fully grounded in the serialized graph. Beyond these faithfulness concerns, recent evidence from text-attributed graph benchmarks suggests that, when rich node semantics already provide strong task-relevant signals, explicit structural encodings may yield only marginal gains or even degrade LLM performance (Xu et al., 2025). Addressing these issues is an important direction for future work. Extending EvoFormat to richer graph settings will require rule spaces that decide which semantic and structural signals to include or omit and how to serialize, order, or compress the included information, together with faithfulness-oriented validation.

Search-time cost. EvoFormat trades additional search-time LLM calls for automatic format discovery. Under the nominal no-retry assumption, its cost scales as $|D|(M+KT)+M+\frac{3}{4}KT$ target-LLM calls, as discussed in Section 4.4, whereas predefined hand-crafted formats require no search. One direction for reducing this cost is to screen each newly generated candidate using a lower-cost proxy, such as a smaller LLM or a surrogate learned from previously evaluated candidates, and reserve full target-LLM evaluation for candidates predicted to be promising. Another direction is to initially evaluate candidates on smaller subsets of the evolution set and reevaluate only promising candidates on the full set. Finally, the search could be stopped when the best-so-far fitness does not improve for a predefined number of steps. These strategies could reduce target-LLM calls, although their effectiveness would depend on how reliably the proxy or reduced evaluation set preserves candidate rankings under full target-LLM evaluation.

Reproducibility under black-box LLM access. EvoFormat assumes black-box access to LLMs, and the discovered formats may be sensitive to factors outside the authors' control, such as API updates, model-version changes, decoding settings, and prompt templates. This creates a potential reproducibility risk over time, especially for closed-source LLMs.

Acknowledgments

This paper is based on results obtained from a project, JPNP22007, commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

AI assistants were used for drafting and language polishing, code-generation assistance, literature search, and brainstorming and refining research ideas. The authors reviewed and verified all AI-assisted outputs and retain full responsibility for the paper.

References

Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. 2024a. GraphWiz: An instruction-following language model for graph computational problems. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 353–364.

Runjin Chen, Tong Zhao, Ajay Kumar Jaiswal, Neil Shah, and Zhangyang Wang. 2024b. LLaGA: Large language and graph assistant. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 7809–7823. PMLR.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. 2022. RLPrompt: Optimizing discrete text prompts with reinforcement learning.

In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3369–3391.

Paul Erdős and Alfréd Rényi. 1959. On random graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2024. Talk like a graph: Encoding graphs for large language models. In *International Conference on Learning Representations*, pages 43909–43934.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2024. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, pages 13481–13544.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Jiayan Guo, Lun Du, Hengyu Liu, Mengyu Zhou, Xinyi He, and Shi Han. 2023. GPT4Graph: Can large language models understand graph structured data? An empirical evaluation and benchmarking. *arXiv preprint arXiv:2305.15066*.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2024. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations*, pages 34133–34156.

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

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. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825*.

Dayeon Ki and Marine Carpuat. 2025. Automatic input rewriting improves translation with large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 10829–10856.

Wooyoung Kim, Byungyoon Park, and Wooju Kim. 2025. Query-aware learnable graph pooling tokens as prompt for large language models. *arXiv preprint arXiv:2501.17549*.

Minchan Kwon, Gaeun Kim, Jongsuk Kim, Haeil Lee, and Junmo Kim. 2024. StablePrompt : Automatic prompt tuning using reinforcement learning for large language model. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9868–9884.

Wenwu Li, Xiangfeng Wang, Wenhao Li, and Bo Jin. 2025. A survey of automatic prompt engineering: An optimization perspective. *arXiv preprint arXiv:2502.11560*.

Wenxin Mao, Ruiqi Wang, Jiyu Guo, Jichuan Zeng, Cuiyun Gao, Peiyi Han, and Chuanyi Liu. 2024. Enhancing text-to-SQL parsing through question rewriting and execution-guided refinement. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2009–2024.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. 2020. TUDataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*.

Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. 2024. Let your graph do the talking: Encoding structured data for LLMs. *arXiv preprint arXiv:2402.05862*.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7957–7968.

Kiran Ramnath, Kang Zhou, Sheng Guan, Soumya Smruti Mishra, Xuan Qi, Zhengyuan Shen, Shuai Wang, Sangmin Woo, Sullam Jeoung, Yawei Wang, Haozhu Wang, Han Ding, Yuzhe Lu, Zhichao Xu, Yun Zhou, Balasubramaniam Srinivasan, Qiaojing Yan, Yueyan Chen, Haibo Ding, and 2 others. 2025. A systematic survey of automatic prompt optimization techniques. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 33078–33110.

Federico Ranaldi, Andrea Zugarini, Leonardo Ranaldi, and Fabio Massimo Zanzotto. 2025. Protoknowledge shapes behaviour of LLMs in downstream tasks: memorization and generalization with knowledge graphs. *arXiv preprint arXiv:2505.15501*.

Xubin Ren, Jiabin Tang, Dawei Yin, Nitesh Chawla, and Chao Huang. 2024. A survey of large language models for graphs. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6616–6626.

Yijun Tian, Huan Song, Zichen Wang, Haozhu Wang, Ziqing Hu, Fang Wang, Nitesh V Chawla, and Panpan Xu. 2024. Graph neural prompting with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19080–19088.

Jianing Wang, Junda Wu, Yupeng Hou, Yao Liu, Ming Gao, and Julian McAuley. 2024. InstructGraph: Boosting large language models via graph-centric instruction tuning and preference alignment. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13492–13510.

Xixi Wu, Yifei Shen, Fangzhou Ge, Caihua Shan, Yizhu Jiao, Xiangguo Sun, and Hong Cheng. 2025. When do LLMs help with node classification? A comprehensive analysis. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 67623–67649. PMLR.

Hanwei Xu, Yujun Chen, Yulun Du, Nan Shao, Wang Yanggang, Haiyu Li, and Zhilin Yang. 2022. GPS: Genetic prompt search for efficient few-shot learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8162–8171.

Haotian Xu, Yuning You, and Tengfei Ma. 2025. When structure doesn't help: LLMs do not read text-attributed graphs as effectively as we expected. In *The Fourth Learning on Graphs Conference*.

Christos Xypolopoulos, Guokan Shang, Xiao Fei, Giannis Nikolentzos, Hadi Abdine, Iakovos Evdaimon, Michail Chatzianastasis, Giorgos Stamou, and Michalis Vazirgiannis. 2024. Graph linearization methods for reasoning on graphs with large language models. *arXiv preprint arXiv:2410.19494*.

Pinar Yanardag and S.V.N. Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 23 others. 2024a. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024b. Large language models as optimizers. In *International Conference on Learning Representations*, pages 12028–12068.

Ruosong Ye, Caiqi Zhang, Runhui Wang, Shuyuan Xu, and Yongfeng Zhang. 2024. Language is all a graph needs. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 1955–1973.

Xiao Zhang, Huiyuan Lai, Qianru Meng, and Johan Bos. 2025. OntoURL: A benchmark for evaluating large language models on symbolic ontological understanding, reasoning and learning. *arXiv preprint arXiv:2505.11031*.

Jianan Zhao, Le Zhuo, Yikang Shen, Meng Qu, Kai Liu, Michael Bronstein, Zhaocheng Zhu, and Jian Tang. 2023. GraphText: Graph reasoning in text space. *arXiv preprint arXiv:2310.01089*.

Xi Zhu, Haochen Xue, Ziwei Zhao, Wujiang Xu, Jingyuan Huang, Minghao Guo, Qifan Wang, Kaixiong Zhou, Imran Razzak, and Yongfeng Zhang. 2025. LLM as GNN: Graph vocabulary learning for text-attributed graph foundation models. *arXiv preprint arXiv:2503.03313*.