LoReC: Rethinking Large Language Models for Graph Data Analysis

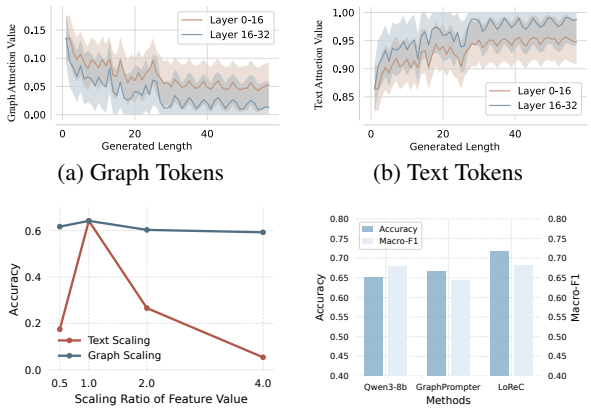
Hongyu Zhan [1] **Qixin Wang** [2] **Yusen Tan** [1] **Haitao Yu** [1] **Jingbo Zhou** [3] **Jia Li** [1] **Shuai Chen** [4] **Xiao Tan** [4]
Jun Xia [1 5 †]

Abstract

The advent of Large Language Models (LLMs) has fundamentally reshaped the way we interact with graphs, giving rise to a new paradigm called GraphLLM. As revealed in recent studies, graph learning can benefit from LLMs. However, we observe limited benefits when we directly utilize LLMs to make predictions for graph-related tasks within GraphLLM paradigm, which even yields suboptimal results compared to conventional GNN-based approaches. Through in-depth analysis, we find this failure can be attributed to LLMs' limited capability for processing graph data and their tendency to overlook graph information. To address this issue, we propose **LoReC** (**Lo**ok, **Re**member, and **C**ontrast), a novel plug-and-play method for GraphLLM paradigm, which enhances LLM's understanding of graph data through three stages: (1) **Look**: redistributing attention to graph; (2) **Remember**: re-injecting graph information into the Feed-Forward Network (FFN); (3) **Contrast**: rectifying the vanilla logits produced in the decoding process. Extensive experiments demonstrate that LoReC brings notable improvements over current GraphLLM methods and outperforms GNN-based approaches across diverse datasets. The implementation is available at https://github.com/Git-King-Zhan/LoReC.

1. Introduction

Recently, the integration of Graph Neural Networks (GNNs) with Large Language Models (LLMs) has catalyzed a new paradigm termed GraphLLM, empowering LLMs to understand complex graph data and conduct downstream graph tasks (Huang et al., 2024; Chen et al., 2024; He et al., 2025). In many cases, existing works have practically demonstrated that LLMs can effectively facilitate graph learning. For example, GraphGPT (Tang et al., 2024) aligns graph structural knowledge with LLMs for graph-related tasks, and Dr.E Liu et al. (2025) utilizes LLMs to comprehend graph data. Nevertheless, we observe minor improvements or even significant performance degradation when directly utilizing LLMs for predictions in graph-related tasks within GraphLLM paradigm. Previous work (Li et al., 2024) also observed unsatisfactory performance when utilizing LLMs to make predictions on graphs, but did not delve into this issue.



(a) Graph Tokens (b) Text Tokens (c) Scaling (d) Performance Comparison

Figure 1. (a-b) Attention distribution change between graph and text tokens across decoding layers as generation length increases. (c) Model performance under different scaling ratios applied to graph and text features. (d) Performance comparison of three approaches on the Citeseer dataset: text-only LLM baseline (Qwen3-8b), GraphPrompter, and LoReC-enhanced GraphPrompter.

To explain these phenomena, we conduct several experiments as shown in Figure 1. As depicted in Figure 1a and Figure 1b, attention to text tokens increases while attention to graph tokens decreases as more tokens are generated, impeding LLMs' understanding of graph data. This issue worsens in deeper layers, where over 90% of attention is allocated to text tokens. To investigate the respective contributions of graph-structured data and textual context to predictions, we proportionally scale text and graph feature values. As shown in Figure 1c, performance degradation from scaling text features is more pronounced than from scaling graph features, indicating that GraphLLM predic-

[1]The Hong Kong University of Science and Technology (Guangzhou) [2]Tongji University [3]Westlake University [4]Ant Group [5]The Hong Kong University of Science and Technology. Correspondence to: Jun Xia <junxia@hkust-gz.edu.cn>.

The Workshop on Graph Foundation Models at the 43rd International Conference on Machine Learning (ICML 2026), Seoul, South Korea. Copyright 2026 by the author(s).

tions are predominantly driven by textual features. Moreover, as demonstrated in Figure 1d, text-only LLM achieves performance comparable to GraphLLM model despite the absence of graph inputs, revealing that predictions are primarily driven by textual priors rather than graph inputs. Additional evidence across more datasets is provided in the Appendix A. As a consequence, performance degradation in GraphLLM models stems from two interrelated factors: *diluted attention allocated to graph tokens* and *overconfidence in textual priors*. Specifically, certain text tokens with minimal predictive utility receive disproportionately high attention weights during decoding, which intensifies as decoding proceeds stepwise. Furthermore, the higher information density inherent in graph-structured data impedes GraphLLM models from effectively comprehending graph inputs, causing these models excessively rely on textual priors. This naturally motivates the following question: *Can we enhance LLMs' understanding of graph data as well as mitigate their over-reliance on textual priors?*

To answer this question, we propose **LoReC** (**Lo**ok, **Re**member, and **C**ontrast), a novel training-free decoding framework for the GraphLLM models, inspired by the natural human cognition process: *when attempting to understand complex information, people instinctively redistribute attention to overlooked evidence, calibrate their memory, and rectify conflicting information before committing to an answer*. Following this principle, we employs a three-stage pipeline: (***i***) attention redistribution to ensure the model "**looks**" at graph tokens, (***ii***) memory calibration via graph information reinjection to ensure it "**remembers**" graph evidence, and (***iii***) dual-contrastive decoding to ensure it "**contrasts**" away false priors. The main contributions can be summarized as follows:

- We conduct comprehensive and in-depth investigation demonstrating why GraphLLM models exhibit suboptimal performance.
- We propose LoReC, a three-stage training-free decoding framework that enhances GraphLLM models' understanding of graph data.
- Extensive experiments across diverse datasets demonstrate that LoReC brings consistent and notable improvements over state-of-the-art GraphLLM methods and outperforms conventional GNN-based approaches.

2. Related Work

2.1. Large Language Models

The emergence of Large Language Models (LLMs) has revolutionized deep learning across multiple domains, demonstrating remarkable capabilities in understanding and generation. Leading models, notably ChatGPT (Achiam et al., 2023) and Gemini (Comanici et al., 2025), have pushed the frontier of artificial intelligence with advanced capabilities in multimodality, long-context processing and agentic reasoning. Concurrently, the open-source ecosystem has thrived, with powerful foundation models such as Llama (Grattafiori et al., 2024), Qwen (Yang et al., 2025) and Gemma (Team et al., 2025). A significant trend in this domain is the enhancement of complex reasoning. For instance, DeepSeek-R1 (Guo et al., 2025) demonstrates that reasoning capabilities can be effectively incentivized through reinforcement learning, complementing prompting strategies like Chain of Thought (Wei et al., 2022) and Tree of Thoughts (Yao et al., 2023). Underpinning these successes are foundational techniques such as In-Context Learning (Dong et al., 2024) for few-shot generalization and Reinforcement Learning from Human Feedback(RLHF) (Ouyang et al., 2022) for instruction alignment. Despite these remarkable strides in text processing and logical reasoning, LLMs exhibit inherent limitations when interpreting complex structural data. A recent benchmark (Dai et al., 2025) investigates how LLMs understand graph patterns and reveals that LLMs struggle to understand graph topologies and structural patterns. This critical gap has motivated the development of GraphLLM paradigm, which aims to equip LLMs with graph comprehension capabilities.

2.2. GraphLLM

Recently, the convergence of Graph Neural Networks (GNNs) and Large Language Models (LLMs) has established the nascent field of GraphLLM. This paradigm fundamentally transforms graph interaction by enabling the alignment between graph structures and natural languages. Initial efforts addressed the gap by projecting graph components into the LLM's token space via instruction tuning, as exemplified by GraphGPT (Tang et al., 2024) and LLaGA (Chen et al., 2024), soft prompting strategies like GNP (Tian et al., 2024) and GraphPrompter (Liu et al., 2024), or efficient adapter modules as seen in GraphAdapter (Huang et al., 2024). To achieve finer granularity and broader generalization, subsequent research introduced token-level quantization techniques, such as Dr.E Liu et al. (2025), and developed unified graph foundation models. Additionally, GOFA (Kong et al., 2024) interleaves GNN layers with LLMs, while Unigraph (He et al., 2025) leverages text-attributed graphs for cross-domain transfer. Furthermore, to address the context window limitations inherent in large-scale graphs, GraphChain (Wei et al., 2025) innovates by treating the LLM as an intelligent agent that performs analysis via logical tool chaining. However, benchmarks including GLBench (Li et al., 2024) and GraCoRe (Yuan et al., 2025) indicate that despite promising performance in semantic understanding and zero-shot scenarios, the misalignment between graph topology and text semantics often impedes

accuracy in downstream tasks.

3. Background

3.1. Preliminaries

Graph-Structure Data. A graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A}, \mathbf{X})$, where $\mathcal{V}$ is the set of N nodes and $\mathcal{E}$ represents the edges. The topological structure is encoded by an adjacency matrix $\mathbf{A} \in \{0,1\}^{N\times N}$, where $\mathbf{A}_{ij} = 1$ indicates a connection between nodes v_i and v_j. Each node v_i is associated with a feature vector $\mathbf{x}_i \in \mathbb{R}^F$, forming the feature matrix $\mathbf{X} \in \mathbb{R}^{N\times F}$.

Graph Neural Networks (GNNs). GNNs learn node representations by recursively aggregating information from local neighborhoods. This process, referred to as the *message-passing paradigm*, distinguishes graph representation learning from architectures designed for regular modalities such as images or text. The layer-wise update for a node v's embedding $\mathbf{h}_v^{(l)}$ is formally defined as:

$$\mathbf{h}_v^{(l+1)} = \psi^{(l)}\left(\mathbf{h}_v^{(l)}, \phi^{(l)}\left(\left\{\mathbf{h}_u^{(l)} : u \in \mathcal{N}(v)\right\}\right)\right), \quad (1)$$

where $\mathcal{N}(v)$ denotes the neighbors of v. Here, $\phi(\cdot)$ serves as the aggregation function (pooling messages from neighbors), and $\psi(\cdot)$ acts as the update function to fuse the aggregated context with the node's current state. After L layers, the final representations $\mathbf{h}^{(L)}$ are utilized for downstream tasks such as classification or link prediction.

Autoregressive Language Models. LLMs are typically instantiated as autoregressive transformer decoders trained via Causal Language Modeling (CLM). Formally, given a sequence of tokens $\mathbf{s} = (s_1, s_2, \ldots, s_R)$, the model estimates the joint probability $p(\mathbf{s})$ by decomposing it into a sequence of conditional probabilities:

$$p(\mathbf{s}) = \prod_{t=1}^{R} p(s_t \mid s_{<t}), \quad (2)$$

where $s_{<t}$ denotes the preceding context. The core mechanism driving this estimation is *masked self-attention*, which allows the model to attend to historical tokens while preventing information leakage from future tokens. For a specific attention head h, the output $\mathbf{O}_h$ is computed as:

$$\mathbf{O}_h = \text{softmax}\left(\frac{\mathbf{Q}_h\mathbf{K}_h^\top}{\sqrt{d_k}}\right)\mathbf{V}_h, \quad (3)$$

where $\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h$ serve as the query, key, and value matrices projected from the input embeddings, and d_k is the scaling factor. The final probability distribution over the vocabulary is obtained by applying a softmax function to the projected hidden states of the last layer.

3.2. Problem Formulation

Given a GraphLLM model $\mathcal{M}_\theta$ parameterized by θ, with a general architecture consisting of a text embedding layer, a graph encoder, a graph-text interface module, a text decoder with L transformer layers, and an affine layer $\varsigma(\cdot)$ that predicts the next-token distribution. For a graph-grounded task with textual query q and input graph $\mathcal{G}$, GraphLLM models first utilize the graph encoder $f_\mathcal{G}(\cdot)$ to transform the raw graph structure into a sequence of latent embeddings $\mathbf{Z}_\mathcal{G} = f_\mathcal{G}(\mathcal{G}) \in \mathbb{R}^{N\times d_g}$, where N is the number of nodes and d_g is the embedding dimension. The graph embeddings $\mathbf{Z}_\mathcal{G}$ are then projected into the LLM's word embedding space $\mathbb{R}^{d_{llm}}$ as $\mathbf{C}_\mathcal{G}$, which are concatenated with the textual instruction tokens $\mathbf{C}_\mathcal{T}$ and autoregressively decoded into the textual response $\mathbf{Y} = \{y_1, y_2, \ldots, y_{max}\}$:

$$p(\mathbf{Y} \mid \mathcal{G}, \mathbf{H}_\mathcal{T}) = \prod_{t=1}^{max} p(y_t \mid \mathbf{H}_\mathcal{G}, \mathbf{H}_\mathcal{T}, y_{<t}; \theta), \quad (4)$$

where $y_{<t}$ denotes previously generated tokens and θ represents the model parameters. This paradigm enables GraphLLM models to perform graph-related tasks.

4. Methodology

Building on the background outlined in Sec. 3, we propose **LoReC**, a framework containing three major components: (***i***) attention redistribution to ensure the model "**looks**" at graph tokens, (***ii***) Graph Re-injection via graph information reinjection to ensure it "**remembers**" graph evidence, and (***iii***) dual-contrastive decoding to ensure it "**contrasts**" away false priors. We present the framework overview in Figure 2 and all details are listed below.

4.1. ***Look***: Attention Redistribution

As demonstrated in Figure 1, GraphLLM models allocate progressively less attention to graph tokens in deeper layers as the generation process proceeds. This phenomenon causes models to overlook the graph tokens $\mathbf{C}_\mathcal{G}$, preventing them from effectively "looking" at the graph data. To counteract this graph attention dilution, we propose a mechanism dubbed **Attention Redistribution** that dynamically amplifies attention to graph tokens based on predictive uncertainty. The overall procedure is outlined in the Appendix D.

Uncertainty-Aware Trigger. We employ the Shannon entropy of the next-token distribution as a proxy for model uncertainty, which is computed by a vocabulary head $\varsigma(\cdot)$ on each layer during decoding (Chuang et al., 2024). Formally, let $P_\theta(i \mid \mathbf{x}_{<t}, \mathcal{G})$ denote the probability distribution over the top N candidates at time step t, conditioned on the preceding context $\mathbf{x}_{<t}$ and the graph input $\mathcal{G}$. The predictive

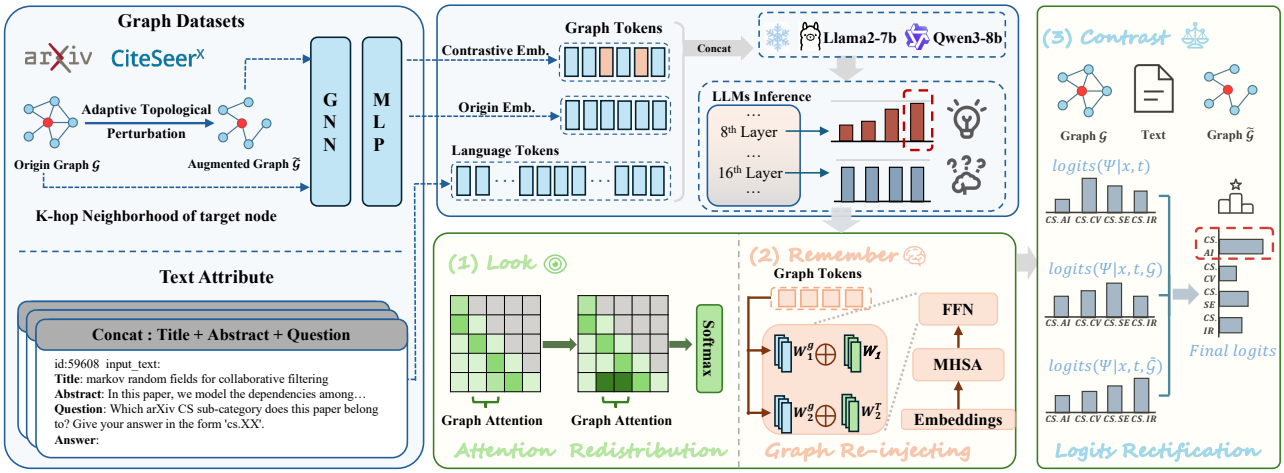


Figure 2. The overall framework of our proposed LoReC.

uncertainty $\mathcal{H}_t^{(l)}$ at layer l is calculated as:

$$\mathcal{H}_t^{(l)} = -\frac{1}{\log N} \sum_{i=1}^{N} P_\theta(i \mid \mathbf{x}_{<t}, \mathcal{G}) \log P_\theta(i \mid \mathbf{x}_{<t}, \mathcal{G}). \tag{5}$$

A low $\mathcal{H}_l$ suggests a sharply peaked distribution, indicating that the model is confident in its prediction. Conversely, a high $\mathcal{H}_l$ signifies a flat distribution, which suggests that the model exhibits high uncertainty regarding its prediction. In practice, we consider that uncertainty exceeding the predefined threshold γ at a given layer warrants intervention to recalibrate attention allocation.

Pay Attention to Graph. To mitigate GraphLLM models' tendency to overlook graph evidence, we intervene directly in the attention layers to enforce a "*Look*" operation. Unlike rigid masking or post-hoc probability scaling, our Attention Redistribution mechanism explicitly reallocates attention weights to graph tokens.

We first extract the graph token attention weights from the pre-softmax attention logits $\tilde{\mathbf{e}}_t$ for the currently generated token. When the model exhibits high uncertainty—as indicated by the layer-wise entropy $\mathcal{H}^{(l)}$ (defined in Eq. (5))—we adaptively rectify the attention distribution to allocate more weight to graph tokens. The rectified attention logits are formulated as follows:

$$\begin{aligned} \tilde{e}_{t,j} &= e_{t,j} + \Gamma(\mathcal{H}^{(l)} > \gamma) \cdot \eta \cdot |e_{t,j}|, \\ &\textit{subject to} \quad j \in \Omega_{\mathcal{G}}, \end{aligned} \tag{6}$$

where η is a tunable hyperparameter controlling the rectification strength, and $\Gamma(\cdot)$ denotes the gating function. The amplification is applied uniformly to all graph tokens $j \in \Omega_{\mathcal{G}}$. Since the model's final vocabulary probability distribution is derived directly from the last token's hidden state projection, we extract the last token's attention weights over the graph tokens by indexing $\tilde{e}_{t,j}$.

Following the attention intervention, we apply softmax function to redistribute attention weights across tokens when reassigning encoded hidden states. This procedure repeats autoregressively for each subsequent token prediction. Furthermore, this approach is mathematically principled as it is topology-preserving by amplifying attention proportionally to graph's intrinsic relevance.

4.2. *Remember*: Graph Re-injection

Simply directing the model's attention to graph tokens through the "*Look*" mechanism is not enough to ensure the model's understanding of graph data. To fully enhance graph comprehension, we must extend our intervention from the attentional surface to the parametric depth, ensuring the model can "*Remember*" graph information. Specifically, we propose **Graph Re-injection**, a mechanism that dynamically injects graph information into the FFN layers based on predictive uncertainty. The overall procedure is outlined in the Appendix D.

Memory Retrieval Formulation. The Feed-Forward Network (FFN) layers constitute approximately two-thirds of the parameters in Transformer-based architectures, serving as the primary repository for factual knowledge (Geva et al., 2021). Consider the input hidden state $x \in \mathbb{R}^d$ for a given layer, the vanilla FFN can be formulated as:

$$\text{FFN}(x) = \phi(xW_1)W_2, \tag{7}$$

where $\phi(\cdot)$ denotes the activation function (e.g., ReLU or SiLU), $W_1 \in \mathbb{R}^{d \times d_m}$ and $W_2 \in \mathbb{R}^{d_m \times d}$ represent the up-projection and down-projection weight matrices, respectively, with d_m being the intermediate dimension. Specifically, the weight matrices W_1 and W_2 can be explicitly

decomposed into sets of vectors as:

$$\mathbf{W_1} = [\mathbf{k}_1, \ldots, \mathbf{k}_{d_m}], \mathbf{W_2} = [\mathbf{v}_1, \ldots, \mathbf{v}_{d_m}]^\top, \quad (8)$$

where $k_i \in \mathbb{R}^d$ and $v_i \in \mathbb{R}^d$ are entries of key and value, respectively. As a consequence, FFN can be reformulated as a weighted aggregation of memory slots (Jie et al.):

$$\text{FFN}(x) = \sum_{i=1}^{d_m} \phi(\langle x, k_i \rangle) \cdot v_i, \quad (9)$$

where $m_i = \phi(\langle x, k_i \rangle)$ serves as the *memory coefficient*, quantifying the relevance of the i-th memory slot to the current input. Considering the formulation, FFN essentially performs a "soft retrieval" operation: the input x queries the parameter space to activate relevant patterns (k_i) and retrieves the associated knowledge (v_i).

Re-inject Graph Information. Motivated by the findings above, we intervene in the FFN layers to implement a "*Remember*" operation following the "*Look*" stage. Specifically, we propose a *Graph Re-injection* mechanism that treats graph tokens as keys and values (in Eq. (9)) and re-injects them into specific FFN layers based on uncertainty triggers.

Let $\mathbf{C}_\mathcal{G} = (\mathbf{g}_1, \ldots, \mathbf{g}_J) \in \mathbb{R}^{d \times J}$ denote the set of graph token embeddings, and let $x \in \mathbb{R}^d$ represent the current hidden state of the language model. We reinterpret the FFN as a key-value memory retrieval mechanism, where x serves as a query vector and graph tokens act as auxiliary memory slots encoding rich graph structural context. Once the model exhibits high unertainty (i.e., $\mathcal{H}^{(l)} > \gamma$, as defined in Eq. (5)), graph information is explicitly re-injected into selected FFN layers. The operation can be expressed as:

$$\Delta(\mathcal{G} \mid x) = \sum_{j=1}^{J} \phi\big(\langle x, \mathbf{g}_j \rangle\big)\, \mathbf{g}_j. \quad (10)$$

To integrate this information, we treat the retrieved graph information as supplementary evidence that operate concurrently with the original FFN output. The calibrated output, $\widehat{\text{FFN}}(x)$, is derived by fusing the vanilla FFN output with the graph-based correction term:

$$\widehat{\text{FFN}}(x) = \underbrace{(1-\alpha) \cdot \phi(x\mathbf{W}_1)\mathbf{W}_2}_{\text{Original Memory}} + \underbrace{\alpha \cdot \phi(x\mathbf{W}_1^g)\mathbf{W}_2^g}_{\text{Auxiliary Graph Memory}}. \quad (11)$$

Here, Original Memory represents the vanilla FFN output, while the Auxiliary Graph Memory represents the output of our Graph Re-injection Module (defined in Eq. (10)). The parameters $\mathbf{W}_1^g$ and $\mathbf{W}_2^g$ are derived from graph tokens and encode supplementary graph-related information in the hidden states. The scalar $\alpha \in [0, 1]$ controls the magnitude of the graph intervention.

Through *Attention Redistribution* that enables the model to "*Look*" at graph information and *Graph Re-injection* that enables it to "*Remember*", LoReC enhances the model's perception and understanding of the input graph structure.

4.3. *Contrast*: Logit Rectification

While the "*Look*" and "*Remember*" modules successfully enhance GraphLLM models' perception and understanding of graph data in latent space, the predictions may still inherit false priors. Standard decoding algorithms greedily select tokens with maximum probability, often polluted by the model's pre-training distribution rather than the specific context. In GraphLLM paradigm, this issue is further compounded by graph-specific prior biases originating from message-passing mechanisms and inductive biases inherent in graph encoders.

To eliminate these biases and enable models to yield less biased predictions, we rectify the models' vanilla logits, ensuring they can "*Contrast*" away false priors. Specifically, we introduce **Logit Rectification**, a mechanism that directly rectifies the final next-token distribution using the divergence between the vanilla output logits and two types of negative logits: text-only logit and augmented graph logit.

Adaptive Topological Perturbation. To mitigate the negative prior that stems primarily from the recursive aggregation of high-degree "hub nodes", which dominate the message-passing trajectory and enforce a rigid homophily prior, we require the augmented graph view $\tilde{\mathcal{G}}$ that preserves dominant hub structures while stochastically pruning sparse, peripheral connections that often contain fine-grained evidence. Naive uniform edge dropout is insufficient as it fails to account for the heterogeneous importance of nodes. Following (Zhu et al., 2021), we employ an adaptive augmentation strategy based on degree centrality to generate $\tilde{\mathcal{G}}$. Formally, let $\varphi(v) = \log(\deg(v) + \epsilon)$ denote the degree centrality of node v. The connectivity strength of edge $(u, v) \in \mathcal{E}$ is defined as s_{uv} and normalized to $\tilde{s}_{uv}$:

$$s_{uv} = \frac{\varphi(u) + \varphi(v)}{2}, \quad \tilde{s}_{uv} = \frac{s_{\max} - s_{uv}}{s_{\max} - s_{avg}}, \quad (12)$$

where $s_{\max}$ and s_{avg} denote the maximum and average strength values in the graph. We further introduce an overall edge drop rate μ and a truncation threshold τ (effectively preventing structural collapse). The final edge dropout distribution is formulated as:

$$w_{uv} = \min\left(\tau,\ \mu \cdot (1 - \tilde{s}_{uv})\right). \quad (13)$$

Note that the augmented graph logits (Ψ_{aug}) will be subtracted from the original logits (Eq. (14)). Therefore, the augmented graph view $\tilde{\mathcal{G}}$ should maximally preserve and amplify the graph inductive bias, so that the contrastive subtraction can effectively remove this bias.

Dual-Contrastive Decoding. To rectify text and graph biases in the original logits Ψ_{orig}, we obtain text-only logits

Ψ_{text} and perturbed-graph logits Ψ_{aug} that amplify these biases, respectively. Specifically, we calculate Ψ_{text} by masking all graph tokens before decoding, and calculate Ψ_{aug} use the adaptively augmented graph $\tilde{\mathcal{G}}$ as input. In practice, we use a gating function $\mathbb{I}_{\text{gate}}$ that prevents invalid perturbations (when no edges are dropped or when cutting edges would remove vital information from sparse graphs). The dual-contrastive decoding process is formulated as:

$$\Psi_{\text{final}} = \Psi_{\text{orig}} + \omega \underbrace{(\Psi_{\text{orig}} - \Psi_{\text{text}})}_{\Delta_{\text{Text de-biasing}}} + \beta \cdot \mathbb{I}_{\text{gate}} \underbrace{(\Psi_{\text{orig}} - \Psi_{\text{aug}})}_{\Delta_{\text{Graph de-biasing}}}, \tag{14}$$

where ω and β are hyperparameters controlling the suppression magnitude. $\Delta_{\text{Text de-biasing}}$ penalizes text bias in Ψ_{text} that deviates from ground truth, while $\Delta_{\text{Graph de-biasing}}$ penalizes graph bias inherent in Ψ_{aug}.

In practice, we define a dynamic candidate set $\mathcal{Z}_{\text{head}}$ that selectively contains only tokens with high probability under the unperturbed prediction distribution, thereby ensuring the coherence and semantic validity of the generated sequence. The complete formulation is as follows:

$$\begin{aligned} y_t \sim \text{Softmax}[&\Psi_{\text{orig}} + \omega \cdot (\Psi_{\text{orig}} - \Psi_{\text{text}}) \\ &+ \beta \cdot (\Psi_{\text{orig}} - \Psi_{\text{aug}}), \\ \textit{subject to} \quad & y_t \in \mathcal{Z}_{\text{head}}(y_{<t}). \end{aligned} \tag{15}$$

Overall, the three-stage LoReC framework, comprising "*Look*", "*Remember*" and "*Contrast*", empowers GraphLLMs with enhanced graph understanding and debiased prediction capabilities, similar to human cognitive processes. The computational costs analysis is provided in Appendix B, and the complete LoReC pseudo code is outlined in Appendix D.

5. Experiments

5.1. Experimental Setup

Datasets and Evaluation Protocols. To evaluate the performance of our proposed LoReC, we conduct experiments on five widely recognized graph datasets: PubMed (He et al., 2023), Cora (Kipf, 2016; Wen & Fang, 2023), Citeseer (Kipf, 2016), Ogbn-arxiv and Ogbn-products (Hu et al., 2020). The statistics of these datasets are summarized in Table 1. We randomly partition all datasets into training, validation, and testing sets with a ratio of 3:1:1. Note that for experiments in GraphGPT, we adhere to the its vanilla split setting with a training, validation, and testing ratio of 6:2:3. Performance is reported using Accuracy for balanced tasks and Micro-F1 for multi-label settings.

Baseline Models. We compare our proposed method against three distinct categories of baselines to ensure a comprehensive evaluation: (1) GNN-based models, specifically GCN (Kipf, 2016), GAT (Veličković et al., 2017), GKD (Yang et al., 2022), and GLNN (Zhang et al., 2022); (2) LLM-only approaches, including Baichuan-7b, Llama2-7b (Touvron et al., 2023), Vicuna-7b (Zheng et al., 2023) and Qwen3-8b (Yang et al., 2025) , which serve as semantic-centric baselines that rely solely on node text while ignoring graph topology; and (3) GraphLLM models, such as GraphGPT (Tang et al., 2024) and GraphPrompter (Liu et al., 2024) , representing state-of-the-art hybrid paradigm against which we benchmark our specific improvements.

Implementation Details. We utilize GraphGPT and GraphPrompter as base models. Following their original setups, we employ Vicuna-7B and Llama2-7B as backbones, respectively. Additionally, we evaluate both GraphPrompter and LoReC-GraphPrompter using a more recent and powerful backbone, Qwen3-8B. Experiments are conducted on 4 $\times$ H100 GPUs (80GB) for GraphGPT and 2 $\times$ A800 GPUs (80GB) for GraphPrompter.

Configuration. For our "*Look*" and "*Remember*" modules, we employ the Shannon entropy of the next-token distribution to trigger attention amplification and graph re-injection, with a shared entropy threshold of γ. Specifically, we apply attention amplification across layers 15-22 with an amplification factor of η. Graph re-injection is applied to layers 8-16 with a fusion ratio of α. For our "*Contrast*" module, we set the text-only contrast weight ω to 0.5 and the augmented graph contrast weight β to 1.0 by default. The adaptive plausibility constraint parameter is set to 1.0 to filter out unlikely tokens. For graph view generation, we employ a degree-based edge dropout strategy with a default rate of $\mu = 0.2$, which varies across datasets and selectively removes edges connected to low-degree nodes to create structurally valid contrastive views. The detailed parameter configurations can be found in the Appendix C.1.

Table 1. Statistics of the datasets used in our experiments.

DATASET	# NODES	# EDGES	# CLASSES
CORA	2,708	10,556	7
CITESEER	3,327	9,228	6
PUBMED	19,717	44,338	3
ARXIV	169,343	1,166,243	40
PRODUCTS	2,449,029	61,859,140	47

5.2. Comparison with State-of-the-art Results

We compare our proposed method against three categories of baselines: (1) GNN-based models (GCN, GAT, GKD, GLNN), (2) General LLMs (Llama2-7b, Vicuna-7b, Baichuan-7b, Qwen3-8b), and (3) GraphLLM models (GraphGPT, GraphPrompter). Note that LoReC is not trained from scratch but rather applied as a plug-and-play decoding method; thus, its performance depends on the

Table 2. Performance comparison based on GraphGPT. We highlight the performance of LoReC with gray background. The highest performances across different datasets are highlighted in **bold**. The results of GNN-based models and general LLMs are taken from published reports in GraphGPT.

Model	Arxiv-Arxiv		Arxiv-Pubmed		(Arxiv+Pubmed)-Arxiv		(Arxiv+Pubmed)-Cora	
	Accuracy	Macro-F1	Accuracy	Macro-F1	Accuracy	Macro-F1	Accuracy	Macro-F1
GNN-based Models								
GCN	0.5267	0.3202	0.3940	0.1884	0.0122	0.0008	0.0187	0.0032
GAT	0.5332	0.3118	0.3940	0.1884	0.1707	0.0285	0.0161	0.0057
GKD	0.5570	0.1595	0.3645	0.2561	0.2089	0.0179	0.0406	0.0037
GLNN	0.6088	0.3757	0.4298	0.3182	0.3373	0.1115	0.0182	0.0092
General LLMs								
Baichuan-7b	0.0946	0.0363	0.4642	0.3876	0.0946	0.0363	0.0405	0.0469
Vicuna-7b-v1.5	0.4962	0.1853	0.6351	0.5231	0.4962	0.1853	0.1489	0.1213
GraphLLM Models								
GraphGPT-std	0.6134	0.2607	0.6770	0.6283	0.6173	0.2594	0.1259	0.0773
GraphGPT-cot	0.5630	0.2483	0.5213	0.4816	0.6476	0.2854	0.1459	0.1287
Ours								
LoReC(GraphGPT-std)	**0.6312**	**0.2831**	**0.7018**	**0.6523**	0.6428	0.2753	0.1278	0.0785
LoReC(GraphGPT-cot)	0.5826	0.2720	0.5437	0.5175	**0.6734**	**0.3278**	**0.1643**	**0.1394**

Table 3. Performance comparison based on GraphPrompter. We highlight the performance of LoReC with gray background. The highest performances across different datasets are highlighted in **bold**.

Model	Cora		Citeseer		Arxiv		Products	
	Accuracy	Macro-F1	Accuracy	Macro-F1	Accuracy	Macro-F1	Accuracy	Macro-F1
General LLMs								
Llama2-7b	0.3764	0.3002	0.3846	0.4731	0.0909	0.0304	0.1117	0.0474
Qwen3-8b	0.6568	0.5647	0.6591	0.6786	0.6204	0.2162	0.1761	0.1461
GraphLLM Models								
GraphPrompter (Llama2-7b)	0.7915	0.7134	0.7029	0.6555	0.7008	0.4153	0.7720	0.3833
GraphPrompter (Qwen3-8b)	0.8026	0.7727	0.6652	0.6430	0.7331	0.3891	0.7422	0.3675
Ours								
LoReC (GraphPrompter-Llama2)	**0.8137**	**0.7931**	0.7119	0.6418	0.7094	0.4348	**0.7767**	**0.3854**
LoReC (GraphPrompter-Qwen3)	0.8026	0.7762	**0.7164**	**0.6804**	**0.7437**	**0.4674**	0.7459	0.3772

underlying GraphLLM models. We present performance comparisons on the benchmarks utilized by GraphGPT and GraphPrompter, with results summarized in Tables 2 and 3.

Overall Performance. As shown in Tables 2 and 3, our method consistently outperforms all baselines across all datasets. Note that following GraphGPT's setup, we evaluate our method in two settings: (1) supervised learning, where models are trained and tested on the same dataset (e.g., Arxiv), and (2) zero-shot transfer, where models trained on one dataset (e.g., Arxiv) are tested on another (e.g., PubMed). In Table 2, "-v1.5-" represents version of the base Vicuna model. "-std" and "cot" denote the use of the standard and generated COT instruction datasets, respectively. While state-of-the-art GraphLLM models (GraphGPT and GraphPrompter) integrate graph features, their standard decoding mechanism overlooks graph information and inherits false textual priors. In contrast, our method achieves accuracy improvements of up to 2.58% and 5.12% over GraphGPT and GraphPrompter, respectively, with macro-F1 gains of up to 4.24% and 7.97%, demonstrating its effectiveness in enhancing graph understanding.

5.3. Ablation Study

To validate the effectiveness of three modules and the hyperparameters we introduced in LoReC, we conducted in-depth ablation experiments as detailed below. All ablation experiments are conducted based on GraphGPT.

Effect of Individual Components. To validate our three-stage design, we conduct ablation studies by enabling "*Look*", "*Remember*", and "*Contrast*" modules individually and in pairs. As illustrated in Figure 3, each individual module provides performance gains over the baseline GraphLLM model, demonstrating its independent contribution. Pairwise combinations consistently outperform individual modules, indicating synergistic effects. However,

only the complete LoReC framework—integrating all three stages—achieves optimal performance, surpassing all partial configurations. These results validate our core design philosophy: "*Look*" strengthens model's perception to graph data at the attention layer, "*Remember*" extends the intervention from the attentional surface to the parametric depth by re-injecting graph evidence into FFN layers, and "*Contrast*" finally rectifies false priors through dual-contrastive decoding. The complementary nature of these three stages is essential for comprehensively enhancing GraphLLM models' understanding of graph data.

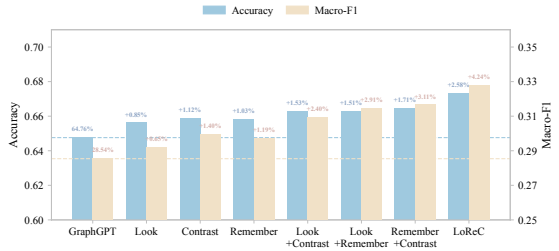


Figure 3. Impact of LoReC's components on the Arxiv dataset.

Effect of Amplification Factor and Injection Ratio. As shown in Figure 4, we analyze the impact of amplification factors η and injection ratios α on both accuracy and macro-F1 scores. For η, values between 0.05 and 0.45 consistently improve model performance, with the optimal setting observed at approximately 0.20. Similarly, injection ratio α in the moderate range of 15% to 35% yields positive effects, peaking at around 25%. Notably, performance degrades when α exceeds 35%, indicating that excessive graph information injection can be detrimental.

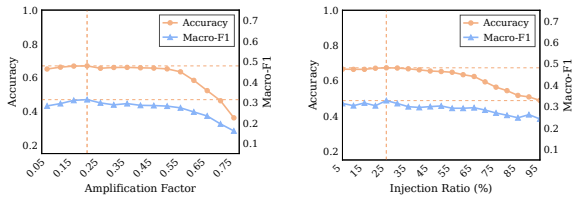


Figure 4. (Left) Results under different amplification factors η (Right) Results under different injection ratios α. Both evaluated on the Arxiv dataset with GraphGPT as the base model.

Effect of Dual-contrastive Decoding Strength. To identify optimal settings for the contrastive weights ω and β, we perform a grid search over the range [0, 10.0]. We observe that while moderate contrastive strength enhances performance, excessive values (> 1.0) can compromise the model's original inference capability by over-correcting the logits. Specifically, the contrastive divergence terms tend to overwhelm the final logit distribution, causing severe over-correction that distorts the model's original inference capability. More details can be found in the Appendix C.2.

As demonstrated in Figure 5, the optimal text-contrastive magnitude is approximately 0.5, while the optimal graph-contrastive magnitude is around 1.0.

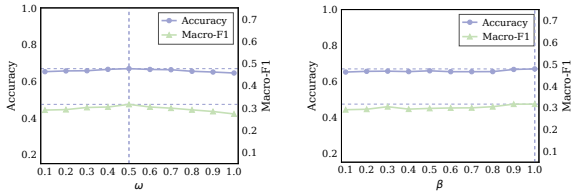


Figure 5. (Left) Results under different text-contrastive magnitude ω. (Right) Results under different graph-contrastive magnitude β. Both settings are evaluated on the Arxiv dataset with GraphGPT as the base model.

Effect of Entropy Threshold and Edge Threshold. Figure 6 demonstrates the impact of entropy threshold and edge threshold. For the entropy threshold, which determines when to activate the "*Look*" and "*Remember*" modules, exhibits optimal performance around 0.75 within an effective range of [0.65, 0.90]. The edge threshold serves as the gating function $\mathbb{I}_{\text{gate}}$ in Eq. (14) to determine whether graph-contrastive decoding should be applied, which achieves peak performance at approximately 10 edges. Understandably, a high edge threshold makes graph-contrastive decoding hard to trigger, while a low edge threshold would remove vital information from sparse graphs.

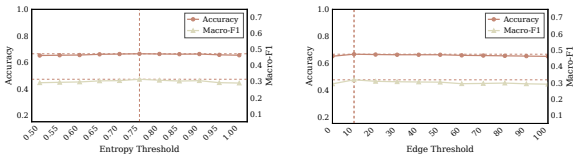


Figure 6. (Left) Results under different entropy threshold. (Right) Results under different edge threshold. Both evaluated on the Arxiv dataset with GraphGPT as the base model.

6. Conclusion

In this paper, we explain why LLMs struggle in GraphLLM paradigm and propose LoReC, a novel decoding method that comprehensively enhances LLMs' understanding of graph data through three stages: "*Look*", "*Remember*", and "*Contrast*". LoReC operates as a plug-and-play method requiring no additional fine-tuning, enabling seamless integration with existing GraphLLM models. Extensive experiments across multiple datasets demonstrate its effectiveness.

Acknowledgement

We sincerely thank the anonymous reviewers for their insightful comments and constructive suggestions. This research is supported by the National Natural Science Foundation of China Project (No. 623B2086), Sponsored by

CCF-GHFund (No. OF 2026005), Sponsored by CIPS-SMP-Zhipu Large Model Fund, Ant Group, and TeleAI of China Telecom.

Impact Statement

This work aims to improve LLMs' ability to understand graph-structured data, which has broad applications in social networks and molecular analysis. Our training-free approach reduces computational costs and makes advanced graph learning more trustworthy. While applicable to sensitive areas like social networks, our work focuses on methodological improvements rather than specific applications.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Chen, R., Zhao, T., JAISWAL, A. K., Shah, N., and Wang, Z. LLaGA: Large language and graph assistant. In *Forty-first International Conference on Machine Learning*, 2024.

Chuang, Y.-S., Xie, Y., Luo, H., Kim, Y., Glass, J. R., and He, P. Dola: Decoding by contrasting layers improves factuality in large language models. In *ICLR*, 2024.

Comanici, G., Bieber, E., Schaekermann, M., Pasupat, I., Sachdeva, N., Dhillon, I., Blistein, M., Ram, O., Zhang, D., Rosen, E., et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Dai, X., Qu, H., Shen, Y., Zhang, B., Wen, Q., Fan, W., Li, D., Tang, J., and Shan, C. How do large language models understand graph patterns? a benchmark for graph pattern comprehension. In *The Thirteenth International Conference on Learning Representations*, 2025.

Dong, Q., Li, L., Dai, D., Zheng, C., Ma, J., Li, R., Xia, H., Xu, J., Wu, Z., Chang, B., et al. A survey on in-context learning. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 1107–1128, 2024.

Geva, M., Schuster, R., Berant, J., and Levy, O. Transformer feed-forward layers are key-value memories. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Guo, D., Yang, D., Zhang, H., Song, J., Wang, P., Zhu, Q., Xu, R., Zhang, R., Ma, S., Bi, X., et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

He, X., Bresson, X., Laurent, T., Perold, A., LeCun, Y., and Hooi, B. Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning. *arXiv preprint arXiv:2305.19523*, 2023.

He, Y., Sui, Y., He, X., and Hooi, B. Unigraph: Learning a unified cross-domain foundation model for text-attributed graphs. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 448–459, 2025.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Huang, X., Han, K., Yang, Y., Bao, D., Tao, Q., Chai, Z., and Zhu, Q. Can gnn be good adapter for llms? In *Proceedings of the ACM Web Conference 2024*, pp. 893–904, 2024.

Jie, S., Tang, Y., Ding, N., Deng, Z.-H., Han, K., and Wang, Y. Memory-space visual prompting for efficient vision-language fine-tuning.

Kipf, T. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Kong, L., Feng, J., Liu, H., Huang, C., Huang, J., Chen, Y., and Zhang, M. Gofa: A generative one-for-all model for joint graph language modeling. *arXiv preprint arXiv:2407.09709*, 2024.

Li, Y., Wang, P., Zhu, X., Chen, A., Jiang, H., Cai, D., Chan, V. W., and Li, J. Glbench: A comprehensive benchmark for graph with large language models. *Advances in Neural Information Processing Systems*, 37:42349–42368, 2024.

Liu, Z., He, X., Tian, Y., and Chawla, N. V. Can we soft prompt llms for graph learning tasks? In *Companion Proceedings of the ACM Web Conference 2024*, pp. 481–484, 2024.

Liu, Z., Wu, L., He, M., Guan, Z., Zhao, H., and Feng, N. Multi-view empowered structural graph wordification for language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 24714–24722, 2025.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A.,

et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Tang, J., Yang, Y., Wei, W., Shi, L., Su, L., Cheng, S., Yin, D., and Huang, C. 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.

Team, G., Kamath, A., Ferret, J., Pathak, S., Vieillard, N., Merhej, R., Perrin, S., Matejovicova, T., Ramé, A., Rivière, M., et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Tian, Y., Song, H., Wang, Z., Wang, H., Hu, Z., Wang, F., Chawla, N. V., and Xu, P. Graph neural prompting with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19080–19088, 2024.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Wei, C., Hu, W., Hao, X., Wang, X., Yang, Y., Wang, Y., Tian, Y., and Chen, Y. Graphchain: Large language models for large-scale graph analysis via tool chaining. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Wen, Z. and Fang, Y. Augmenting low-resource text classification with graph-grounded pre-training and prompting. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 506–516, 2023.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Yang, C., Wu, Q., and Yan, J. Geometric knowledge distillation: Topology compression for graph neural networks. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Yuan, Z., Liu, M., Wang, H., and Qin, B. Gracore: Benchmarking graph comprehension and complex reasoning in large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 7925–7948, 2025.

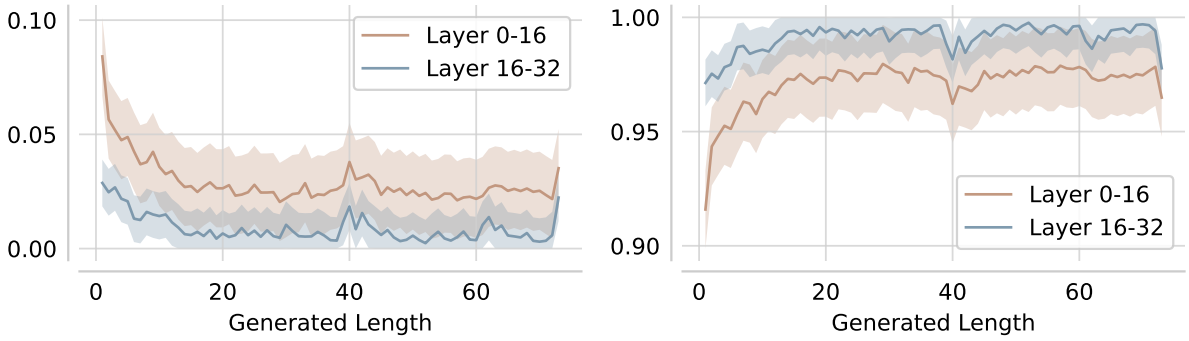
Zhang, S., Liu, Y., Sun, Y., and Shah, N. Graph-less neural networks: Teaching old MLPs new tricks via distillation. In *International Conference on Learning Representations*, 2022.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

Zhu, Y., Xu, Y., Yu, F., Liu, Q., Wu, S., and Wang, L. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*, pp. 2069–2080, 2021.

Appendix of LoReC

A. Attention Distribution Across Graph and Text Tokens on PubMed Dataset.



(a) Graph Tokens

(b) Text Tokens

Figure 7. Visualization of the attention distribution between graph and text tokens across decoding layers on the PubMed dataset. The results illustrate the dynamic evolution of attention patterns as the generation proceeds.

As illustrated in Figure 7, our experiments using GraphGPT on the PubMed dataset reveal that attention progressively shifts from graph tokens to text tokens as generation proceeds. This phenomenon explains the suboptimal performance of GraphLLM models: due to their autoregressive nature, LLMs tend to over-rely on textual features while neglecting graph information. Notably, this attention imbalance becomes more pronounced in deeper layers, confirming our hypothesis.

B. Computational Costs Analysis

LoReC operates "Look" and "Remember" dynamically based on uncertainty. Specifically, when uncertainty remains low across all layers, indicating high model confidence, these operations are not triggered. This mechanism efficiently enhances model's comprehension of graph data without extra computation. Similarly, for the "Contrast" operation, we employ a gate function $\mathbb{I}_{\text{gate}}$ to dynamically trigger graph-contrast logits calculation only when needed, thereby optimizing both computational time and memory consumption. Overall, while LoReC incurs a modest increase in time and memory costs, it achieves substantial performance improvements. The comparisons on latency, time cost, and memory are shown in Table 4.

Table 4. Performance comparison of different methods and LoReC in latency, time cost, memory usage, and accuracy.

Method	Latency (ms/token)	Memory Cost (GB))	Time Cost (s)	Accuracy (%)
LoReC	82.69	27.93	280.65	71.64
Qwen3-8b	13.67	18.22	259.15	65.91
GraphPrompter	45.23	18.43	201.11	66.52

C. Hyper-parameters Analysis

C.1. Complete Configurations

Following GraphGPT (Tang et al., 2024) and GraphPrompter (Liu et al., 2024), we adopt their reported parameter initializations and only optimize the hyperparameters specific to LoReC. For GraphGPT, the edge dropout probability μ ranges from 0.2 to 0.3 to maintain essential graph structural integrity. Additionally, we adaptively change the edge threshold across different datasets to avoid removing vital information from sparse graphs. All detailed dataset-specific hyperparameter configurations are summarized in Table 5.

Table 5. Parameter settings for LoReC across different base models and datasets.

Base Model	Dataset	μ	τ	ω	β	η	α	Entropy threshold	Edge threshold
GraphGPT	Arxiv	0.2	0.7	0.5	1.0	0.2	0.25	0.75	10
	PubMed	0.2	0.7	1.0	1.0	0.1	0.1	0.75	10
	Cora	0.3	0.7	0.8	1.0	0.15	0.2	0.75	20
GraphPrompter	Cora	0.2	0.7	1.0	1.0	0.1	0.1	0.75	10
	Citeseer	0.2	0.7	1.0	1.0	0.1	0.1	0.75	10
	Arxiv	0.2	0.7	1.0	1.0	0.1	0.1	0.75	10
	Products	0.2	0.7	1.0	1.0	0.1	0.1	0.75	10

C.2. Supplementary Analysis of Dual-contrastive Decoding Strength.

As illustrated in Figure 8, setting either contrastive weight (ω and β) above 1.0 leads to performance degradation. In such cases, the contrastive divergence terms excessively dominate the final logit distribution, over-correcting the model's predictions and compromising its original inference capability. Therefore, we conduct a refined grid search over the range [0.1, 1.0] to determine optimal values, with results presented in Figure 5.

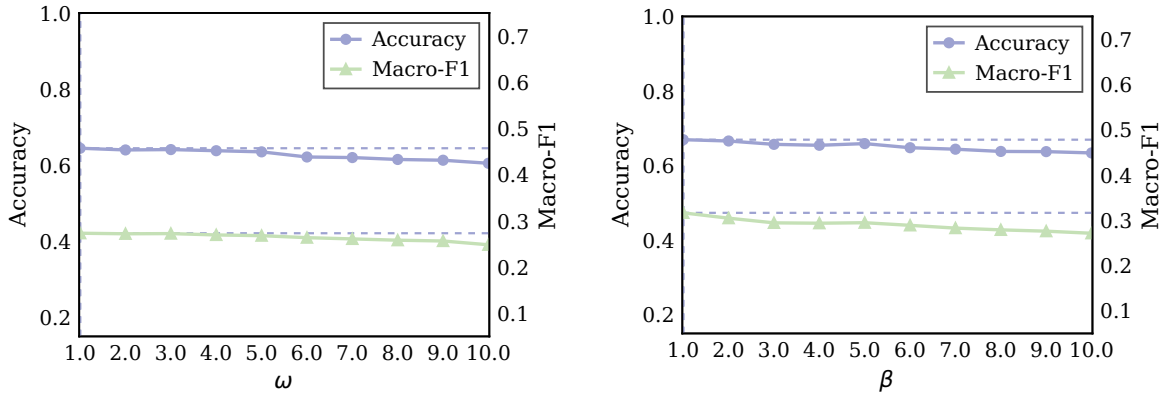


Figure 8. (Left) Results under different text-contrastive magnitude ω. (Right) Results under different graph-contrastive magnitude β. Both evaluated on the Arxiv dataset with GraphGPT as the base model.

D. Pseudo Codes of LoReC

We present the algorithm pipeline as follows. The equations mentioned in the algorithm can be seen in the main text.

Algorithm 1 *Look*: Attention Redistribution

Input: Graph tokens $\mathbf{C}_{\mathcal{G}}$, Text tokens $\mathbf{C}_{\mathcal{T}}$, Model $\mathcal{M}$.
At every decoding step t:
Initial set *trigger* = False.
for $l = 0$ **to** $L - 1$ **do**
  $\mathcal{H}_t^{(l)} = -\frac{1}{\log N} \sum_{i=1}^{N} p_\theta^{(l)} \log p_\theta^{(l)}$ (Eq. (5)).
  if $\mathcal{H}_t^{(l)} > \gamma$ **then**
    trigger = True.
  end if
end for
for $l = l + 1$ **to** $L - 1$ **do**
  Select graph attention logits $\mathbf{e}_j^{(l)}$ at layer l.
  Execute $\tilde{e}_j^{(l)} = e_j^{(l)} + \eta \, |e_j^{(l)}|$ (Eq. (6)).
  Recalculate attention weights $\tilde{\mathcal{A}}^{(l)} = \text{Softmax}(\tilde{\mathbf{e}}^{(l)})$.
end for
Output: Redistributed attention weights $\tilde{\mathcal{A}}$.

Algorithm 2 *Remember*: Graph Re-injection

Input: Input embedding $x^{(0)}$, Graph Tokens $\mathbf{C}_{\mathcal{G}}$.
Initial set *trigger* = False.
for $l = 0$ **to** $L - 1$ **do**
  $\mathbf{FFN}_{\text{base}} = \phi(\mathbf{x}^{(l)} \mathbf{W}_1^{(l)}) \mathbf{W}_2^{(l)}$;
  $\mathcal{H}_t^{(l)} = -\frac{1}{\log N} \sum_{i=1}^{N} p_{\theta,i}^{(l)} \log p_{\theta,i}^{(l)}$ (Eq. (5)).
  if $\mathcal{H}_t^{(l)} > \gamma$ and *trigger* == False **then**
    trigger = True.
    $\mathbf{W}_1^g \leftarrow \mathbf{C}_{\mathcal{G}}, \quad \mathbf{W}_2^g \leftarrow \mathbf{C}_{\mathcal{G}}^\top$;
    $\mathbf{Mem}_{\text{graph}} = \phi(\mathbf{x}^{(l)} \mathbf{W}_1^g) \mathbf{W}_2^g$;
    $\mathbf{FFN}_{\text{final}} = (1 - \alpha) \cdot \mathbf{FFN}_{\text{base}} + \alpha \cdot \mathbf{Mem}_{\text{graph}}$.
  else
    $\mathbf{FFN}_{\text{final}} = \mathbf{FFN}_{\text{base}}$.
  end if
end for
Output: Final hidden state $x^{(L)}$.

Algorithm 3 LoReC Strategy

Input: Graph tokens $\mathbf{C}_{\mathcal{G}}$, Text tokens $\mathbf{C}_{\mathcal{T}}$, Model $\mathcal{M}$.
Construct perturbed graph $\tilde{\mathcal{G}}$ via Adaptive Augmentation (Eq. (13)).
At every decoding step t:
for $l = 0$ **to** $L - 1$ **do**
  Compute Uncertainty $\mathcal{H}_t^{(l)}$ (Eq. (5)).
  if $\mathcal{H}_t^{(l)} > \gamma$ **then**
    Apply **Look**: Reallocate Attention (Eq. (6)).
    Apply **Remember**: Inject Graph (Eq. (11)).
  end if
  Calculate logits $\mathbf{v}_{\text{orig}} = \mathcal{M}(\mathcal{G}, \mathbf{x}_{<t})$.
  Calculate logits $\mathbf{v}_{\text{text}} = \mathcal{M}(\emptyset, \mathbf{x}_{<t})$.
  Calculate logits $\mathbf{v}_{\text{aug}} = \mathcal{M}(\tilde{\mathcal{G}}, \mathbf{x}_{<t})$.
  Apply **Contrast**: $\mathbf{v}_{\text{final}} \leftarrow \mathbf{v}_{\text{orig}}, \mathbf{v}_{\text{text}}, \mathbf{v}_{\text{aug}}$ (Eq. (14)).
  Sample $y_t \sim \text{Softmax}(\mathbf{v}_{\text{final}})$.
  $\mathbf{x}_{<t+1} \leftarrow \text{Concat}(\mathbf{x}_{<t}, y_t)$.
end for
Output: Model prediction y_t