SMORE: Knowledge Graph Completion and Multi-hop Reasoning in Massive Knowledge Graphs

Hongyu Ren*†
hyren@cs.stanford.edu
Stanford University
Stanford, CA, USA

Hanjun Dai*
hadai@google.com
Google
Mountain View, CA, USA

Bo Dai
bodai@google.com
Google
Mountain View, CA, USA

Xinyun Chen
xinyun.chen@berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Denny Zhou
dennyzhou@google.com
Google
Bellevue, WA, USA

Jure Leskovec
jure@cs.stanford.edu
Stanford University
Stanford, CA, USA

Dale Schuurmans
schuurmans@google.com
Google / University of Alberta
Mountain View, CA, USA

ABSTRACT

Knowledge graphs (KGs) capture knowledge in the form of *head–relation–tail* triples and are a crucial component in many AI systems. There are two important reasoning tasks on KGs: (1) single-hop knowledge graph completion, which involves predicting individual links in the KG; and (2), multi-hop reasoning, where the goal is to predict which KG entities satisfy a given logical query. Embedding-based methods solve both tasks by first computing an embedding for each entity and relation, then using them to form predictions. However, existing scalable KG embedding frameworks only support single-hop knowledge graph completion and cannot be applied to the more challenging multi-hop reasoning task. Here we present *Scalable Multi-hOp REasoning (SMORE)*, the first general framework for both single-hop and multi-hop reasoning in KGs. Using a *single machine* SMORE can perform multi-hop reasoning in Freebase KG (86M entities, 338M edges), which is 1,500× larger than previously considered KGs. The key to SMORE's runtime performance is a novel bidirectional rejection sampling that achieves a square root reduction of the complexity of online training data generation. Furthermore, SMORE exploits asynchronous scheduling, overlapping CPU-based data sampling, GPU-based embedding computation, and frequent CPU–GPU IO. SMORE increases throughput (*i.e.*, training speed) over prior multi-hop KG frameworks by 2.2× with minimal GPU memory requirements (2GB for training 400-dim embeddings on 86M-node Freebase) and achieves near linear speed-up with the number of GPUs. Moreover, on the simpler single-hop knowledge graph completion task SMORE achieves comparable or even better runtime performance to state-of-the-art frameworks on both single GPU and multi-GPU settings.

ACM Reference Format:
Hongyu Ren, Hanjun Dai, Bo Dai, Xinyun Chen, Denny Zhou, Jure Leskovec, and Dale Schuurmans. 2022. SMORE: Knowledge Graph Completion and Multi-hop Reasoning in Massive Knowledge Graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, Washington, DC, USA, 11 pages. https://doi.org/10.1145/3534678.3539405

1 INTRODUCTION

A *knowledge graph* (KG) is a heterogeneous graph structure that captures knowledge encoded in a form of *head–relation–tail* triples, where the *head* and *tail* are two entities (*i.e.*, nodes) and the *relation* is an edge between them (*e.g.*, *(Paris, CapitalOf, France)*). Knowledge graphs form the backbone of many AI systems across a wide range of domains: recommender systems [27, 28], question answering [21, 23] and commonsense reasoning [12, 14].

Reasoning over such KGs consists of two types of tasks: (1) single-hop link prediction (also known as knowledge graph completion), where given a *head* and a *relation* the goal is to predict one or more *tail* entities. For example, given *TuringAward–Win–?* (*i.e.*, Who are the Turing Award winners?), the goal is to predict entities *GeoffHinton, DonKnuth*, etc.; And, (2) multi-hop reasoning, where one needs to predict (one or many) of the *tails* of a multi-hop logical query. For example, answering "Who are co-authors of Canadian Turing Award winners?" (Figure 1(A)). Finding answers to such query requires imputation and prediction of multiple edges across two parallel paths, while also using logical set operations (*e.g.*, intersection, union). Figure 1(B) shows the query computation plan and to determine the entities that are the answers to such a complex multi-hop query, missing links typically need to be implicitly inferred (Figure 1(C)). Notice that both tasks are closely related to each other. Knowledge graph completion can be viewed as a special case of a multi-hop reasoning task when the query

*Both authors contributed equally to this research.
†Work done during Hongyu Ren's internship at Google Brain.

KDD '22, August 14–18, 2022, Washington, DC, USA.

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539405

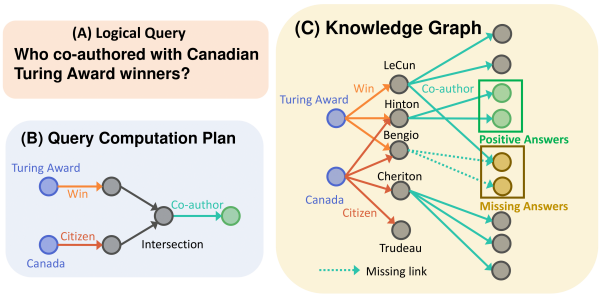

Figure 1: Query embedding methods aim to answer multi-hop logical queries (A) by avoiding explicit knowledge graph traversal and executing the query directly in the embedding space by following the query computation plan (B). Such methods are robust against missing links (C).

consists of a single relation (*e.g.*, 1-step path *TuringAward–Win–?* vs. more complex structure in Figure 1(B)). Multi-hop reasoning is a strict generalization of knowledge graph completion with much broader applicability but with its own set of unique computational and scalability challenges.

Currently there are no frameworks that support multi-hop reasoning on massive Knowledge graphs. For example, among many recent works on multi-hop reasoning [4, 5, 7, 10, 13, 19, 20, 22, 30] the largest KG used has only 63K entities and 592K relations. Moreover, while there are scalable frameworks for single-hop KG completion [11, 15, 31, 32], such frameworks cannot be directly used for multi-hop reasoning due to the more complex nature of the multi-hop reasoning task.

Scaling up embedding-based multi-hop KG reasoning methods is a critical need for many real-world AI applications and remains largely unexplored. Two significant challenges exist: (1) on the algorithmic side, given a massive KG (with hundreds of millions of entities), it is no longer feasible to materialize training instances, and training data needs to be efficiently sampled on the fly with a high throughput to ensure GPUs are fully utilized. And (2), on the system side, recent single-hop large-scale KG embedding frameworks are based on graph-partitioning [11, 15, 31, 32] which is problematic for multi-hop reasoning. Multi-hop reasoning requires traversing multiple relations in the graph, which will often span across multiple partitions.

To combat these challenges, we propose *Scalable Multi-hOp REasoning (SMORE)*, the *first general framework* for single- and multi-hop reasoning on massive KGs. SMORE performs algorithm-system co-optimization for scalability. **On the algorithmic side**, the key is to efficiently generate training examples online. To generate a training example with a set of positive and negative entities, we first instantiate a query on a given KG (Figure 2(B)) from a set of query logical structures (Figure 2(A)). The root of the instantiated query represents a known positive (answer) entity. To obtain a set of negative entities (non-answers), näive execution of the query computation plan (Figure 1(B)) using KG traversal (Figure 1(C)) to identify positive/negative entities has exponential complexity with respect to the number of hops of the query. Therefore, we propose a bidirectional rejection sampling approach to efficiently obtain high-quality negative entities for the instantiated queries. The key insight of the training data sampler is to identify the *optimal node cut* (red node in Figure 2(C)) of the computation plan via dynamic programming, then performing forward KG traversal (Figure 2(C)) as well as backward verification (Figure 2(D)) simultaneously, hence bidirectional rejection sampling. The nodes in the optimal cut cache the intermediate results from the forward KG traversal; for backward verification, we propose positive and negative candidate entities, traverse backward to the optimal cut and perform rejection sampling based on the overlap of the forward and backward sets. This reduces the worst case complexity by a square root, which makes it feasible to generate a training query, a positive answer entity and negative non-answer entities on the fly.

On the system side, SMORE operates on the full KG directly in a shared memory environment with multiple GPUs, while storing embedding parameters in the CPU memory to overcome the limited GPU memory. This design choice bypasses the potential drawbacks of graph partitioning for multi-hop reasoning in current KG embedding systems but also brings efficiency challenges. We design an asynchronous scheduler to maximize the throughput of GPU computation, via overlapping sampling, asynchronous embedding read/write, neural network feed-forward, and optimizer updates, as depicted in Figure 5. We obtain an efficient implementation that achieves near linear speed-up with respect to the number of GPUs.

We demonstrate the scalability of SMORE on three multi-hop reasoning algorithms (GQE [7], Q2B [19], BetaE [20]), six single-hop KG completion approaches (TransE [3], RotatE [24], DistMult [29], ComplEx [26], Q2B [19] and BetaE [20]) on six different benchmark KGs. The largest is the Freebase KG [2, 31] that contains 86M nodes and 338M edges, which is about $1,500\times$ larger than the largest KG previously considered for multi-hop reasoning. The new sampling technique improves the worst case runtime of enumerative search by 4 orders of magnitude. On small KGs, SMORE runs $2.2\times$ faster with 30.6% less GPU memory usage compared to the existing (non-scalable) multi-hop reasoning framework [18]. SMORE also supports link prediction. Here SMORE achieves comparable or even better efficiency with SOTA frameworks (which do not support multi-hop reasoning) on both single and multi-GPU settings.

SMORE can be deployed in a single-machine environment with a minimum requirement on the capacity of GPU memory. For example, it uses less than 2GB GPU memory when training a 400 dimensional embedding on the Freebase KG with 86M entities. SMORE's throughput scales nearly linearly with the number of GPUs. SMORE also provides an easy-to-use interface where implementing a new embedding model takes less than 50 lines of code. SMORE is open sourced at https://github.com/google-research/smore.

2 MULTI-HOP REASONING ON KG

A knowledge graph (KG) $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ consists of a set of nodes $\mathcal{V}$, edges $\mathcal{E}$ and relations $\mathcal{R}$. Each edge $e \in \mathcal{E}$ represents a triple (v_h, r, v_t) where $r \in \mathcal{R}$ and $v_h, v_t \in \mathcal{V}$. We are interested in performing multi-hop reasoning on KGs. Multi-hop reasoning queries include relation traversals as well as several logical operations including conjunction ($\wedge$), disjunction ($\vee$), existential quantification ($\exists$) and negation ($\neg$). Here we consider first order logical queries in their disjunctive normal form [20].

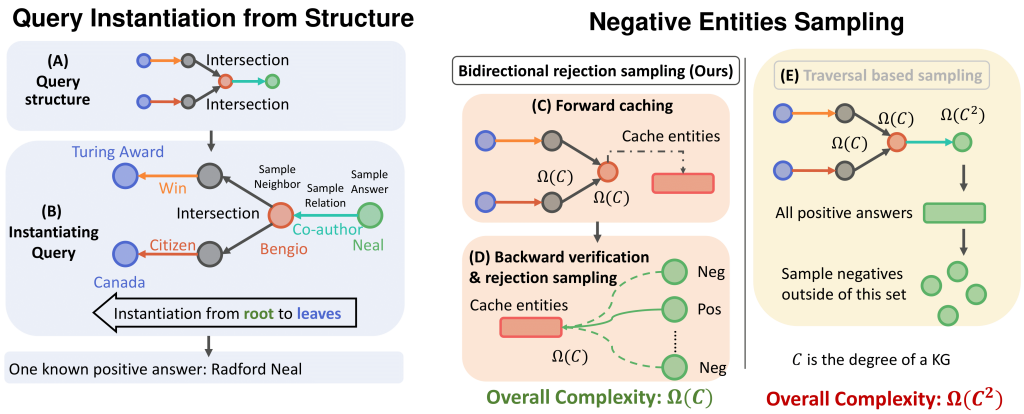

Figure 2: Overview of the sampling process for query and positive/negative answers. Our system instantiates queries using query structures from root to leaves. The entity in the root naturally becomes positive answer to the instantiated query. For negative entities, we propose bidirectional rejection sampling, which has a square root computation complexity compared to the traversal-based method.

Definition 1 (Logical queries [20]). *A first-order logical (FOL) query q consists of a non-variable anchor entity set $\mathcal{V}_q \subseteq \mathcal{V}$, existentially quantified bound variables $V_1, \ldots, V_k$ and a single target variable $V_?$ (answer). The disjunctive normal form of a query q is defined as follows:*

$$q[V_?] = V_? \,.\, \exists V_1, \ldots, V_k : c_1 \vee c_2 \vee \ldots \vee c_n$$

- *Each c represents a conjunction of one or more literals e. $c_i = e_{i1} \wedge e_{i2} \wedge \cdots \wedge e_{im}$.*
- *Each e represents an atomic formula or its negation. $e_{ij} = r(v_a, V)$ or $\neg\, r(v_a, V)$ or $r(V', V)$ or $\neg\, r(V', V)$, where $v_a \in \mathcal{V}_q$, $V \in \{V_?, V_1, \ldots, V_k\}$, $V' \in \{V_1, \ldots, V_k\}$, $V \neq V'$, $r \in \mathcal{R}$.*

Query computation plan. A query computation plan (Figure 1(B)) provides a plan for executing the query. The computation plan consists of nodes $\mathcal{V}_q \cup \{V_1, \ldots, V_k, V_?\}$, where each node corresponds to *a set of entities* on the KG. The edges in the computation plan represent a logical/relational transformation of this set, including relation projection, intersection, union and complement/negation. We adopt the same definition of computation plan as in [20] (details in Appendix A). By following the computation plan, we may either traverse the KG for answers or embed the given query. More details can be found in Appendix A.

Contrastive learning for KG embeddings. Given a data sampler $\mathcal{D}$ during training, each sample in $\mathcal{D}$ is a tuple $(q, \mathcal{A}_q^{\mathcal{G}}, \mathcal{N}_q^{\mathcal{G}})$, which represents a query q, its answer entities $\mathcal{A}_q^{\mathcal{G}} \subseteq \mathcal{V}$ and the negative samples $\mathcal{N}_q^{\mathcal{G}} \subseteq \overline{\mathcal{A}_q^{\mathcal{G}}}$. The contrastive loss Eqn (1) is designed to minimize the distance between the query embedding and its answers $\texttt{Dist}(f_\theta(q), f_\theta(v)), v \in \mathcal{A}_q^{\mathcal{G}}$ while maximizing the distance between the query embedding and the negative samples $\texttt{Dist}(f_\theta(q), f_\theta(v')), v' \in \mathcal{N}_q^{\mathcal{G}}$,

$$\mathcal{L}(\theta) = -\frac{1}{|\mathcal{A}|} \sum_{v \in \mathcal{A}_q^{\mathcal{G}}} \log \sigma\left(\gamma - \texttt{Dist}(f_\theta(q), f_\theta(v))\right) - \frac{1}{|\mathcal{N}|} \sum_{v' \in \mathcal{N}_q^{\mathcal{G}}} \log \sigma\left(\texttt{Dist}(f_\theta(q), f_\theta(v')) - \gamma\right), \quad (1)$$

where γ is a hyperparameter that defines the margin and σ is the sigmoid function.

We emphasize that due to the multi-hop structure in reasoning, identifying/computing $\mathcal{A}_q^{\mathcal{G}}$ and $\mathcal{N}_q^{\mathcal{G}}$ involves complex first-order logical operations, which are significantly more expensive than sampling in classical (single link) KG completion tasks, and thus is the bottleneck for scaling-up. To resolve this, we propose Scalable Multi-hOp REasoning (SMORE) to scale up single- and multi-hop KG reasoning methods (Tables 1 and 2), with an efficient sampling algorithm and parallel training, for a given contrastive loss. Next we first discuss our efficient training data sampling algorithm.

3 EFFICIENT TRAINING DATA SAMPLING

Unlike in link prediction, where sampling the training data, *head–relation–tail* can be quickly performed via dictionary look up [31, 32], sampling training data for multi-hop reasoning is much more complicated. It involves generating queries q by instantiating query structures, performing KG traversal to find answers $\mathcal{A}_q^{\mathcal{G}}$ as well as negative answers $\mathcal{N}_q^{\mathcal{G}}$, which is computationally expensive. We propose an efficient way to sample training data for contrastive learning for multi-hop reasoning. During inference, there are usually a pre-generated test set or user can input the test query of interest. This section focuses on efficient sampling during training.

3.1 Instantiating query structure

A query logical structure (Figure 3) specifies the backbone of a query q, including the types of operation (intersection, relation projection, negation, union) and its structure. It can be seen as an abstraction of the query computation plan, where anchor nodes and relation types are not grounded. Instantiating a query structure requires specifying a relation $r \in \mathcal{R}$ for each edge in the structure, as well as the anchor entities $\mathcal{V}_q$ (blue nodes in Figure 3).

A naïve way to instantiate a query (construct a concrete query given its logical structure) is to first ground the anchor entities by randomly sampling entities in the KG, and then randomly select relations $r \in \mathcal{R}$ for all the relation projection edges. However, in most cases such randomly generated queries have no answers in

Table 1: Three multi-hop KG reasoning models supported by SMORE.

Model	Embedding Space	Relation Projection	Intersection	Negation	Distance
GQE [7]	$\mathbf{q} \in \mathbb{R}^d, \mathbf{v} \in \mathbb{R}^d$	$\mathbf{q}+\mathbf{r}$	$\texttt{DeepSet}(\{\mathbf{q_i}\})$	-	$\|\mathbf{q}-\mathbf{v}\|$
Q2B [19]	$\mathbf{q} \in \mathbb{R}^{2d}, \mathbf{v} \in \mathbb{R}^d$	$\mathbf{q}+\mathbf{r}$	$\text{Cen}(q) = \sum_i \mathbf{a}_i \odot \text{Cen}(q_i)$ $\text{Off}(q) = \min(\{\text{Off}(q_i)\}) \odot \sigma(\texttt{DeepSet}(\{\text{Off}(q_i)\}))$	-	$\text{dist}_{\text{out}} + \alpha\text{dist}_{\text{in}}$
BetaE [20]	$\mathbf{q} \in \mathbb{R}^d, \mathbf{v} \in \mathbb{R}^d$	$\texttt{MLP}(\mathbf{q}, \mathbf{r})$	$\mathbf{q} = [(\sum w_i\alpha_i, \sum w_i\beta_i)]$	$\frac{1}{\mathbf{q}}$	$\texttt{KL}(\text{Beta}(\mathbf{v}); \text{Beta}(\mathbf{q}))$

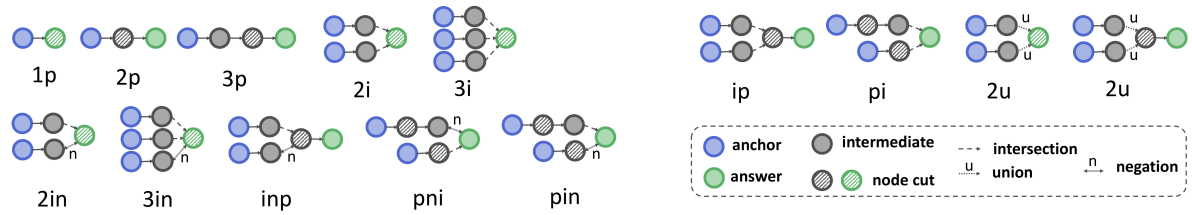

Figure 3: Different query structures and their optimal node cuts (shaded nodes) used by our bidirectional rejection sampling.

Table 2: Six single-hop models supported by SMORE.

Model	Embedding Space	Distance
TransE [3]	$\mathbf{h}, \mathbf{t} \in \mathbb{R}^d, \mathbf{r} \in \mathbb{R}^d$	$\|\mathbf{h}+\mathbf{r}-\mathbf{t}\|$
RotatE [24]	$\mathbf{h}, \mathbf{t} \in \mathbb{C}^d, \mathbf{r} \in \mathbb{C}^d$	$\|\mathbf{h} \circ \mathbf{r} - \mathbf{t}\|$
DistMult [29]	$\mathbf{h}, \mathbf{t} \in \mathbb{R}^d, \mathbf{r} \in \mathbb{R}^d$	$-<\mathbf{h} \circ \mathbf{r}, \mathbf{t}>$
ComplEx [26]	$\mathbf{h}, \mathbf{t} \in \mathbb{C}^d, \mathbf{r} \in \mathbb{C}^d$	$-\text{Re}(<\mathbf{h} \circ \mathbf{r}, \bar{\mathbf{t}}>)$
Q2B [19]	$\mathbf{h}, \mathbf{t} \in \mathbb{R}^d, \mathbf{r} \in \mathbb{R}^{2d}$	$\text{dist}_{\text{out}} + \alpha\text{dist}_{\text{in}}$
BetaE [20]	$\mathbf{h}, \mathbf{t} \in \mathbb{R}^d, \mathbf{r} \in \mathbb{R}^d$	$\texttt{KL}(\text{Beta}(\mathbf{t}); \text{Beta}(\texttt{MLP}(\mathbf{h}, \mathbf{r})))$

the KG as sampled entities may not even have relations of predetermined types, and intersections of random entities will almost always be empty. This means such samples have to be rejected and the sampling process has to start all over again. Such a naïve method leads to huge computation cost.

Instead, we instantiate a query structure via *reverse directional sampling*, *i.e.*, we first ground the root node (*i.e.*, the answer node) and then proceed towards the anchors. This is the reverse process of KG traversal for answers (Section 2). The main benefit of reverse sampling is that it can always instantiate a given query structure (Appendix C). Reverse directional sampling uses depth-first search (DFS) over the query structure from the root (answer) to the leaves (anchor entities). During the DFS, each node on the query structure is grounded to an entity on the KG and an edge to a relation on the KG associated with the previously grounded entity. An example of the process is shown in Figure 2(B).

Reverse sampling procedure can always obtain valid queries (with non-empty answer set). Another advantage of the above sampling process is that, the overall complexity is $O(C|q|)$, same as the complexity of DFS, where $|q|$ indicates the maximum depth of a path (from root to leaves) in the query structure, and C is the maximum degree of entities in the KG. The sampling process returns the instantiated query q, the anchor entities $\mathcal{V}_q$, and a single positive answer $a_q \in \mathcal{A}_q^{\mathcal{G}}$ (the instantiated entity at root).

3.2 Negative sampling

After the instantiating (node/edge grounding) the query structure, we obtain the tuple $(q, \mathcal{V}_q, \{a_q\})$ as a positive sample while we still need $\mathcal{N}_q^{\mathcal{G}}$ to optimize Eqn (1). We find that a single answer entity is sufficient in each step of stochastic training, while typically we need $k = |\mathcal{N}_q^{\mathcal{G}}|$ in Eqn (1) to be thousands for negative samples in the contrastive learning objective. Next we explain how to efficiently obtain a set of negative entities $\mathcal{N}_q^{\mathcal{G}}$ (*i.e.*, non-answers).

A naïve approach samples negative entities (non-answers) at random from the KG, independent of the query q. However, notice that a valid query may have many answers entities in the order of $O(C^{|q|})$. Such an approach implies that many of the sampled negatives are actually answers to the query, which would lead to noisy training data that confuses the model. An alternative would be to execute the query q and perform KG traversal to obtain all the answers $\mathcal{A}_q^{\mathcal{G}}$ (as presented in Section 2). We could then obtain negative samples $\mathcal{N}_q^{\mathcal{G}}$ by simply sampling from $\mathcal{V} \backslash \mathcal{A}_q^{\mathcal{G}}$. Although it is possible to re-order the relation projection operations to get better scheduling, in the worst case, $|\mathcal{A}_q^{\mathcal{G}}|$ is still in the order of $O(C^{|q|})$, regardless of the query scheduling. Thus, such exhaustive traversal is prohibitive for negative sampling on large KGs.

Our solution: Bidirectional rejection sampling. Since $\mathcal{N}_q^{\mathcal{G}}$ does not have to contain all the non-answer entities during stochastic training, we propose to exploit rejection sampling to locate a subset of negative entities efficiently. That is to say, starting with a random proposal $v \in \mathcal{V}$, we only need to check whether $v \in \mathcal{A}_q^{\mathcal{G}}$, rather than to enumerate the entire $\mathcal{A}_q^{\mathcal{G}}$. Inspired by bidirectional search, our key insight is to obtain a node cut (formally defined in Def. 2) on the query computation plan, *i.e.*, a subset of nodes that cut all the paths between each leaf node and the root node. Then we perform bidirectional search. We first start the traversal from the leaves (anchors) to the node cut and cache the entities we obtained in traversal, which we term *forward caching* (Figure 2(C)). We then sample negative entities, traverse from the root to the node cut and verify whether they are true negatives by checking the overlap of the cached entities and the traversed set. We term the second process *backward verification* (Figure 2(D)).

Definition 2 (Node cut). *A node cut c_q of a query q is a set of nodes in the computation plan, such that every path between anchor node (leaf) and answer node (root) contains exactly one node in c_q. A node cut is also minimal,* i.e., *no subset of c_q can be a node cut.*

We illustrate this idea in Figure 2. Given a two-hop query "*Who co-authored papers with Canadian Turing Award winners?*", we set the node after the intersection operation (red node) as the single

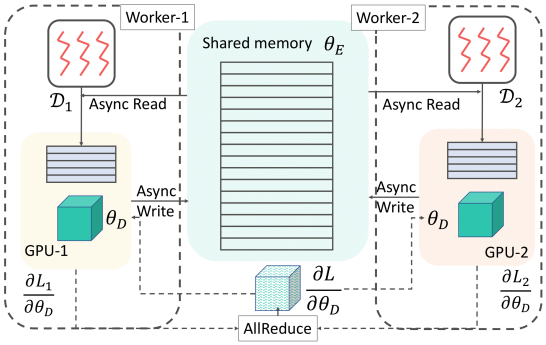

Figure 4: Overall training paradigm of SMORE.

node in the node cut. Then we can obtain the set of "Canadian Turing Award winners" via *forward traversal* and cache the intermediate results (*i.e.*, `Bengio`). Overall this process requires $O(C)$ computation/memory cost, where C is the degree of the KG. Then given a candidate negative entity v, one can spend $O(C)$ cost to verify whether the set of co-authors of v overlaps with the cached entities in the node cut. In our implementation we propose constant number of candidate negative entities, thus the overall computation cost would be $O(C)$, which is a reduction of square root from $O(C^2)$ using exhaustive traversal.

Next, we calculate the computation cost for any given node cut c_q, and then we propose an efficient algorithm to find the optimal node cut, *i.e.*, one with the lowest cost in bidirectional search.

Given a reasoning path $P_{(v_a,V_?)} = [v_0 = v_a, v_1, \ldots, v_t = V_?]$ in the query computation plan that starts from an anchor node (leaf) $v_a \in \mathcal{V}_q$ and ends at the answer node (root) $V_?$, for a node cut c_q, by definition there exists a unique node $v_i \in c_q \cap P_{(v_a,V_?)}$. Then the worst-case computation/memory cost for negative sampling for reasoning path $P_{(v_a,V_?)}$ can be estimated as $\texttt{cost}(c_q, P_{(v_a,V_?)}) = \max\left\{C^i, C^{t-i}\right\}$, *i.e.*, the maximum cost of forward traversal or backward verification. The optimal scheduling is recast as

$$\min_{c_q} \quad \max_{v_a \in \mathcal{V}_q} \texttt{cost}(c_q, P_{(v_a,V_?)}) \qquad \text{s.t. } c_q \text{ is a node cut of } q. \tag{2}$$

As the computation plan is a tree, we propose to solve the above optimization problem with dynamic programming (DP). This can be solved in a linear time w.r.t. $|q|$, and construct the node cut using the function $o(\cdot)$. We provide the details in Appendix C. Example query structures and their corresponding optimal node cuts are shown in Figure 3[1].

4 EFFICIENT TRAINING SYSTEM

SMORE is built for a shared memory environment with multi-cores and multi-GPUs. It combines the usage of CPU and GPU, where the dense matrix computations are deployed on GPUs, and the sampling operations are on CPUs.

4.1 Distributed training paradigm

Here we give a high-level introduction to the distributed training. Most of the KG embedding methods would maintain an embedding matrix $\theta_E \in \mathbb{R}^{|\mathcal{V}|\times d}$, where d is the embedding dimension which can typically be 512 or larger. For a large KG with more than millions of entities, the embeddings θ_E cannot be stored in GPUs, since most GPUs would have 16GB or lower memory. Thus similar as recent works [31], we put the embedding matrix on shared CPU memory, while putting a copy of other parameters $\theta_D = \theta \setminus \theta_E$, *e.g.*, neural logical operators, in each individual GPU.

We launch one worker process per GPU device. For simplicity, we use subscript w as an index of a worker. Worker w gets the shared access to θ_E and local GPU copy of dense parameters θ_D. Each worker repeats the following stages as shown in Figure 4:

(1) Collect a mini-batch of training samples $\{D_i\}_w$ from $\mathcal{D}_w$, which is the sampler.
(2) Load relevant entity embeddings from CPU to GPU;
(3) Compute gradients locally, and perform gradient `AllReduce` using $\frac{\partial \mathcal{L}_w}{\partial \theta_D}$. Update local copy θ_D.
(4) Update shared θ_E asynchronously with $\frac{\partial \mathcal{L}_w}{\partial \theta_E}$.

In the shared memory with multi-GPU scenario, the heavy CPU/GPU memory read/write with θ_E is necessary for every round of stochastic gradient update. This significantly lowers the FLOPS on GPU devices if we execute the above stages in a serialized way. So we design the asynchronous pipeline that is covered in Section 4.2.

The different storage location of parameters also brings different read/update mechanisms. For the embedding parameters θ_E, as only a tiny portion will be accessed during each iteration in stochastic training, the asynchronous update on the shared CPU memory would still result in a convergent behavior [16]. Unlike link prediction models, most multi-hop reasoning models are additionally equipped with dense neural logical operators, used in all batches and iterations. To minimize the loss of performance of multi-GPU training of these dense parameters, we choose to synchronously update θ_D with the `AllReduce` operation implemented with NVIDIA Collective Communication Library (NCCL).

4.2 Asynchronous design

In this section, we present the asynchronous mechanism for pipelining the stages in each stochastic gradient update. The stages can be virtually categorized into four kinds of *meta-threads*, where each kind of meta-thread may consist of multiple CPU threads or CUDA streams. These meta-threads run concurrently, with possible synchronization events for pending resources, as illustrated in Figure 5. Below we will elaborate each type of meta-thread individually.

Multi-thread sampler. Each worker w maintains one sampler $\mathcal{D}_w$ that has access to the shared KG. The sampler contains a thread pool for sampling queries and the corresponding positive/negative answers in parallel. The data sampler works concurrently with the other meta-threads. The pre-fetching mechanism will obtain samples for the next mini-batch while training happens using current batch on other threads. So if the sampler is efficient enough, then the runtime can almost be ignored.

Sparse embedding read/write. For the embedding matrix θ_E, we will create a single background thread with a CUDA stream for embedding read and write. Specifically, when loading the embedding of some entities into GPU, the background thread first loads that into a pinned memory area, then the CUDA asynchronous stream will perform pinned memory to GPU memory copy [6].

[1] Although query structures considered in current literature are small enough to find the optimal cut with brute force, our DP significantly improves efficiency when query structures are large.

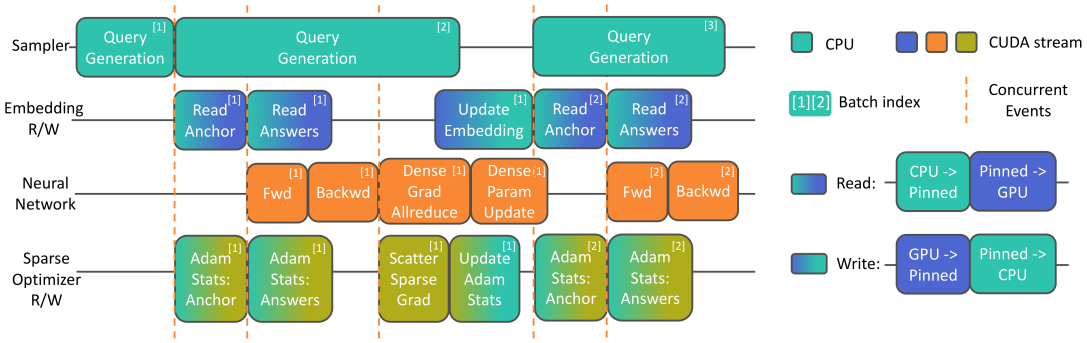

Figure 5: SMORE pipeline of a single worker process.

Table 3: KG statistics with number of entities, relations, training, validation and test edges.

Dataset	Entities	Relations	Training Edges	Validation Edges	Test Edges	Total Edges
FB400k	409,829	918	1,075,837	537,917	537,917	2,151,671
ogbl-wikikg2	2,500,604	535	16,109,182	429,456	598,543	17,137,181
Freebase	86,054,151	14,824	304,727,650	16,929,318	16,929,308	338,586,276

This read operator is non-blocking, and will not be synchronized until the CUDA operator in the main CUDA stream asks for it. The write operation works similarly but in the reverse direction. It is also possible to have multiple background threads. Right part of Figure 5 illustrates the idea.

Dense computation. The feed-forward of model f_θ starts when training data $(q, \mathcal{V}_q, \mathcal{A}_q^{\mathcal{G}}, \mathcal{N}_q^{\mathcal{G}})$ is ready and the embedding of the anchor entities $\mathcal{V}_q$ is fetched into GPU. The embeddings of $\mathcal{A}_q^{\mathcal{G}}$ and $\mathcal{N}_q^{\mathcal{G}}$ can be fetched as late as when we compute the loss function, in order to overlap the computation and memory copy. After obtaining the local gradients $\frac{\partial \mathcal{L}_w}{\partial \theta_E}$ and $\frac{\partial \mathcal{L}_w}{\partial \theta_D}$, the asynchronous update for θ_E will be invoked first without blocking, and at the same time the `AllReduce` operation will start, followed by the dense parameter update of θ_D on the GPU.

Sparse optimizer with asynchronous read/write. Different from θ_D, only a small set of rows of θ_E will be involved in each stochastic update. So we only keep track of $\theta_E^{\mathcal{V}_q}$, $\theta_E^{\mathcal{N}_q^{\mathcal{G}}}$, $\theta_E^{\mathcal{A}_q^{\mathcal{G}}}$ and their gradients, *i.e.*, the embeddings that are relevant to positive/negative and anchor entities. Once the back-propagation is finished, we will scatter $\frac{\partial \mathcal{L}_w}{\partial \theta_E^{\mathcal{V}_q}}$, $\frac{\partial \mathcal{L}_w}{\partial \theta_E^{\mathcal{N}_q^{\mathcal{G}}}}$ and $\frac{\partial \mathcal{L}_w}{\partial \theta_E^{\mathcal{A}_q^{\mathcal{G}}}}$ into a single continuous memory, due to the potential overlap among the sets $\mathcal{V}_q$, $\mathcal{N}_q^{\mathcal{G}}$ and $\mathcal{A}_q^{\mathcal{G}}$. We use Adam for SMORE, thus we need to additionally keep the first and second order moments of gradients in CPU. They are treated in the same way as θ_E, and thus will have the same asynchronous read/write behavior as discussed above. Every time a set of embeddings is retrieved from θ_E, the optimizer will also start to pre-fetch the corresponding first/second order moments in a different background thread ("Adam Stats" in Figure 5).

5 EXPERIMENTAL RESULTS

Here we evaluate SMORE on KG completion and multi-hop reasoning tasks on KG. The task is to answer multi-hop complex logical queries on KGs using various query embeddings. Besides the three small KGs used in prior works [19, 20], we propose a novel set of multi-hop reasoning benchmarks on three extremely large KGs with more than 86 million nodes and 338 million edges. We first demonstrate the scalability of SMORE on query sampling, multi-gpu speedup, GPU utilization over the three large KGs where all existing implementations fail. Additionally, we compare our single-hop KG completion runtime with state-of-the-art frameworks DGL-KE [31], Pytorch-Biggraph (PBG) [11] and Marius [15]. Then we evaluate the end-to-end training performance of SMORE on multi-hop reasoning over small as well as large KGs. This includes (1) calibration of the performance of three prior works GQE [7], Q2B [19] and BetaE [20] using SMORE framework on the three small KGs; (2) a thorough evaluation of the query embedding models on the three large KGs where prior implementation [18] is not applicable.

5.1 Experimental setup

Task setup. Following the standard experimental setup [19], given an incomplete KG, the goal is to train query embedding methods to discover missing answers of complex logical queries. Following the standard evaluation metrics, we adopt mean reciprocal rank (MRR) with the filtered setting as our metric, which is same across all previous works.

Datasets. We used the three datasets FB15k, FB15k-237, NELL from Ren and Leskovec [20]. These three KGs are small-scale with at most 60k entities. To create a set of large-scale multi-hop KG reasoning benchmarks, we further sample queries on three large KGs: FB400k, ogbl-wikikg2 and Freebase. Ogbl-wikikg2 is a KG from the Open Graph Benchmark [8]. FB400k is a subset of Freebase [2] which is derived based on a knowledge graph question answering dataset ComplexWebQuestion (CWQ) [25]. We further look at the complete Freebase KG used in DGL-KE. For FB15k, FB15k-237 and NELL, we directly take the validation and test queries from Ren and Leskovec [20]. For ogbl-wikikg2 and Freebase, we randomly sample validation and test queries using the official edge splits. We consider the same 14 query structures proposed in BetaE [20]. For FB400k, we directly take the SPARQL annotations of the validation and test questions in the CWQ as our validation and test queries.

Table 4: Performance of SMORE compared to KGReasoning package [18] on small KGs. Under the same model configurations and hyperparameters, SMORE achieves similar MRR (25.67 vs. 25.65) but with significant speed-up (2.2× faster training/throughput) and GPU memory saving (-30.6%).

Dataset	Model	MRR (%)		Training queries (x512/Sec)		GPU Memory (MB)	
		KGReasoning	SMORE	KGReasoning	SMORE	KGReasoning	SMORE
FB15k	BetaE	41.6	40.39	12.93	45.66	4022	1942
	Q2B	38.0	41.54	53.43	104.20	2146	1616
	GQE	28.0	30.60	55.67	118.05	2126	1778
FB15k-237	BetaE	20.9	19.67	12.92	45.46	4010	1876
	Q2B	20.1	20.42	55.82	107.05	2138	1582
	GQE	16.3	15.68	61.89	120.17	2116	1738
NELL995	BetaE	24.6	23.17	10.24	35.82	4852	3014
	Q2B	22.9	21.84	40.69	85.71	2406	2172
	GQE	18.6	17.53	41.32	68.81	3062	2922
Average		25.67	25.65	38.32	81.21 (+2.2×)	2986	2071 (-30.6%)

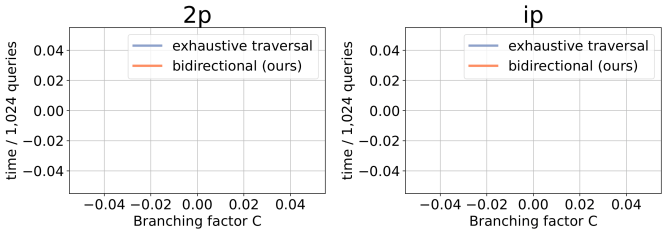

Figure 6: Speed-up of our bidirectional sampler over a naïve sampler that performs KG traversal, on 2p, ip and pni query structures (Figure 3). Our bidirectional sampling is significantly more efficient than the baseline.

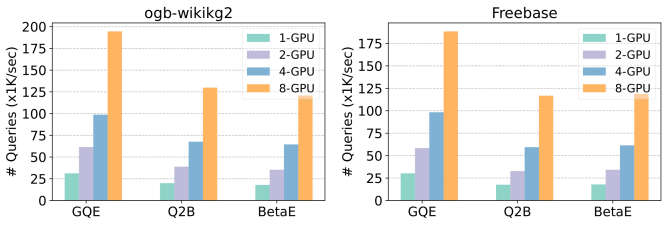

Figure 7: SMORE enjoys almost linear speed-up with respect to the number of GPUs, on both ogbl-wikikg2 and Freebase KGs, and across multiple embedding methods GQE, Q2B and BetaE.

Table 5: Runtime performance on Freebase KG. *Results taken from Mohoney et al. [15] with the same GPUs.

System	Epoch Time (s)			
	1-GPU	2-GPU	4-GPU	8-GPU
Marius [15]*	727	-	-	-
DGL-KE [31]*	-	1068	542	277
PBG [11]*	3060	1400	515	419
SMORE	760	411	224	121

Table 6: Speed and GPU memory of SMORE on the three large KGs with embedding dimension = 400 and per device batch size = 512. Our system design enables the (almost) graph-size agnostic speed and GPU memory usage.

Model	Dataset	Queries (x1K/Sec)	GPU Mem (MB)
BetaE	FB400k	123	1,872
	ogbl-wikikg2	121	1,860
	Freebase	118	2,014
Q2B	FB400k	135	1,796
	ogbl-wikikg2	130	1,776
	Freebase	116	2,382
GQE	FB400k	189	1,726
	ogbl-wikikg2	195	1,750
	Freebase	188	2,056

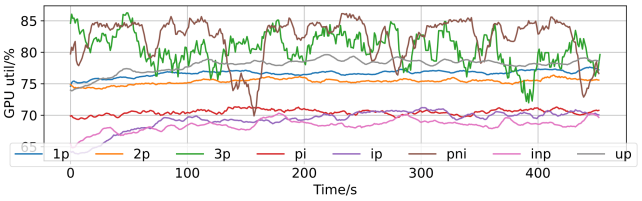

Figure 8: GPU utilization with different query structures on Freebase with BetaE [20].

Statistics of all the datasets can be found in Table 3. The KGs we use here are up to 1500× larger than those considered by prior work.

Software and training details. We implement all models in Python 3.8 using Pytorch 1.9 [17] with customized CUDA ops, and the samplers in C++ with multithreading. The machine has 8 NVIDIA V100 GPUs, 96x 2.00 GHz CPUs and 600GB RAM. All models have same embedding dimension for fair comparison.

5.2 Scalability

Speedup of bidirectional sampler. We verify the speed-up of the proposed bidirectional sampler over naïve exhaustive traversal one in Figure 6 (more results on other query structures can be found in Appendix E). We test different query structures where the bidirectional sampler is expected to achieve a square root reduction of the computation cost compared with traversal. We vary the

Table 7: MRR results (%) on large KGs with different methods all trained using SMORE.

Dataset \ Model	GQE	Q2B	BetaE
FB400k	36.02	**51.74**	50.50
ogbl-wikikg2	32.91	41.88	**44.42**
Freebase	80.71	**85.67**	84.33

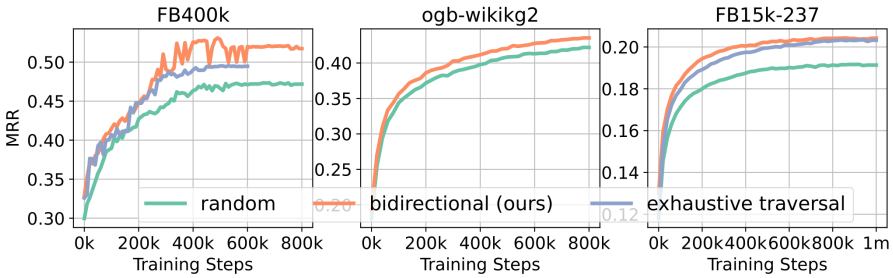

Figure 9: Accuracy (MRR) of different Q2B models trained using different samplers. We observe that our bidirectional sampler leads to more accurate models.

Table 8: Percentage of queries that span multiple partitions.

1p	2p	3p	4p	2i	3i	4i	ip	pi	up
11.5	18.8	25.3	30.7	20.7	28.8	35.5	26	25.3	26

constant C, *i.e.*, the maximum expansion of each relation, and plot the time for sampling a minibatch of 1024 queries as the function of C using the Freebase KG. We can see the traversal approach runs out of the time limit quickly as C grows, while our sampler stays significantly more efficient across all query structures (Figure 3).

GPU memory, utilization and end-to-end training speed. Here we show the scalability of SMORE on all six KGs. We first compare SMORE with prior implementation [18] on the three small benchmark datasets created in Ren and Leskovec [20]. As shown in the last two columns in Table 4, SMORE significantly improves the efficiency of end-to-end training of various query embeddings since SMORE adopts a query sampling scheme that shares the negative samples for a sampled batch of queries. Specifically, SMORE increases the speed by 119.4% and reduces GPU memory usage by 30.6% on average. Then we scale query embeddings up to the three large KGs. As shown in Table 6, given the same embedding dimension and batch size, our system design allows for (almost) graph-size agnostic speed and GPU-memory usage across all methods.

In Figure 8 we show the GPU utilization of BetaE with different multi-hop query structures on Freebase. We plot the average utilization of 8 GPUs with smoothing of 10 seconds. More complex queries like `3p` and `pni` have higher fluctuation due to the variance of sampling time. However as these queries also require more neural ops which in turn brings up the GPU utilization. Generally our system can keep a high GPU utilization for all query structures.

Multi-GPU speed up. We illustrate the speed-up of training on {1, 2, 4, 8} GPUs with different methods on ogbl-wikikg2 and Freebase datasets in Figure 7. Overall the speed grows almost linearly w.r.t. the number of GPUs, which shows the effectiveness of our asynchronous training and the communication overhead is negligible. Also the computationally heavier approaches like BetaE would benefit more from multi-GPU on SMORE.

Graph partitioning leads to lossy queries. Various prior large-scale KG embedding systems consider graph partitioning. However, since our system focuses on complex queries, with graph partitioning, a large number of queries will thus span multiple partitions. We conducted experiments on the largest Freebase KG to prove this using the state-of-the-art graph partitioning algorithm METIS [9]. Below we show the percentage of queries that would be lost (*i.e.*, a query that spans multiple partitions) if we partition the graph into 8 parts. We find that 1p (i.e., a 1-hop link) suffers the least while multi-hop queries may suffer greatly (more than 3x than 1p). This shows that although graph partitioning is a viable option for prior large-scale KG link prediction systems, it is not for multi-hop reasoning. This validates the necessity of our design choice.

KG completion runtime. Here we compare the single-hop link prediction (KG completion) runtime performance with state-of-the-art large-scale KG frameworks including Marius, DGL-KE and PBG. We report the results for ComplEx model with 100 embedding dimension on Freebase. We use the same multi-GPU V100 configuration as in Marius, and the current official release versions are also the same. So for baseline results we reuse the table 7 from Mohoney et al. [15]. As shown in Table 5, SMORE achieves significantly faster runtime in 1-GPU setting than PBG, while being slightly slower than Marius. Also it scales better than the other systems, while Marius does not officially support multi-GPU parallel training functionality at the current stage. Given that SMORE adopts the design choice of synchronized gradient update for dense parameters, and we do not partition the graph for the sake of multi-hop reasoning, it is nontrivial to be still comparable in the single-hop case.

5.3 Predictive performance

Calibration on small KGs. We first calibrate SMORE for GQE, Q2B, and BetaE on three small benchmark datasets created in BetaE. As shown in Table 4, SMORE achieves overall comparable performance as the models trained on a fixed set of training queries using a single GPU. Specifically, SMORE achieves comparable results for GQE and BetaE respectively while is able to further improve the performance of Q2B on FB15k with 3.54% increase in MRR.

Query answering on large KGs. We also benchmark the embedding models on three large KGs FB400k, ogbl-wikikg2 and Freebase. As shown in Table 7, on ogbl-wikikg2, BetaE performs the best among all the other baselines. On both FB400k and Freebase, box embedding model Q2B achieves the best results. For all these methods, the baseline implementation [18] cannot scale to such massive KGs due to limited memory and computationally expensive exhaustive query sampling. Our system SMORE easily scales query embeddings to these large KGs using an asynchronous design with sparse embedding and optimizer. The three KGs serve as an important benchmark for future multi-hop KG reasoning models.

Performance with different samplers. We compare the performance of Q2B trained with different samplers, *i.e.*, the naïve sampler (`exhaustive traversal`), bidirectional sampler (`bidirectional`)

and randomly sampling KG entities as negative answers (`random`) on FB15k-237, FB400k and ogbl-wikikg2. As shown in Figure 9, random sampling performs the worst since random negative sampling does not guarantee that the sampled entities are truly the negative (non-answer) entities. `bidirectional` performs comparable or even better than `exhaustive traversal`, as it can choose much larger C during query sampling. Note that `exhaustive traversal` is very slow on large graphs, and it requires months for the baseline implementation with exhaustive traversal sampler to train a model with the same number of queries on ogbl-wikikg2.

6 CONCLUSION

We present SMORE, the first general framework for both single- and multi-hop reasoning that scales up a plenty of different embedding methods with multi-GPU support to KGs with 86M nodes and 338M edges. It performs the algorithm-system co-optimization for scalability. Our work can also serve as the benchmark for future research on large-scale multi-hop KG reasoning.

ACKNOWLEDGMENTS

We thank Theo Rekatsinas, Rok Sosic, Xikun Zhang, Serena Chang, Michael Xie and Sharmila Nangi for providing feedback on our manuscript, Jason Mahoney and Roger Waleffe for discussions on Marius, Matthias Fey for discussions on sparse embeddings. We also gratefully acknowledge the support of DARPA under Nos. HR00112190039 (TAMI), N660011924033 (MCS); ARO under Nos. W911NF-16-1-0342 (MURI), W911NF-16-1-0171 (DURIP); NSF under Nos. OAC-1835598 (CINES), OAC-1934578 (HDR), CCF-1918940 (Expeditions), NIH under No. 3U54HG010426-04S1 (HuBMAP), Stanford Data Science Initiative, Wu Tsai Neurosciences Institute, Amazon, Docomo, Hitachi, Intel, JPMorgan Chase, Juniper Networks, KDDI, NEC, Toshiba, and UnitedHealth Group. Hongyu Ren is supported by the Masason Foundation Fellowship, the Apple PhD Fellowship and the Baidu Scholarship.

REFERENCES

[1] Erik Arakelyan, Daniel Daza, Pasquale Minervini, and Michael Cochez. 2021. Complex Query Answering with Neural Link Predictors. In *International Conference on Learning Representations (ICLR)*.

[2] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *ACM SIGMOD international conference on Management of data (SIGMOD)*. ACM.

[3] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[4] Xuelu Chen, Ziniu Hu, and Yizhou Sun. 2021. Fuzzy Logic based Logical Query Answering on Knowledge Graph. In *International Conference on Machine Learning (ICML)*.

[5] Nurendra Choudhary, Nikhil Rao, Sumeet Katariya, Karthik Subbian, and Chandan K Reddy. 2021. Probabilistic Entity Representation Model for Chain Reasoning over Knowledge Graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[6] Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. 2021. GNNAutoScale: Scalable and Expressive Graph Neural Networks via Historical Embeddings. In *International Conference on Machine Learning (ICML)*.

[7] Will Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. 2018. Embedding Logical Queries on Knowledge Graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[8] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[9] George Karypis and Vipin Kumar. 1995. METIS–unstructured graph partitioning and sparse matrix ordering system, version 2.0. (1995).

[10] Bhushan Kotnis, Carolin Lawrence, and Mathias Niepert. 2021. Answering complex queries in knowledge graphs with bidirectional sequence encoders. In *AAAI Conference on Artificial Intelligence (AAAI)*.

[11] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. Pytorch-biggraph: A large-scale graph embedding system. In *Conference on Machine Learning and Systems (MLSys)*.

[12] Bill Yuchen Lin, Xinyue Chen, Jamin Chen, and Xiang Ren. 2019. Kagnet: Knowledge-aware graph networks for commonsense reasoning. In *Empirical Methods in Natural Language Processing (EMNLP)*.

[13] Lihui Liu, Boxin Du, Heng Ji, ChengXiang Zhai, and Hanghang Tong. 2021. Neural-Answering Logical Queries on Knowledge Graphs. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*.

[14] Shangwen Lv, Daya Guo, Jingjing Xu, Duyu Tang, Nan Duan, Ming Gong, Linjun Shou, Daxin Jiang, Guihong Cao, and Songlin Hu. 2020. Graph-based reasoning over heterogeneous external knowledge for commonsense question answering. In *AAAI Conference on Artificial Intelligence (AAAI)*.

[15] Jason Mohoney, Roger Waleffe, Henry Xu, Theodoros Rekatsinas, and Shivaram Venkataraman. 2021. Marius: Learning Massive Graph Embeddings on a Single Machine. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*.

[16] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J Wright. 2011. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[18] Hongyu Ren. 2021. Snap-stanford KGReasoning: Multi-hop Reasoning on KGs. https://github.com/snap-stanford/KGReasoning.

[19] Hongyu Ren, Weihua Hu, and Jure Leskovec. 2020. Query2box: Reasoning over Knowledge Graphs in Vector Space using Box Embeddings. In *International Conference on Learning Representations (ICLR)*.

[20] Hongyu Ren and Jure Leskovec. 2020. Beta Embeddings for Multi-Hop Logical Reasoning in Knowledge Graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[21] Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. 2020. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

[22] Haitian Sun, Andrew O Arnold, Tania Bedrax-Weiss, Fernando Pereira, and William W Cohen. 2020. Faithful Embeddings for Knowledge Base Queries. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[23] Haitian Sun, Tania Bedrax-Weiss, and William W Cohen. 2019. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. In *Empirical Methods in Natural Language Processing (EMNLP)*.

[24] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations (ICLR)*.

[25] Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

[26] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *International Conference on Machine Learning (ICML)*.

[27] Hongwei Wang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. DKN: Deep knowledge-aware network for news recommendation. In *Proceedings of the International World Wide Web Conference (WWW)*.

[28] Qitian Wu, Hengrui Zhang, Xiaofeng Gao, Peng He, Paul Weng, Han Gao, and Guihai Chen. 2019. Dual graph attention networks for deep latent representation of multifaceted social effects in recommender systems. In *Proceedings of the International World Wide Web Conference (WWW)*.

[29] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations (ICLR)*.

[30] Zhanqiu Zhang, Jie Wang, Jiajun Chen, Shuiwang Ji, and Feng Wu. 2021. ConE: Cone Embeddings for Multi-Hop Reasoning over Knowledge Graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[31] Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. 2020. Dgl-ke: Training knowledge graph embeddings at scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[32] Zhaocheng Zhu, Shizhen Xu, Jian Tang, and Meng Qu. 2019. Graphvite: A high-performance cpu-gpu hybrid system for node embedding. In *Proceedings of the International World Wide Web Conference (WWW)*.

A QUERY COMPUTATION PLAN

As shown in Figure 2, the computation plan of a query consists of nodes $\mathcal{V}_q \cup \{V_1, \ldots, V_k, V_?\}$. Note each node of the computation plan corresponds to *a set of entities* on the KG. The edges on the computation plan represent a logical/relational transformation of this set:

(1) **Relation Projection:** Given a set of entities $S \subseteq \mathcal{V}$ and relation type $r \in \mathcal{R}$, compute adjacent entities $\cup_{v \in S} A_r(v)$ related to S via r: $A_r(v) \equiv \{v' \in \mathcal{V} : \ (v, r, v') \in \mathcal{E}\}$.
(2) **Intersection:** Given sets of entities $\{S_1, S_2, \ldots, S_n\}$, compute their intersection $\cap_{i=1}^{n} S_i$.
(3) **Complement/Negation:** Given a set of entities $S \subseteq \mathcal{V}$, compute its complement $\overline{S} \equiv \mathcal{V} \setminus S$.

Using De Morgan's laws, $\cup_{i=1}^{n} S_i$ is equivalent to $\overline{\cap_{i=1}^{n} \overline{S}}$, union operation can be replaced with three negation and one intersection operations.

Traversing the KG using the computation plan to find answers. Conceptually, (assuming no noise, no missing relations in the KG) a logical query can be answered by traversing the edges of the KG. For a valid query, the computation plan is a tree, where the anchor entity set, $\mathcal{V}_q$, are the leaves and the target variable $V_?$ is the single root, representing the set of answer entities (Figure 1(C)). Following the computation plan, we start with the anchor entities, traverse the KG and execute logical operators towards the root node. The answers $\mathcal{A}_q^{\mathcal{G}}$ to the query q are stored in the root node after the KG traversal. Note that this conceptual traversal would have exponential computational complexity with respect to the number of hops and also cannot handle noisy or missing relations in the KG, which are both very common in real-world KGs (Figure 1(C)).

Embedding-based "traversal" of the KG. Embedding-based reasoning methods avoid explicit KG traversal. Instead, they start with the embeddings of anchored entities, and then apply a sequence of neural logical operators according to the query computation plan. This way we obtain the embedding of the query where each embedding-based logical operator (*e.g.*, negation) takes the current input embedding and transforms it into a new output embedding. Such operators are then combined according to the query structure. The answers to the query q are then entities v that are embedded close to the final query embedding. The distance is measured by a pre-defined function $\texttt{Dist}(f_\theta(q), f_\theta(v))$, where $f_\theta(q)$ and $f_\theta(v)$ represents the query and entity embedding respectively. Note the distance function $\texttt{Dist}(\cdot, \cdot)$ is tailored to different embedding space and model design f_θ (see Table 1 and Appendix B).

B MULTI-HOP REASONING MODELS AND NEURAL LOGICAL OPERATORS

Our SMORE covers three published works, *i.e.*, GQE [7], Q2B [19] and BetaE [20]. In order to perform logical reasoning in the embedding space, all methods design a projection operator $\mathcal{P}$ and intersection operator $\mathcal{I}$. As introduced in Section 2, $\mathcal{P}$ represents a mapping from a set of entities (represented by an embedding) to another set of entities (also represented by an embedding) with one relation, *i.e.*, $\mathcal{P} : \mathbb{R}^d \times \mathcal{R} \to \mathbb{R}^d$, assuming the embedding dimension is d. The $\mathcal{I}$ takes as input multiple embeddings and outputs the embedding that represents the intersected set of entities: $\mathcal{I} : \mathbb{R}^d \times \cdots \times \mathbb{R}^d \to \mathbb{R}^d$. Different models may have different instantiations of these two operators.

GQE [7] embeds a query q to a point in the vector space by iteratively following the computation plan of the query. Query2box (Q2B) [19] proposes to embed queries as hyperrectangles with a center embedding and an offset embedding. BetaE [20] further proposes to embed queries as Beta distributions so that it can faithfully handle conjunction and negation operation in distribution space with KL-divergence as distance.

B.1 Comparison with neural link predictor [1]

Here we discuss the neural link predictor [1], which uses fuzzy logic to answer complex queries. It either requires online optimization for test query answering or traverses the KG with beam search, which is hard to scale to large KGs with their own unique challenges. In this paper, we focus on scaling up query embedding methods and leave scaling fuzzy-logic-based methods as future work.

C REVERSE DIRECTIONAL SAMPLING

Following prior work [19, 20], we use reverse directional sampling to construct queries from a KG. The overall instantiation process corresponds to a depth-first search where at each step we aim to ground a node/edge on the query structure with an entity/relation from the KG. Here we use one example to illustrate the whole idea. As shown in Figure 2, if we aim to instantiate an `ip` query from the query structure, we start the instantiation process from the root node, where we randomly sample an entity from the KG, here being `Neal`. Following the query structure, we aim to ground the edge that points to the root, and we sample a relation type from the KG that points to the entity `Neal`, *e.g.*, `Co-author`. Then we ground the next node, which has the relation `Co-author` with `Neal`, *e.g.*, `Bengio`. In the next step, since the edge is a logical operation `Intersection`, we may directly ground the next node with the same entity `Bengio`, and we sample another relation on the KG the relates to `Bengio`, *e.g.*, `Win`, and finally reaches the anchor entity (leaf) by sampling an entity on KG that has the relation `Win` with `Bengio`, *e.g.*, `Turing Award`. The overall complexity of this process is linear with respect to the number of hops of a query (structure).

Dynamic programming for optimal node cut. As the computation plan of q is a tree, we propose to solve the above optimization problem in Eqn (2) with dynamic programming (DP). Before presenting the algorithm, we first need to understand the cost of each operation, so as to only model the dominating cost in the dynamic programming. We consider the following operations:

- *Relation projection*: this operation enlarges the current set of entities by a factor of C (the maximum node degree) in the worst case. Thus the total cost would grows exponentially with the number of relation projection operations in a reasoning path.
- *Intersection / Union*: if we maintain the set of entities as a sorted list, intersection / union of the two sets only takes linear time w.r.t the number of entities in both sets. Thus it will not be the limiting factor in the overall computation cost if we only merge a constant number of sets together.
- *Negation / Complement*: The computation cost of a single set complement operation $O(|\mathcal{V}|)$, *i.e.*, the total number of entities in the KG. However, we can delay the complement operation to the next step on the computation plan, *i.e.*, perform

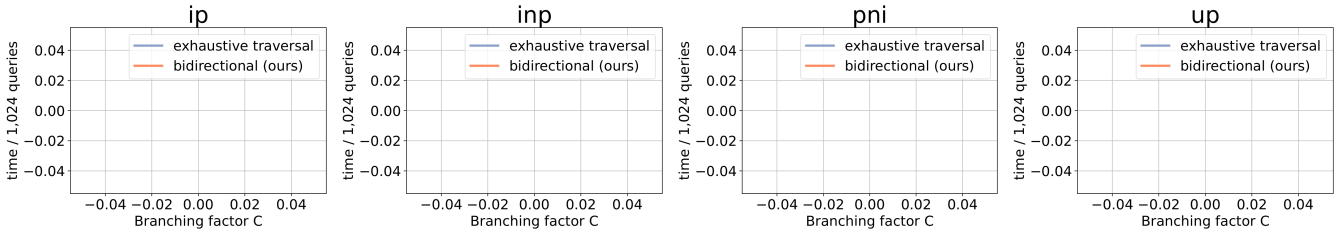

Figure 10: Speedup of bidirectional rejection sampler over exhaustive search based sampler, on different query structures.

complement+union/intersection simultaneously. This reduces the complexity from $O(\mathcal{V})$ to that of an intersection operation. For example in $(\neg a) \wedge b$, instead of first finding the complement of a (of complexity $|O(\mathcal{V})|$) and then do $\wedge$ (of complexity $O(|\mathcal{V} - a| + |b|)$), we can directory do set difference $b - a$ (of complexity $O(|a| + |b|)$. The only exception is the relation projection after the negation immediately. However there are other equivalent forms for this expression and thus we explicitly exclude such possibility in constructing the query structure.

With the analysis above, the bottleneck is the maximum number of relation projections in any reasoning path (*i.e.*, a path that connects a leaf/anchor entity $v \in \mathcal{V}_q$ and the root/answer entity $V_?$).

We first define a set of functions:

- $u(v)$: number of projections in the path from node v to root $V_?$;
- $s(v)$: the maximum length of path from v to any anchors. The "length" is measured by the number of projections on that path.
- $o(v)$: the optimal cost of resolving all the reasoning paths that includes v. Note this cost only cares the dominating one under the big-O notation, not the cost of entire search/reasoning.

Note that the "cost" are measured in the logarithmic scale, as we only care the dominating order of the polynomials in the complexity calculation. We use $p(v)$ to denote the parent of node v, and $ch(v)$ to denote the set of children. When v only has one child node, then we overload $ch(v)$ to denote that specific child. Then we can work on the recursion as below:

$$u(v) = u(p(v)) + \text{IsRel}(v \to p(v))$$

$$o(v) = \begin{cases} u(v), & \text{if } v \in \mathcal{V}_q \\ \min\left\{ \max_{z \in ch(v)} o(z), \max\{u(v), s(v)\} \right\}, & \text{else} \end{cases}$$

$$s(v) = \begin{cases} 0, & \text{if } v \in \mathcal{V}_q \\ \max_{z \in ch(v)} s(z), & \\ \qquad \text{if edges between } v \text{ and } ch(v) \text{ are } \wedge \text{ or } \vee & \\ s(ch(v)) + \text{NotNeg}(ch(v) \to v), & \text{else} \end{cases}$$

$\text{IsRel}(v \to p(v))$ returns 1 if the edge between v and $p(v)$ represents a relation projection and 0 otherwise; $\text{NotNeg}(ch(v) \to v)$ returns 1 if the edge between $ch(v)$ and v is not negation and 0 otherwise.

After solving the above DP, we can construct the node cut from solution $o(\cdot)$ in a top-down direction:

- If for any node v we have $\max_{z \in ch(v)} o(z)$ larger than $\max\{u(v), s(v)\}$, then we add v to node cut;
- Otherwise, we do the check recursively for $z \in ch(v)$.

The above procedure works linearly w.r.t. the size of q, which is good enough for large queries containing hundreds of operations. Example query structures and their optimal node cuts are shown in Figure 3. Note the structures considered in the current literature are small enough to find the optimal cut with the brute force algorithms. But our DP can greatly improve the efficiency when the query structures are large enough in certain applications.

D FURTHER OPTIMIZATION OF SMORE

In addition to the above optimized system design for SMORE, there are several other important optimization that further speeds up the training, which we highlight below.

Sharing negative samples. Although with the asynchronous design we can overlap the embedding R/W with GPU computation, it is still important to keep the size of memory exchange small. Inspired by Zheng et al. [31], we share the negative answers among the queries in a mini-batch. Each mini-batch data is formatted as $\left(\mathcal{N}, \{(q_i, \mathcal{V}_{q_i}, \mathcal{A}_{q_i}\}_{i=1}^{M}, Mask\right)$, where $\mathcal{N} \subset \mathcal{V}$ are the shared negative answers for all queries. $Mask \in \{0,1\}^{M \times |\mathcal{N}|}$ is an indicator matrix. $Mask_{i,j}$ specifies whether the j-th entity in $\mathcal{N}$ is a negative sample for q_i.

Customized CUDA distance kernel. Given M queries and negative answer candidates $\mathcal{N}$, the computation of $M \times |\mathcal{N}|$ pairs of distances is model dependent. While distances like inner-product or L2 can be implemented with efficient matrix multiplication, geometries designed for multi-hop reasoning like box or beta distribution requires more complicated distance metrics like KL divergence. We provide a generic interface to parallelize the computation with customized CUDA kernel, which also enables the operation fusion to reduce GPU memory consumption.

E ADDITIONAL RESULTS ON BIDIRECTIONAL REJECTION SAMPLER

Figure 10 shows the speed-up of bidirectional rejection sampler over exhaustive samplers on several more query structures. Ours is consistently significantly better than exhaustive traversal.

F BASELINE CODE

Following Marius [15], we adopt the same official release of DGL-KE and PBG. For Marius, we use the release at https://github.com/marius-team/marius/tree/osdi2021.