
Supplementary Materials

Anonymous Author(s)

Affiliation

Address

email

1 A Introduction of GFlowNet

2 Generative Flow Networks (GFlowNets) [2] aim to learn a stochastic policy that generates objects
3 $x \in \mathcal{X}$ through sequential decisions, such that the marginal probability of generating x is proportional
4 to a reward function $R(x) > 0$. Given a complete trajectory $\tau = (s_0, a_1, s_1, \dots, a_T, s_T = x)$ that
5 terminates in object x , the forward trajectory probability is:

$$P_F(\tau) = \prod_{t=0}^{T-1} P_F(a_{t+1}|s_t)$$

6 and the backward probability (used to reverse the trajectory) is:

$$P_B(\tau) = \prod_{t=1}^T P_B(a_t|s_t)$$

7 **Trajectory Balance (TB) Loss [11].** The Trajectory Balance objective ensures the ratio of forward
8 to backward probability matches the reward:

$$\frac{P_F(\tau)}{P_B(\tau)} = \frac{R(x)}{Z} \iff \log P_F(\tau) - \log P_B(\tau) = \log R(x) - \log Z$$

9 where Z is the global partition function. The loss function is then defined as:

$$\mathcal{L}_{\text{TB}} = (\log P_F(\tau) - \log P_B(\tau) - \log R(x) + \log Z)^2$$

10 In practice, $\log Z$ is treated as a learnable scalar parameter.

11 **Subtrajectory Balance (SubTB) Loss [10].** To enable learning from partial trajectories, the
12 Subtrajectory Balance loss generalizes TB to arbitrary subpaths. For any subtrajectory $\tau_{i:j} =$
13 $(s_i, a_{i+1}, \dots, s_j)$ from state s_i to s_j , the balance condition becomes:

$$\frac{P_F(\tau_{i:j})}{P_B(\tau_{i:j})} = \frac{Z(s_j)}{Z(s_i)} \iff \log P_F(\tau_{i:j}) - \log P_B(\tau_{i:j}) = \log Z(s_j) - \log Z(s_i)$$

14 This leads to the Subtrajectory Balance loss:

$$\mathcal{L}_{\text{SubTB}} = (\log P_F(\tau_{i:j}) - \log P_B(\tau_{i:j}) - \log Z(s_j) + \log Z(s_i))^2$$

15 Here, $Z(s)$ denotes the flow or partition function at state s , typically parameterized by a neural
16 network as $F_\phi(s) = \log Z(s)$. SubTB enables more flexible and sample-efficient training, especially
17 for long-horizon generation tasks. However, directly implementing GFlowNet on KG-based RAG
18 faces several challenges. First, the objectives such as Trajectory balance and sub-trajectory balance
19 are computed on the whole trajectories, leading to computational burden in KG-based RAG where
20 entities are associated with long texts. Second, many states and transitions in KGs are less-valued and
21 not visited, making the traditional GFlowNet objective inefficient. Second, the discrete and symbolic
22 nature of KGs poses difficulty in defining state transitions and flow dynamics, especially when
23 integrating pretrained language models to interpret semantic relevance. These factors collectively
24 make it challenging to directly apply GFlowNet to KG-based retrieval without significant adaptations
25 in trajectory design, reward shaping, and exploration strategy.

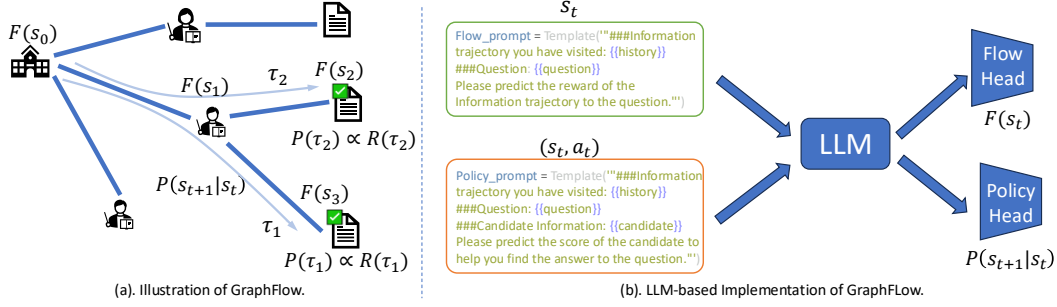


Figure 1: An overview of the proposed GraphFlow framework. (a). GraphFlow employs a flow estimator $F(\cdot)$ to factorize the outcome reward $R(\tau)$ of a retrieval trajectory τ to flow value $F(s_t)$. The flow value guides to learn a policy $P(s_{t+1}|s_t)$ that leads to accurate and diverse retrieval results for complex queries. (b) We introduce an LLM-based implementation of GraphFlow to enhance KG-based RAG on text-rich KGs.

26 B Implementation of GraphFlow with LLMs

27 **Model Architecture.** We use LLaMA3-8B-Instruct as the backbone LLM to implement GraphFlow.
 28 Specifically, we first employ the following flow prompt template to wrap the retrieval trajectory $\tau_{\leq t}$
 29 at state s_t into a text sequence for flow estimation.

###Information trajectory you have visited: {history}
 ###Question: {question}
 Please predict the reward of the Information trajectory to the question:

30
 31 Here {history} is the concatenation of documents of previously visited entities. {question} is the
 32 input complex query. The backbone LLM encodes the above wrapped text sequence. The embedding
 33 of the last token is treated as the representation of the wrapped sequence used for flow estimation. We
 34 employ a 1-layer MLP as the flow head, which receives the representation of the wrapped sequence
 35 and outputs the log value of the estimated flow $\log s_t$.

36 Then we employ the following policy prompt template to warp the retrieval trajectory $\tau_{\leq t}$ at state s_t
 37 into a text sequence for policy learning.

###Information trajectory you have visited: {history}
 ###Question: {question}
 ###Candidate Information: {candidate}
 Please predict the score of the candidate to help you find the answer to the question:

38
 39 Here {history} is the concatenation of documents of previously visited entities. {question} is the
 40 input complex query. And {candidate} is one action a_t that leads to the next state s_{t+1} . The backbone
 41 LLM encodes the above wrapped text sequence. The embedding of the last token is treated as the
 42 representation of the wrapped sequence used for learning $P(s_{t+1}|s_t)$. Specifically, we employ a
 43 1-layer MLP with a ReLU function to parameterize $\sigma_\theta(s_t, a_t)$. The forward policy $P(s_{t+1}|s_t)$ is
 44 calculated as below:

$$P(s_{t+1}|s_t) = \frac{\sigma_\theta(s_t, a_t)}{\sum_{a_t} \sigma_\theta(s_t, a_t)}. \quad (1)$$

45 **Training Configuration.** we apply LoRA [4] to inject learnable adapters into the frozen backbone
 46 of the LLM, and update the parameters of the flow head and the policy head. This design enables
 47 joint optimization of policy learning and flow estimation in a parameter-efficient manner, while also
 48 capturing rich contextual information through the LLM encoder. The parameters of these modules



Figure 2: Training dynamics of GraphFlow.

are trained by optimizing the detailed balance with local exploration (DBLE) objective:

$$\begin{aligned}
 \mathcal{L}_{\text{DBLE}}(s_t) &= \sum_{i=0}^k [\log F(s_t) - \log F(s'_{t+1,i}) + \log P(s_{t+1} = s'_{t+1,i} | s_t)]^2 \\
 &= \sum_{i=0}^k [\log F(s_t) - \log F(s'_{t+1,i}) + r_{\theta}(s_t, a'_{t,i}) - \log \sum_{i=0}^k e^{r_{\theta}(s_t, a'_{t,i})}]^2.
 \end{aligned} \tag{2}$$

Experimental Settings. To facilitate training the LoRA module and the flow head and the policy head on the STaRK benchmark, we first collect training dataset consisting of transitions between states. For a given question Q with the set of ground truth retrieval entities V_T in the training set, we first identify the initial entity V_0 using vector similarity between the embedding of Q and V_0 . Then, we sample the trajectory $\tau_{\leq T} = V_0 \rightarrow \dots \rightarrow V_T$ starting from V_0 and ending at V_T . We collect the all the transitions between s_t to s_{t+1} in the example the trajectory $\tau_{\leq T} = V_0 \rightarrow \dots \rightarrow V_T$ to implement local exploration as introduced in Section 3.2 in the main paper. For every training step, we construct mini-batch of traditions between states to calculate the loss in Eq. 2. The training dynamic is shown in Figure 2. Here, training transition loss is calculated using the transition between non-terminal states. And training starting loss and training end loss are calculated using boundary condition $F(s_0) = F(s_T) = 0$. Training total loss and eval loss are calculated on all the transitions between states on the training and evaluation dataset. Eval policy accuracy is the accuracy of policy $P(s_{t+1} | s_t)$ on the evaluation dataset. We training GraphFlow on these dataset for one epoch, other important parameters are shown in Table 1.

C Implementations of Baselines

To the best of our knowledge, few KG-based RAG methods are implemented on the text-rich STaRK benchmark. Instead, many KG-based RAG methods employ simple KGQA datasets such as CWQ, WEBQSP. Thus, we choose representative retrieval-based and agent-based baselines with explicit retrieval results (i.e., the retrieved node index) on the STaRK benchmark [14]. We provide the implementation details of the used baseline methods as below.

Dense-Retriever is implemented with SentenceBERT [12] to encode both questions and the documents of KG nodes into dense embeddings and retrieve the documents with top vector similarity. We choose

Table 1: Parameters of GraphFlow training on STaRK benchmark.

	STaRK-AMAZON	STaRK-MAG	STaRK-PRIME
Accumulation steps		2	
alpha		16	
batch_size		1	
num_gpu		8	
depth_cutoff		6	
doc_cutoff		400	
eval_ratio		0.8	
eval_step		100	
lora_dropout		0.05	
lr		1.00E-05	
max_length		1024	
n_epochs		1	
num_exploration		4	
r		32	
window_size		3	

Table 2: We provide data statistics of STaRK. The statistics are from the STaRK benchmark [14].

	entity type	relation type	avg. degree	entities	relations	tokens
STaRK-AMAZON	4	5	18.2	1,035,542	9,443,802	592,067,882
STaRK-MAG	4	4	43.5	1,872,968	39,802,116	212,602,571
STaRK-PRIME	10	18	125.2	129,375	8,100,498	31,844,769

72 SentenceBERT as the text document to be consistent with prior works [3], where SentenceBERT is
 73 used to encode the text information in KGs. Although STaRK benchmark provide the pre-processed
 74 text embedding of entities and relationships in KGs using text-embedding-ada-002 model, we find
 75 the inconsistency between the entities IDs and the entities embeddings. Some entities in KGs are
 76 not converted into embeddings. Thus, we rerun the encoding model using SentenceBERT to obtain
 77 the full entities embeddings. After encoding the text information into embeddings, we employ the
 78 vector similarity between the question embedding and text embeddings for retrieval. We evaluate the
 79 retrieval performance on top 20 retrieval results.

80 G-Retriever [3] is a two-stage method for KG-based RAG. It first employs the Prize-Collecting
 81 Steiner Tree (PCST) [1] algorithm to retrieve a subgraph from KGs relevant to the query. Then, the
 82 retrieved subgraph is encoded into the token space of LLM using a GNN for question answering (QA).
 83 To further improve the QA performance, G-Retriever also applies LoRA module to fine-tune LLM.
 84 Since we focus on the evaluating the retrieval performance of different KG-based RAG methods, we
 85 do not fine-tune the GNN and LLM for QA. To make PCST algorithm feasible on STaRK benchmark,
 86 we adopt a hybrid approach that first identify the 20 seed nodes and implement the PCST algorithm
 87 to extract the subgraphs around 2-hop ego graph around the seed nodes. We drop the seed nodes with
 88 dense neighborhoods to avoid computation overhead [5, 6].

89 SubgraphRAG [7] integrates a learnable subgraph retrieval module to retrieve from KGs. Since
 90 training the subgraph retrieval module on the STaRK benchmark is infeasible, we employ the ego-
 91 graph setting similar to G-Retriever. We identify the up-to 2 hop neighbor hood graph around the
 92 seed node to construct the training and testing set for SubgraphRAG. We also drop the the seed
 93 node that has dense neighborhood to avoid computation overhead. This ego-graph setting is also
 94 employed to construct the test set for the other KG-based RAG models. We follow the default setting
 95 of SubgraphRAG to reproduce it on STaRK benchmark.

96 ToG [13] employs an LLM agent to search from the KG to support KG-based question anwsering.
 97 ToG is implemented with frozen LLMs by prompt engineering instead of fine-tuning. Specifically,
 98 ToG employs tree-based search [16] to transverse the KG and search the relevant information for
 99 KG-based QA. Since we focus on evaluating the retrieval performance of KG-based RAG models,

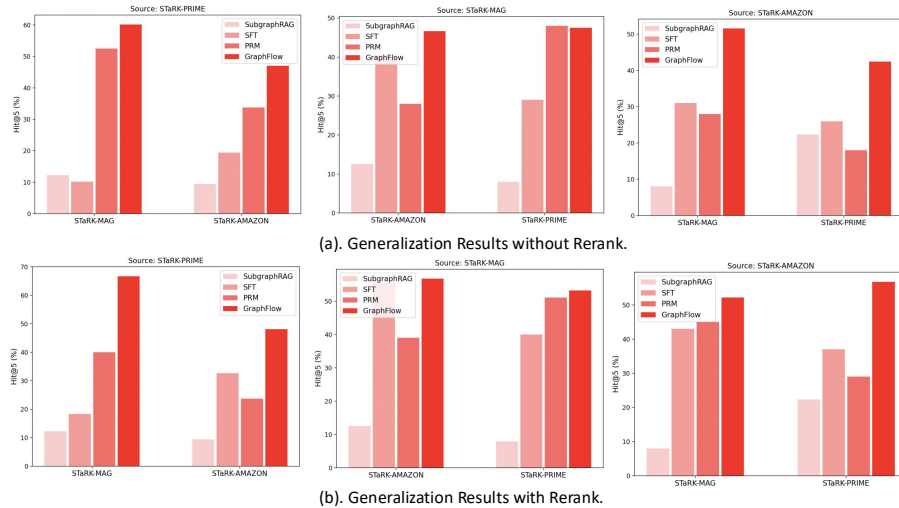


Figure 3: Generalization Performance (Hit@5) of KG-based RAG methods. GraphFlow shows superior cross-domain generalization performance, especially under the rerank setting (best viewed in color).

we modify ToG to retrieve the relevant document at each searching steps instead of incorporating the retrieved document to update the question answering results. Since running ToG on the whole KGs in STaRK is infeasible, we identify the seed node for ToG searching using vector similarity and constrain the searching area around the 2-hop neighborhood of the seed node. We instantiate ToG using both LLaMA3-8B and GPT-4o as backbone models, denoted as ToG+LLaMA3 and ToG+GPT4o, respectively.

We also implement SFT and PRM as two fine-tuning baselines build upon ToG and LLaMA3-8B-Instruct. We use the sample training dataset to train ToG using SFT and PRM as GraphFlow for a fair comparison. We employ the **TRL** (Transformer Reinforcement Learning) package to fir SFT and PRM fine-tuning. We apply LoRA funetuning to improve the efficiency.

Other potential baselines but hard to implement on STaRK. There are alternative KG-based RAG baseline methods for evaluation. However, we find it hard to implemented these baseline on STaRK, mostly due to the compatibility issues. We list some examples as below.

QAGNN [17] is designed for improving the QA performance on KG-based QA task. Although its retrieval performance is reported on STaRK benchmark, detailed implementation code on STaRK is not publicly available. Although recent concurrent work [6, 5] tried to implement QAGNN on STaRK, the reported performances of QAGNN diverge from the reported results on STaRK benchmark.

RoG [8] adopts similar approach as it finetunes the LLM to search from KGs. It first employs an LLM to generate retrieval trajectories for the input queries and use the generated retrieval trajectories to construct a training dataset to fine-tune the retrieval agent by SFT. However, we find that LLM usually generate invalid retrieval trajectory, leading to low quality training datasets for SFT fine-tuning. Thus, we finetune the retrieval agent using the valid retrieval trajectories by SFT in the main paper.

ToG-2.0 [9] is a recently proposed method to retrieve from the structured database and unstructured database. The key to ToG-2.0 is to identify the topic entities for a given questions. However, the implementation of topic entity recognition is absent, making it difficult to reproduce ToG-2.0 on STaRK benchmark.

HybridRAG [5], **Mixture of RAG** [6], and **KAR** [15] are recent pre-prints on Arxiv focusing on retrieving from text-rich KGs. However, their codes are not available yet, making it difficult for us to reproduce these methods.

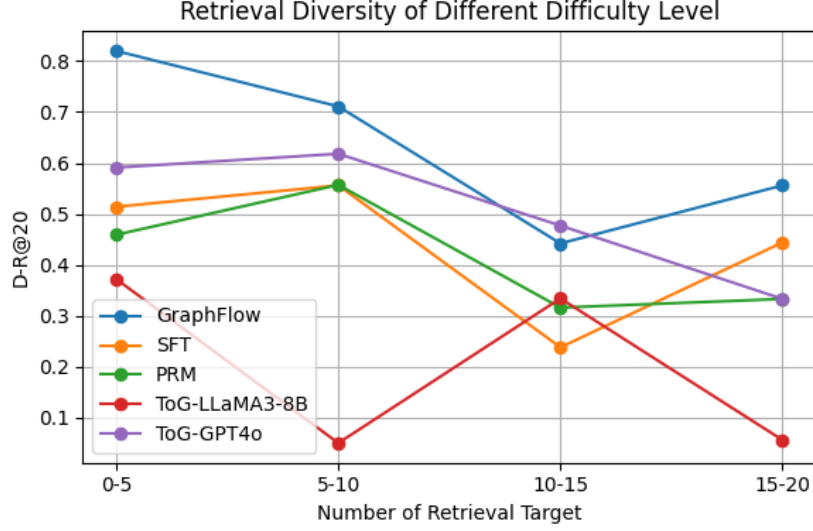


Figure 4: GraphFlow shows improved retrieval diversity on different difficulty levels of retrieval queries on STaRK-PRIME.

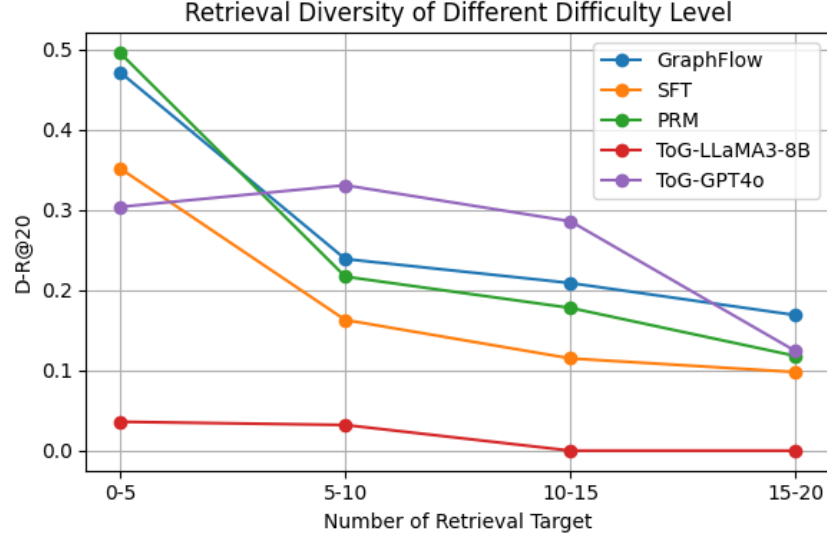


Figure 5: GraphFlow shows improved retrieval diversity on different difficulty levels of retrieval queries on STaRK-AMAZON.

129 D More results of Cross-domain Generalization

130 We show more generalization performance in terms of Hit@5 in Figure 3.

131 E More results of Hard Cases

132 We categorize the retrieval queries with different numbers of retrieval targets into 4 difficulty levels.
 133 We provide the performance of different KG-based RAG on STaRK-PRIME, STaRK-MAG, and
 134 STaRK-AMAZON at different difficulty levels. The results are shown in Figure 4, Figure 5, and
 135 Figure 6.

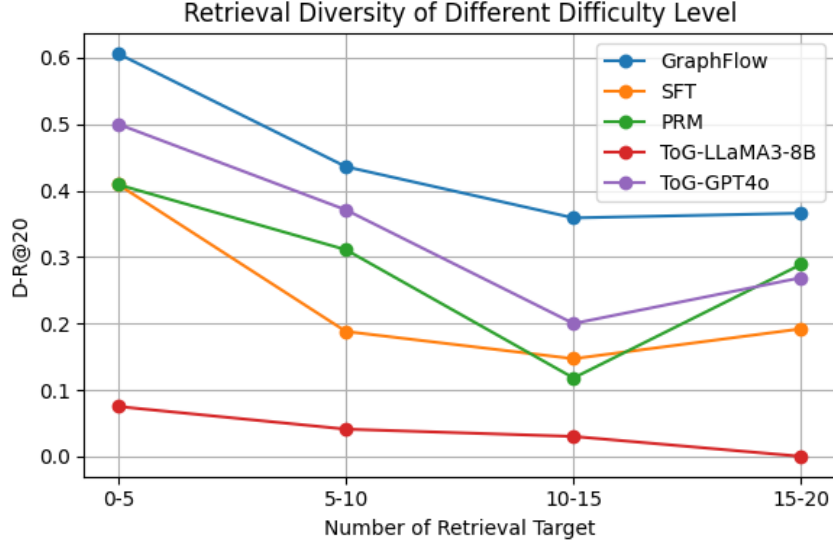


Figure 6: GraphFlow shows improved retrieval diversity on different difficulty levels of retrieval queries on STaRK-MAG.

F Benchmark Information

We provide benchmark information in Table 2.

G Computing Resources

We run all experiments on 8 NVIDIA-A800-SXM4-80GB GPUs and 56 Intel(R) Xeon(R) Platinum 8336C CPUs.

H Broader Impact and Limitations

GraphFlow introduces a novel framework for retrieval-augmented generation over text-rich knowledge graphs, enabling Large Language Models (LLMs) to reason more effectively through process supervision using GFlowNets. By modeling retrieval as a generative process that balances diverse and relevant paths, GraphFlow promotes both interpretability and coverage in knowledge-based reasoning. This has broad implications for applications such as scientific discovery, open-domain question answering, and medical decision support, where combining structured knowledge with free-text reasoning is crucial. Moreover, GraphFlow can serve as a foundation for future research in integrating generative decision-making with symbolic structures, thereby pushing forward the synergy between LLMs and knowledge graphs. One potential limitation is that we only evaluate the generalization ability of GraphFlow on two new domains.

References

- [1] Aaron Archer, MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Howard Karloff. Improved approximation algorithms for prize-collecting steiner tree and tsp. *SIAM journal on computing*, 40(2):309–332, 2011.
- [2] Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J Hu, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. *Journal of Machine Learning Research*, 24(210):1–55, 2023.
- [3] Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. G-retriever: Retrieval-augmented generation for textual graph

- understanding and question answering. *Advances in Neural Information Processing Systems*, 37:132876–132907, 2024.
- [4] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.
- [5] Meng-Chieh Lee, Qi Zhu, Costas Mavromatis, Zhen Han, Soji Adeshina, Vassilis N Ioannidis, Huzefa Rangwala, and Christos Faloutsos. Hybgrag: Hybrid retrieval-augmented generation on textual and relational knowledge bases. *arXiv preprint arXiv:2412.16311*, 2024.
- [6] Yongjia Lei, Haoyu Han, Ryan A Rossi, Franck Dernoncourt, Nedim Lipka, Mahantesh M Halappanavar, Jiliang Tang, and Yu Wang. Mixture of structural-and-textual retrieval over text-rich graph knowledge bases. *arXiv preprint arXiv:2502.20317*, 2025.
- [7] Mufei Li, Siqi Miao, and Pan Li. Retrieval or reasoning: The roles of graphs and large language models in efficient knowledge-graph-based retrieval-augmented generation. In *The Thirteenth International Conference on Learning Representations*, 2025.
- [8] LINHAO LUO, Yuan-Fang Li, Reza Haf, and Shirui Pan. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *The Twelfth International Conference on Learning Representations*, 2024.
- [9] Shengjie Ma, Chengjin Xu, Xuhui Jiang, Muzhi Li, Huaren Qu, Cehao Yang, Jiaxin Mao, and Jian Guo. Think-on-graph 2.0: Deep and faithful large language model reasoning with knowledge-guided retrieval augmented generation. *arXiv preprint arXiv:2407.10805*, 2025.
- [10] Kanika Madan, Jarrod Rector-Brooks, Maksym Korablyov, Emmanuel Bengio, Moksh Jain, Andrei Cristian Nica, Tom Bosc, Yoshua Bengio, and Nikolay Malkin. Learning gflownets from partial episodes for improved convergence and stability. In *International Conference on Machine Learning*, pages 23467–23483. PMLR, 2023.
- [11] Nikolay Malkin, Moksh Jain, Emmanuel Bengio, Chen Sun, and Yoshua Bengio. Trajectory balance: Improved credit assignment in gflownets. *Advances in Neural Information Processing Systems*, 35:5955–5967, 2022.
- [12] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, 2019.
- [13] Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *The Twelfth International Conference on Learning Representations*, 2024.
- [14] Shirley Wu, Shiyu Zhao, Michihiro Yasunaga, Kexin Huang, Kaidi Cao, Qian Huang, Vassilis Ioannidis, Karthik Subbian, James Y Zou, and Jure Leskovec. Stark: Benchmarking llm retrieval on textual and relational knowledge bases. *Advances in Neural Information Processing Systems*, 37:127129–127153, 2024.
- [15] Yu Xia, Junda Wu, Sungchul Kim, Tong Yu, Ryan A Rossi, Haoliang Wang, and Julian McAuley. Knowledge-aware query expansion with large language models for textual and relational retrieval. *arXiv preprint arXiv:2410.13765*, 2024.
- [16] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.
- [17] Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. Qa-gnn: Reasoning with language models and knowledge graphs for question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 535–546, 2021.