KGAccel: A Domain-Specific Reconfigurable Accelerator for Knowledge Graph Reasoning

Hanning Chen HANNINGC@UCI.EDU
University of California Irvine, Irvine, CA, USA.

Ali Zakeri ZAKERIJ@UCI.EDU
University of California Irvine, Irvine, CA, USA.

Yang Ni YNI3@UCI.EDU
University of California Irvine, Irvine, CA, USA.

Fei Wen FEI8WEN@GMAIL.COM
Texas A&M University, College Station, TX, USA.

Behnam Khaleghi BKHALEGHI@UCSD.EDU
University of California San Diego, La Jolla, USA.

Hugo Latapie HLATAPIE@CISCO.COM
Cisco Systems, San Jose, CA, USA.

Alvaro Velasquez ALVARO.VELASQUEZ@DARPA.MIL
University of Colorado Boulder, Boulder, CO 80309.

Mohsen Imani M.IMANI@UCI.EDU
University of California Irvine, Irvine, CA, USA.

Editors: G. Pappas, P. Ravikumar, S. A. Seshia

Abstract

Recent hardware accelerators for graph learning have largely overlooked knowledge graph reasoning (KGR), which demands more complex models and longer training times than typical graph tasks. Existing approaches rely on single or distributed GPUs to accelerate translational embedding models, but these general-purpose solutions lag in handling reinforcement learning-based KGR. To address this gap, we introduce KGAccel, the first domain-specific accelerator for RL-based KGR on FPGA. We develop a knowledge-graph compression method and propose a resource-aware mechanism that enables high-speed training even on smaller FPGAs. KGAccel achieves up to 65× speedup over CPU, 8× over GPU, and over 30× higher energy efficiency.

Keywords: Reinforcement Learning, Neuro Symbolic AI, Computer Architecture

1. Introduction

Knowledge graphs (KGs) play a crucial role in AI, structuring vast knowledge as multi-relational graphs where entities and relations form factual triples (h, r, t). Open-source KGs like Freebase Bollacker et al. (2008), DBpedia Auer et al. (2007), and YAGO Suchanek et al. (2007) store billions of facts, supporting applications in Q&A Huang et al. (2019), recommendation Lee et al. (2020), and information retrieval Wang et al. (2017). However, KGs remain incomplete, as incorporating unlimited concepts and keeping pace with evolving real-world data is challenging.

Reasoning, the ability to infer new knowledge from evidence, has long been a goal in computing. Early AI approaches focused on symbolic logic McCarthy et al. (1960); Xiong et al. (2022), later applied in machine learning Lao et al. (2011). However, their lack of generalization led to the adoption of vector representations. In KGs, reasoning helps detect errors and infer new relations, enabling advanced applications and automated knowledge discovery.

Training AI models for Knowledge Graph Reasoning (KGR) is computationally expensive, necessitating acceleration. While hardware accelerators exist for graph tasks like analysis Asiatici and Ienne (2021), clustering Yan et al. (2020); Li et al. (2021), and mining Chen et al. (2021), KGR remains underexplored. Prior efforts focused on speeding up translation models, such as TransE Bordes et al. (2013) and TransR Lin et al. (2015), on GPUs. These models map entities and relations into embeddings but struggle with complex multi-hop reasoning due to low accuracy.

Reinforcement learning (RL)-based models Xiong et al. (2017); Lin et al. (2018) offer higher reasoning accuracy for KGR but suffer from long training times. Studies Cho et al. (2019); Rothmann and Porrmann (2022) show that reconfigurable platforms outperform GPGPUs in RL acceleration. However, existing RL accelerators Rothmann and Porrmann (2022) do not address KGR's large-scale graph data and complex state transitions. Additionally, KGR tasks have a dynamic and sparse action space, requiring efficient on-chip resource utilization. **Thus, developing domain-specific RL accelerators for symbolic KGR is crucial.**

In this paper, we propose KGAccel, an FPGA-based accelerator targeting KGR tasks. We implement both the knowledge graph environment and RL agent on a single FPGA. Here we summarize our main contributions:

- The first domain-specific FPGA accelerator for RL-based KGR, achieving superior accuracy and efficiency over prior GPU-based methods.
- An efficient CSR-based KG compression technique that reduces storage overhead and minimizes costly off-chip memory accesses.
- A fine-tuned systolic array-based neural network accelerator integrating forward propagation and weight updates in a single pass for efficient RL training.
- A context-switching mechanism validated on mid-tier FPGAs for flexible deployment across edge and cloud environments.

We evaluate our approach by the task of missing link prediction, over two widely used knowledge graph datasets. We implement our KGAccel on two different classes of FPGA platforms: the large-size Xilinx Alveo U280 and the small-size Xilinx ZCU104. The results demonstrate that KGAccel on Alveo U280 achieves an average of 65$\times$ speedup and 45$\times$ higher energy efficiency compared to state-of-the-art RL-based reasoning models Xiong et al. (2017) on Intel Xeon Silver 4114. When deploying KGAccel on a resource-limited FPGA board ZCU104, it shows over 15$\times$ speedup and over 50$\times$ energy efficiency improvement over the CPU platform.

2. Knowledge graph reasoning

2.1. Definitions

A knowledge graph (KG) is a directed graph $G = \{(e_s, r, e_o) \mid e_s, e_o \in \xi,\ r \in R\}$, where ξ is the entity set and R is the relation set Lin et al. (2018). Figure 1.(a) illustrates a KG, where each directed edge represents a fact triple (e_s, r, e_o). For example, (`Lionel_Messi`, `born_in`, `Rosario`) and (`Jordi_Alba`, `live_in`, `Miami`) encode real-world facts. Entities can belong

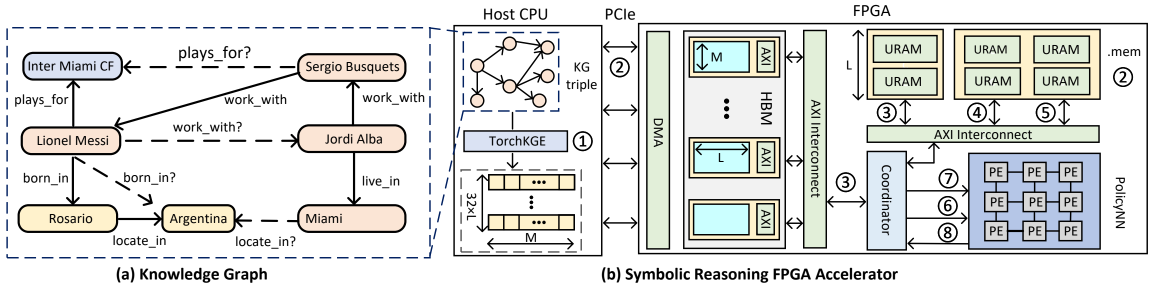

Figure 1: (a). An example of a knowledge graph with potential reasoning queries. (b). CPU-FPGA knowledge graph reasoning (KGR) acceleration architecture. 1. Entity and relation word embedding. 2. Transfer of embedding vector from host CPU to kernel FPGA via PCIe interface and Xilinx DMA. 3. Entity embedding vectors preload. 4. Address access. 5. Outgoing relations access. 6. Teacher's guide for agent 7. Provision of state, reward, and loss to RL agent. 8. Agent's action probability distribution based on current policy.

to multiple triples, with relations forming directed edges—source nodes as subjects and destination nodes as objects. In `Rosario is located in Argentina`, `Rosario` is the subject and `Argentina` the object.

Reasoning tasks infer explicit formulas within a KG Xiong et al. (2017, 2024), differing from clustering or pattern mining. As shown in Figure 1.(a), a KGR model can predict missing relations, such as (`Lionel_Messi`, `born_in`, `Argentina`), a task known as link prediction. Another reasoning task is fact evaluation, where the model verifies statements. For instance, (`Sergio_Busquets`, `plays_for`, `Inter Miami CF`) should be accepted, while (`Miami`, `locate_in`, `Argentina`) should be rejected. These models are crucial for complex Q&A systems. A comparison of path-based and embedding-based KGR is in Appendix 7.1.

3. Reinforcement Learning for KGR

Reinforcement Learning is a powerful technique to solve decision-making problems such as robotics control. Most RL models are built around two main structures: external environment and agent. The environment for the KGR is modeled as a Markov Decision Process (MDP), based on Xiong et al. (2017); Das et al. (2017). MDPs are formally represented as a 4-tuple $(S,\ A,\ P_a,\ R_a)$, in which S is the state space, A is the action space, $P_a(s,\ s') = P(s_{t+1} = s' \,|\, s_t = s,\ a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t+1$, we have $s_t \in S$ and $a_t \in A$, and $R_a(s,\ s')$ is the reward received after transitioning from state s to s', due to action a. In contrast to previous RL acceleration works that maintain the external environment on CPU Cho et al. (2019); Yang et al. (2021), our work introduces an on-chip KG RL environment as presented in Section 4.3.

With regard to the agent, a new action distribution will be generated at each time step based on the policy $\pi_\theta(s,\ a) = p(a|s;\theta)$, where θ is the policy parameter Ding et al. (2023). Depending on the similarity of observational policy and target policy, the RL algorithm can be divided into on-policy and off-policy categories. Since the action space in the pathfinding task for knowledge graphs is generally very large, previous work Xiong et al. (2017); Das et al. (2017) suggests using on-policy RL as the policy optimization strategy. For a comprehensive discussion on RL in the context of KGR tasks, refer to Appendix 7.2.

4. Architecture Design of KGAccel

4.1. Accelerator design motivation

Although previous works mentioned the possibility of accelerating GNN targeting knowledge graphs Zhou et al. (2022); Liang et al. (2020), these works cannot be directly applied to KGR (KGR) tasks. The RL-based KGR algorithm has an inherent sparsity problem since the agent can only pick one action from multiple potential relations at each time step. Furthermore, as each vertex only connects with certain specific types of relations, the action space for the agent varies as it moves along different vertices. Hence, a mechanism is needed to remove the sparsity during action selection processes. Compared to the RL tasks in prior acceleartion works Cho et al. (2019); Meng et al. (2020), RL-based KGR tasks incur large-scale graph data access during the state transition between time steps. To reduce the expensive off-chip memory accesses, the objective is to store the entire knowledge graph in on-chip storage.

4.2. CPU-FPGA Heterogeneous Acceleration Architecture

Figure 1 provides an overview of KGAccel on the Xilinx Alveo platform. The host CPU preprocesses the input KG using a pre-trained embedding model like TransE Bordes et al. (2013), implemented via TorchKGE Boschin (2020). Entities are represented as M-dimensional embedding vectors, quantized to 16-bit fixed-point values, and stored in FPGA off-chip DRAM or HBM Choi et al. (2021). To optimize memory bandwidth, we distribute entity vectors across 32 HBM pseudo channels (PCs). KG triples, including entity neighbors, are stored in on-chip UltraRAM (URAM). Due to KG sparsity, efficient neighbor access is challenging (see Section 4.3). For each KGR task, KGAccel preloads entity embeddings into URAM, triggering a context switch when a vector is missing (Section 4.6). An MLP-based policy network (PolicyNN) guides agent actions, while the coordinator IP serves as both the RL environment and a training guide, issuing AXI access requests to on-chip storage (Sections 4.5, 4.6).

4.3. KG compression and on-chip storage

The size of the knowledge graph (KG) could be huge. Here size is defined from two perspectives, the entity size and the triple size. As discussed in section II-A, each graph node is an entity, and the node-to-node combination is triple. Take FB15K-237 Toutanova et al. (2015) for example, the entities' size is over 10K and the triples' size is over 300K. Suppose a 32-bit unsigned fixed-point number represents each entity. The storage overhead of KG is on a megabyte(MB) level without graph compression. For FPGA-based computing, an optimized KG representation and storage format are necessary since on-chip storage resources are fast but limited.

To address KG sparsity, we use the Compressed Sparse Row (CSR) format Buluç et al. (2009). As shown in Figure 2, consider a graph with |V| entities, |E| triples, and 11 relations. Entity e_{13} has only four outgoing edges, though the total relation count is 11. Storing edges in a standard adjacency matrix leads to excessive zero entries and inefficient memory access due to varying edge counts per entity. CSR efficiently represents KG using three vectors: vertex $\boldsymbol{V}$, relation $\boldsymbol{R}$, and neighbor $\boldsymbol{N}$. $\boldsymbol{V}$ has size |V|+1, while $\boldsymbol{R}$ and $\boldsymbol{N}$ have size |E|. For entity e_i, $V[i]$ gives the starting

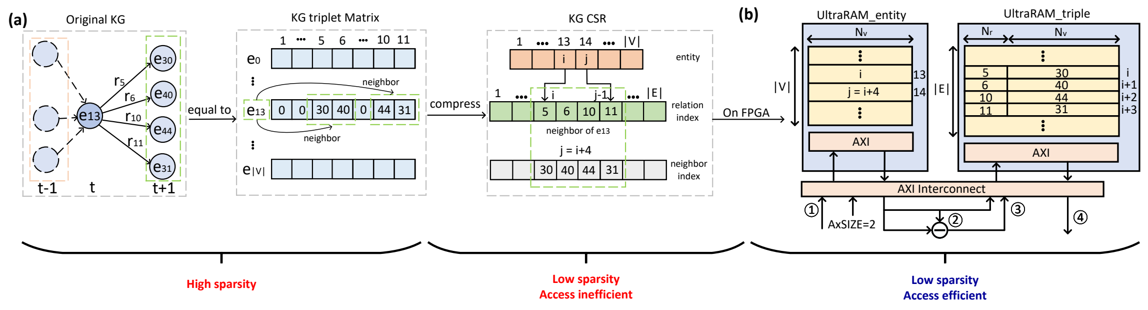

Figure 2: Knowledge Graph (KG) compression and storage on FPGA. $|E|$ and $|V|$ represent the size of triples and entities in KG, respectively. We also assume the total number of relations ($|R|$) is 11, without loss of generality. (a) Knowledge graph compression from naive adjacent matrix representation into CSR representation. (b) CSR represented knowledge graph storage on FPGA on-chip storage. Here we use UltraRAM as an example. Process 1 and 3 represent AXI burst reading request. Process 2 represents vertex neighbor address range calculation. Process 4 represents the access of the entity's neighbor index and corresponding relation.

index of its neighbors, and $V[i+1] - V[i]$ provides the range. If index j falls within this range, $R[j]$ and $N[j]$ store the relation r_j and neighbor t_j of e_i.

$$< e_i, r_j, t_j > \subseteq G == True \quad when \quad j \in H_i \tag{1}$$

Figure 2 also presents the actual CSR format KG memory storage layout on FPGA UltraRAM. Suppose each entity is represented by a 18-bits unsigned fixed point number. Since Xilinx UltraRAM's bit width is controlled by the user, for larger graph with more entities, increasing memory width is necessary. The memory width of Vertex RAM N_V is:

$$N_V = \lceil log_2(|E|) \rceil \quad where\ |E|\ is\ triple\ size \tag{2}$$

Figure 2.b presents entity's neighbor access process. To access entity ith's neighbor information, as is shown in Figure 2, two AXI burst read requests are needed. The first request will be sent to $UltraRAM_{entity}$ with $i \times N_V + a_{base}$ as AXI base address and 2 as AXI burst size. Based on the two return value, we can get the base address and range for the current entity's information in $UltraRAM_{triple}$. Then the second AXI read request will be sent to $UltraRAM_{triple}$. After the second AXI request is finished, we get the information of the entity's every neighbor. The format of AXI request is based on AMBA AXI4 protocol Prasanth and Raj (2014).

4.4. Policy network acceleration

This section presents the RL agent's policy network design. At each time step, KGAccel inputs the current state vector into the policy network to generate an action probability distribution, determining the relation connected to the current entity in KGR. Figure 3.(a) illustrates a fully connected three-layer neural network. The input state matrix has dimensions **B** $\times$ **E**, where **B** is the batch size and **E** is the state vector dimension. As discussed in Appendix Section 7.2.1, the state vector concatenates the source entity and distance embeddings, making **E** twice the embedding dimension. For

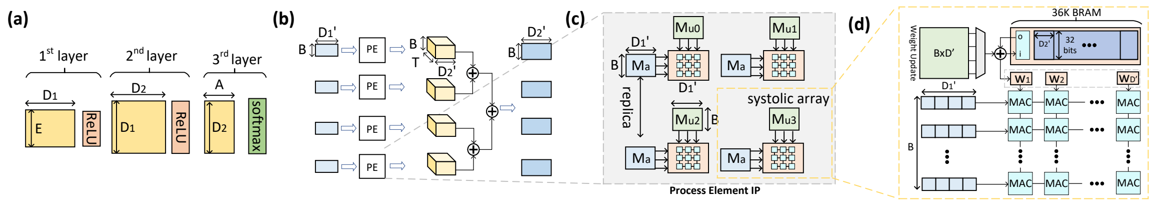

Figure 3: Policy Network Architecture Design. (a) three-layer neural network structure. (b) second layer data path. (c) Processing Element (PE) IP architecture design. (d) Systolic array IP microarchitecture

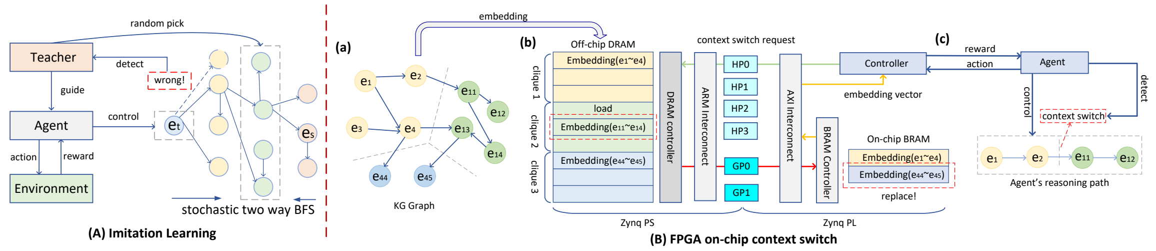

Figure 4: (A) Imitation learning illustration. (B) On-chip storage context switch. Here we use Xilinx Zynq platform as an example. Xilinx Zynq platform includes ARM core as process system (PS) and FPGA logic (PL). The large off-chip DRAM is connected with PS. Limited on-chip BRAM can only store part of knowledge graph embedding clique. (a) Knowledge graph partitions. (b) Context switch between off-chip DRAM and on-chip BRAM architetcure illustration. (c) RL agent's reasoning path over knowledge graph.

a three-layer network, weight matrices have dimensions $\mathbf{E} \times \mathbf{D_1}$, $\mathbf{D_1} \times \mathbf{D_2}$, and $\mathbf{D_2} \times \mathbf{A}$, where $\mathbf{A}$ is the action space. The forward pass follows:

$$P_A = softmax(W3 * ReLU(W_2 * ReLU(W_1 * I))) \tag{3}$$

Here P_A is the action probability distribution. The action space in KGR equals the total KG relations, making the output layer dimension large and requiring high hidden layer capacity. To optimize FPGA resources like LUTs and BRAMs, KGAccel splits hidden layer matrices. Figure 3.(b) illustrates splitting the $B \times D_1 \times D_2$ systolic array into $T \times T$ smaller $B \times D_1' \times D_2'$ arrays, where $D_1' = D_1/T$ and $D_2' = D_2/T$. With $T = 4$, each layer contains four Processing Elements (PEs), each hosting four systolic arrays. Figure 3.(c) and Figure 3.(d) show the internal PE and systolic array designs, respectively. BRAM is distributed across FPGA logic fabric to maximize utilization. KGAccel ensures each systolic array IP's weight matrix fits within 18K or 36K BRAM, the most common sizes in Xilinx FPGAs. Section IV-D details policy network updates.

4.5. Policy network update and imitation learning acceleration

As is shown in Figure 3.(d), the policy network update module is integrated into the forward path. KGAccel chooses not to maintain specific back propagation logic since, in the RL training process, the model update is unnecessary for each time step. Therefore we choose to maintain an Adam optimizer IP inside the coordinate IP and update the model inside the forward path. As we introduce imitation learning into our framework, the agent's policy network update will be guided by a teacher. Next, we will discuss it. As depicted in Figure 4.(A), the imitation learning framework comes to play when the agent fails during its retraining. This happens when the agent is not able to find a path during the maximum time limit of an episode. In such case, the supervised learning policy, i.e. the teacher, suggests paths for the agent to learn, which helps the agent update much faster comparing to vanilla RL training. Note that the negative reward of the episode due to failing is applied as usual, but the considers teacher's knowledge as well during its update at the end of the episode. We provide more analysis of KGR challenge in Appendix section 7.3.

4.6. On-chip entity embedding vector switch

Our framework stores KG triples in on-chip URAM, while entity embedding vectors reside in off-chip DRAM or HBM. As shown in Figure 1.(B), step 1, KGAccel employs a pre-trained text embedding model for feature extraction, generating high-dimensional vectors. For instance, a 50-dimensional vector for a KG with 10,000 entities requires nearly 2MB of storage, making full on-chip storage impractical for edge devices like the Xilinx Zynq board. Inspired by Wen et al. (2021b,a), we adopt a hierarchical memory scheme: on-chip URAM as primary storage and off-chip DRAM as secondary. Figure 4.(B) illustrates the context switch process on Zynq, where KGAccel partitions the KG into smaller cliques, loading one at a time into the FPGA kernel. If an entity's embedding vector is absent on-chip, KGAccel swaps stored vectors with the required ones. This strategy enables small-scale FPGAs to handle large KGs efficiently.

5. Implementation and Experimental Result

5.1. KG datasets and experiment setup

Hardware platform We implement DeepPath Wang et al. (2014) as a KGR baseline on Intel Xeon Silver 4114 (CPU) and NVIDIA GPUs (GTX 1660 Ti, RTX 3090) using TensorFlow Abadi (2016). KGAccel is synthesized on Xilinx FPGAs: Alveo U280 (U280) and ZCU104, where GTX 1660 and ZCU104 represent edge platforms, while RTX 3090 and U280 serve cloud platforms. For reasoning accuracy, we use TransE Bordes et al. (2013) and TransR Lin et al. (2015) from TorchKGE Boschin (2020). U280 offers more on-chip storage than ZCU104, impacting RL training speed due to context switching (Section 4.6). For U280, we use Xilinx Vitis Kathail (2020) to manage CPU-FPGA communication via PCIe and DMA, though overhead is minimal as the KG is stored on FPGA. Implementation details are in Appendix 7.4.

Knowledge Graph Datasets Table 1 presents the datasets used in our experiments: FB15K-237 Toutanova et al. (2015) and NELL-995 Wang et al. (2019), the most common KGs for evaluating reasoning capabilities. Our agent infers entity relations from existing knowledge, with task details covered in Section 5.2. Beyond basic parameters, we report dataset sparsity (inverse of density) and storage overhead. Higher sparsity increases RL decision complexity. $Overhead_1$ in Table 1 represents KG triple storage, while $Overhead_2$ accounts for adjacency matrix maintenance during training. Without optimization, this overhead is prohibitive for edge devices like FPGAs. Section 4.3 discusses our compression method's impact, further detailed in Section 5.3.

Table 1: Knowledge graph datasets parameter

	#Entity	#Relation	#Triples	#Tasks	Density	Overhead_1	Overhead_2
FB15K-237	14505	237	310116	20	9.02%	1.18MB	13.11MB
NELL-995	75492	200	154213	12	1.0%	0.58MB	57.59MB

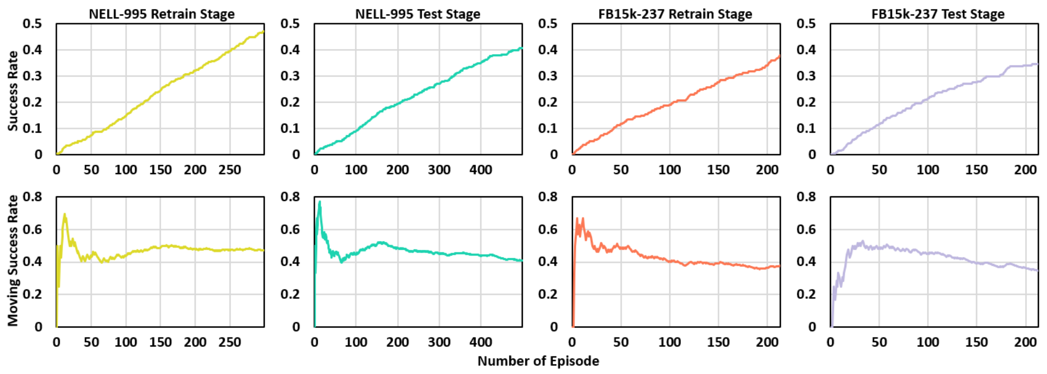

Figure 5: Success rate plots during retraining and testing stages of the algorithm, for NELL-995 and FB15K-237 datasets.

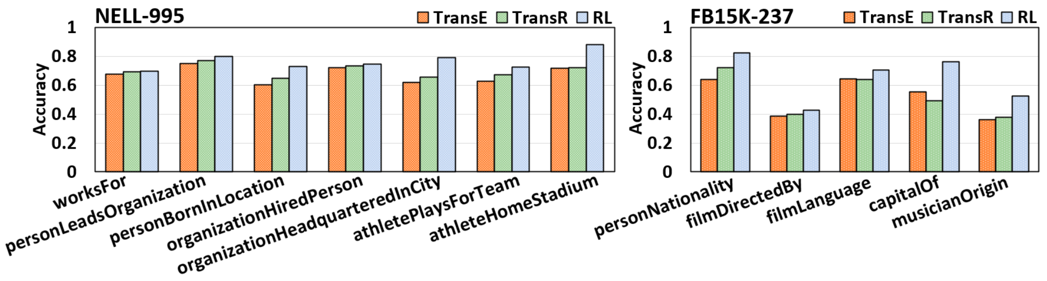

Figure 6: Reasoning results of the link prediction task for NELL-995 and FB15K-237 datasets.

5.2. KGAccel reasoning experiments and results

To evaluate KGAccel, we conduct link prediction, a standard KGR task addressing KG incompleteness. The model is trained on NELL-995 and FB15K-237, performing reasoning across diverse entity-relation domains. For each task, all instances of a relation and its inverse are removed and treated as queries. Training occurs in three stages: (1) supervised policy learning using bidirectional BFS, (2) RL retraining with reward-based updates, guided by the supervised policy as a fallback, and (3) testing without teacher assistance. Training runs for up to 500, 300, and 500 episodes per stage, respectively, with one task per episode. Figure 5 depicts success rate and moving success rate of the agent for its training. Success rate is determined by ratio of current number of successful episodes to the total number of episodes, while moving success rate is calculated as the ratio of current number of successful episodes to current number of episodes. Figure 6 shows the mean average precision (MAP) results, used as an evaluation metric to compare our model with two main embedding methods, TransE and TransR. We present more detailed analysis of reasoning accuracy in Appendix Section 7.5.

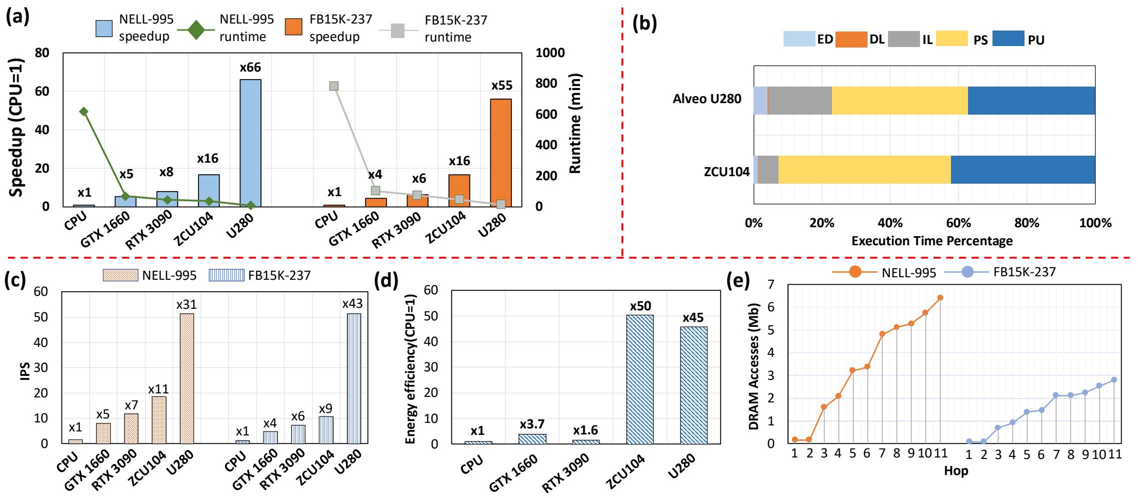

Figure 7: (a) Training latency/speedup of different platforms. (b) Breakdown of the execution time of two FPGA platforms. Here, ED represents entity embedding time, DL represents data loading from CPU to FPGA, IL represents imitation learning process, PS represents path search process, and PU represents agent policy update. (c) Training throughput of different platforms. (d) Energy efficiency of different platforms. CPU's energy efficiency is set as baseline. (e) Data switch size per episode with varying reasoning hop number.

5.3. The hardware performance of KGAccel

This section evaluates KGAccel's acceleration in terms of latency, throughput, energy efficiency, and memory footprint. Throughput is measured in inferences per second (IPS), following prior RL acceleration works Cho et al. (2019); Meng et al. (2020); Yang et al. (2021). We emphasize memory footprint on the edge FPGA (Xilinx ZCU104), as Alveo U280 has enough on-chip resources to store KG triples, embeddings, and PolicyNN weights, minimizing memory access. In contrast, ZCU104 holds only part of the graph clique's embeddings, occasionally triggering the context switch process (Section 4.6) during training.

Training Latency Figure 7.(a) shows the average training latency per task across five platforms. While ZCU104 has lower performance than Alveo U280 due to limited resources, both FPGA platforms outperform GPUs by storing network weights and KG data on-chip, reducing data transfer overhead. Increasing CUDA cores improved GPU training speed by only $\sim 1.6\times$ due to the sequential nature of reasoning tasks, limiting parallelism. FPGA's parallelism, reconfigurability, and near-memory computing make it more suitable for KGR. KGAccel further optimizes efficiency by analyzing action space before computation. Figure 7.(b) breaks down the total runtime on FPGA platforms across five key training stages: entity embedding (ED), data loading (DL), imitation learning (IL), path search (PS), and policy update (PU). In CPU-FPGA setups, ED and DL occur on the CPU, with ED using a pre-trained TransE model Bordes et al. (2013); Boschin (2020). Unlike prior work Zheng et al. (2020), which stops at ED, our method continues through IL, PS, and PU, which dominate computation time and are essential for improving reasoning accuracy.

RL Agent Reasoning Throughput Figure 7.(c) shows reasoning throughput (IPS) across five platforms. Compared to RL accelerators for OpenAI Gym tasks Cho et al. (2019); Meng et al. (2020); Yang et al. (2021), KGAccel's throughput is lower due to two factors. First, unlike Gym tasks with fixed action spaces, KGR's available actions change dynamically. If the agent makes a wrong move, it remains at the current entity, updating its policy via imitation learning, reducing exploration. Second, IPS definitions differ: in standard RL, IPS measures environment interactions, whereas in KGR, it tracks the agent's reasoning path length per second. Unlike prior RL accelerators using multi-threaded CPU environments, KGAccel stores the KG environment on FPGA, focusing on finding a successful path.
Energy Efficiency Appendix Table 3 reports KGAccel's power consumption on two FPGA platforms, measured via Xilinx Power Estimator (XPE). ZCU104 consumes less power than Alveo U280 due to fewer systolic array IPs and no HBM. Figure 7.(d) compares energy efficiency (IPS/W) across platforms. CPU and GPU power usage is measured via Intel Powertop Larsson (2011) and nvidia-smi Imani et al. (2020). GPUs suffer from high power consumption, making them less suitable for edge environments, while FPGAs offer superior efficiency. Using Xilinx XRT, we recorded Alveo U280 at 59.3W, and a Hioki 3334 power meter measured ZCU104 at 19.2W.
Memory Footprint Due to limited on-chip resources, ZCU104 stores only part of the entity embeddings, prioritizing network weights and compressed KG triples (Section 4.6). When an entity's embedding is missing, the agent requests it from the host CPU, triggering a context switch. Context switch frequency increases with reasoning hops; for single-hop reasoning, it occurs in 10% of episodes. Figure 7.(e) shows the average switch size per hop. To mitigate this, an efficiency reward (Appendix Section 7.2.1) helps shorten reasoning paths, reducing data transfers. Solutions include using larger boards like Alveo U280 or enhancing CPU-FPGA bandwidth with optimized interconnect IPs, which is a focus for future research.

6. Conclusion

In this paper, we propose the first FPGA acceleration work of KGR to the best of our knowledge. To reduce the overhead of host CPU and kernel FPGA communication, we apply CSR-based compression to the original high-sparsity knowledge graph and maintain the knowledge graph within FPGA on-chip storage. We implement on-chip imitation learning using systolic arrays to accelerate the reinforcement learning agent's training. Furthermore, we propose an on-chip storage replacement mechanism and validate it on a small-sized FPGA, empowering edge computing devices to conduct reasoning tasks.

Acknowledgments

This work was supported in part by the DARPA Young Faculty Award, the National Science Foundation (NSF) under Grants #2127780, #2319198, #2321840, #2312517, and #2235472, the Semiconductor Research Corporation (SRC), the Office of Naval Research through the Young Investigator Program Award, the U.S. Army Combat Capabilities Development Command (DEVCOM) Army Research Laboratory under Support Agreement No. USMA 21050, and Grants #N00014-21-1-2225 and N00014-22-1-2067. Additionally, support was provided by the Air Force Office of Scientific Research under Award #FA9550-22-1-0253, along with generous gifts from Xilinx and Cisco.

References

Martín Abadi. Tensorflow: learning functions at scale. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, pages 1–1, 2016.

Mikhail Asiatici and Paolo Ienne. Large-scale graph processing on fpgas with caches for thousands of simultaneous misses. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 609–622. IEEE, 2021.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250, 2008.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

Armand Boschin. Torchkge: Knowledge graph embedding in python and pytorch. In *International Workshop on Knowledge Graph: Mining Knowledge Graph for Deep Insights*, Aug 2020.

Aydin Buluç, Jeremy T Fineman, Matteo Frigo, John R Gilbert, and Charles E Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 233–244, 2009.

Jiayu Chen, Jingdi Chen, Tian Lan, and Vaneet Aggarwal. Multi-agent covering option discovery based on kronecker product of factor graphs. *IEEE Transactions on Artificial Intelligence*, 2022.

Jiayu Chen, Jingdi Chen, Tian Lan, and Vaneet Aggarwal. Learning multiagent options for tabular reinforcement learning using factor graphs. *IEEE Transactions on Artificial Intelligence*, 4(5), 2023. ISSN 2691-4581.

Xuhao Chen, Tianhao Huang, Shuotao Xu, Thomas Bourgeat, Chanwoo Chung, and Arvind Arvind. Flexminer: A pattern-aware accelerator for graph pattern mining. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 581–594. IEEE, 2021.

Hyungmin Cho, Pyeongseok Oh, Jiyoung Park, Wookeun Jung, and Jaejin Lee. Fa3c: Fpga-accelerated deep reinforcement learning. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 499–513, 2019.

Young-kyu Choi, Yuze Chi, Weikang Qiao, Nikola Samardzic, and Jason Cong. Hbm connect: High-performance hls interconnect for fpga hbm. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 116–126, 2021.

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. *arXiv preprint arXiv:1711.05851*, 2017.

Aolin Ding et al. Get your cyber-physical tests done! data-driven vulnerability assessment of robotic vehicle. In *2023 53nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE Computer Society, jun 2023. doi: 10.1109/DSN58367.2023.00020.

Tong Geng, Ang Li, Runbin Shi, Chunshu Wu, Tianqi Wang, Yanfei Li, Pouya Haghi, Antonino Tumeo, Shuai Che, Steve Reinhardt, et al. Awb-gcn: A graph convolutional network accelerator with runtime workload rebalancing. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 922–936. IEEE, 2020.

Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. Knowledge graph embedding based question answering. In *Proceedings of the twelfth ACM international conference on web search and data mining*, pages 105–113, 2019.

Yu Huang, Long Zheng, Pengcheng Yao, Qinggang Wang, Xiaofei Liao, Hai Jin, and Jingling Xue. Accelerating graph convolutional networks using crossbar-based processing-in-memory architectures. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1029–1042. IEEE, 2022.

Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

Mohsen Imani, Mohammad Samragh Razlighi, Yeseong Kim, Saransh Gupta, Farinaz Koushanfar, and Tajana Rosing. Deep learning acceleration with neuron-to-memory transformation. In *2020 IEEE international symposium on high performance computer architecture (HPCA)*, pages 1–14. IEEE, 2020.

Vinod Kathail. Xilinx vitis unified software platform. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 173–174, 2020.

Ni Lao, Tom Mitchell, and William Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the 2011 conference on empirical methods in natural language processing*, pages 529–539, 2011.

Petter Larsson. Energy-efficient software guidelines. *Intel Software Solutions Group, Tech. Rep*, pages 1–11, 2011.

Dongho Lee, Byungkook Oh, Seungmin Seo, and Kyong-Ho Lee. News recommendation with topic-enriched knowledge graphs. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pages 695–704, 2020.

Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. Pytorch-biggraph: A large scale graph embedding system. *Proceedings of Machine Learning and Systems*, 1:120–131, 2019.

Jiajun Li, Ahmed Louri, Avinash Karanth, and Razvan Bunescu. Gcnax: A flexible and energy-efficient accelerator for graph convolutional neural networks. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 775–788. IEEE, 2021.

Shuangchen Li, Dimin Niu, Yuhao Wang, Wei Han, Zhe Zhang, Tianchan Guan, Yijin Guan, Heng Liu, Linyong Huang, Zhaoyang Du, et al. Hyperscale fpga-as-a-service architecture for large-scale distributed graph neural network. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 946–961, 2022.

Youjie Li, Iou-Jen Liu, Yifan Yuan, Deming Chen, Alexander Schwing, and Jian Huang. Accelerating distributed reinforcement learning with in-switch computing. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 279–291. IEEE, 2019.

Shengwen Liang, Ying Wang, Cheng Liu, Lei He, LI Huawei, Dawen Xu, and Xiaowei Li. Engn: A high-throughput and energy-efficient accelerator for large graph neural networks. *IEEE Transactions on Computers*, 70(9):1511–1525, 2020.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Ting-Ru Lin, Drew Penney, Massoud Pedram, and Lizhong Chen. A deep reinforcement learning framework for architectural exploration: A routerless noc case study. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 99–110. IEEE, 2020.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping. *arXiv preprint arXiv:1808.10568*, 2018.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

Yi-Chien Lin, Bingyi Zhang, and Viktor Prasanna. Hp-gnn: Generating high throughput gnn training implementation on cpu-fpga heterogeneous platform. In *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 123–133, 2022.

John McCarthy et al. *Programs with common sense*. RLE and MIT computation center Cambridge, MA, USA, 1960.

Yuan Meng, Sanmukh Kuppannagari, and Viktor Prasanna. Accelerating proximal policy optimization on cpu-fpga heterogeneous platforms. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 19–27. IEEE, 2020.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. A novel embedding model for knowledge base completion based on convolutional neural network. *arXiv preprint arXiv:1712.02121*, 2017.

M Prasanth and Juhi Raj. Synthesizable axi4 protocol checker. *International Journal of Advances in Engineering & Technology*, 6(6):2587, 2014.

Marc Rothmann and Mario Porrmann. A survey of domain-specific architectures for reinforcement learning. *IEEE Access*, 10:13753–13767, 2022.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. M-walk: Learning to walk over graphs using monte carlo tree search. *Advances in Neural Information Processing Systems*, 31, 2018.

Linghao Song, Youwei Zhuo, Xuehai Qian, Hai Li, and Yiran Chen. Graphr: Accelerating graph processing using reram. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 531–543. IEEE, 2018.

Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706, 2007.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*, 2019.

Nishil Talati, Haojie Ye, Yichen Yang, Leul Belayneh, Kuan-Yu Chen, David T Blaauw, Trevor N Mudge, and Ronald G Dreslinski. Ndminer: accelerating graph pattern mining using near data processing. In *ISCA*, pages 146–159, 2022.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1499–1509, 2015.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. *arXiv preprint arXiv:1911.03082*, 2019.

Heng Wang, Shuangyin Li, Rong Pan, and Mingzhi Mao. Incorporating graph attention mechanism into knowledge graph reasoning based on deep reinforcement learning. 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 2623–2631, 2019.

Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12): 2724–2743, 2017.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI conference on artificial intelligence*, volume 28, 2014.

Fei Wen, Mian Qin, Paul Gratz, and Narasimha Reddy. An fpga-based hybrid memory emulation system. In *31st International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2021a. doi: 10.1109/FPL53798.2021.00039.

Fei Wen, Mian Qin, Paul Gratz, and Narasimha Reddy. Openmem: Hardware/software cooperative management for mobile memory system. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 109–114, 2021b. doi: 10.1109/DAC18074.2021.9586186.

Siheng Xiong, Yuan Yang, Faramarz Fekri, and James Clayton Kerce. Tilp: Differentiable learning of temporal logical rules on knowledge graphs. In *The Eleventh International Conference on Learning Representations*, 2022.

Siheng Xiong, Yuan Yang, Ali Payani, James C Kerce, and Faramarz Fekri. Teilp: Time prediction over knowledge graphs via logical reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 16112–16119, 2024.

Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. *arXiv preprint arXiv:1707.06690*, 2017.

Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. Hygcn: A gcn accelerator with hybrid architecture. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 15–29. IEEE, 2020.

Je Yang, Seongmin Hong, and Joo-Young Kim. Fixar: A fixed-point deep reinforcement learning platform with quantization-aware training and adaptive parallelism. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 259–264. IEEE, 2021.

Pengcheng Yao, Long Zheng, Zhen Zeng, Yu Huang, Chuangyi Gui, Xiaofei Liao, Hai Jin, and Jingling Xue. A locality-aware energy-efficient accelerator for graph mining applications. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 895–907. IEEE, 2020.

Haoran You, Tong Geng, Yongan Zhang, Ang Li, and Yingyan Lin. Gcod: Graph convolutional network acceleration via dedicated algorithm and accelerator co-design. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 460–474. IEEE, 2022.

Bingyi Zhang, Hanqing Zeng, and Viktor K Prasanna. Decgnn: A framework for mapping decoupled gnn models onto cpu-fpga heterogeneous platform. In *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 154–154, 2022.

Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. Dgl-ke: Training knowledge graph embeddings at scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 739–748, 2020.

Hongkuan Zhou, Bingyi Zhang, Rajgopal Kannan, Viktor Prasanna, and Carl Busart. Model-architecture co-design for high performance temporal gnn inference on fpga. *arXiv preprint arXiv:2203.05095*, 2022.

Zhaocheng Zhu, Shizhen Xu, Meng Qu, and Jian Tang. Graphvite: A high-performance cpu-gpu hybrid system for node embedding. In *The World Wide Web Conference*, pages 2494–2504. ACM, 2019.

Youwei Zhuo, Chao Wang, Mingxing Zhang, Rui Wang, Dimin Niu, Yanzhi Wang, and Xuehai Qian. Graphq: Scalable pim-based graph processing. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 712–725, 2019.

7. Appendix

7.1. Path-based KGR versus Embedding-based KGR

Graph convolutional neural networks (GCNs) are widely used to tackle multi-hop reasoning tasks Vashishth et al. (2019). Nevertheless, such neural network models' reasoning procedures are known to lack transparency and interpretability. Consequently, there is a tendency to train RL agents to conduct reasoning tasks on KGs by formulating path-finding between entity pairs as a sequential decision making process Chen et al. (2023, 2022). In this work, we focus on RL-based pathfinding instead of traditional embedding-based methods, since embedding-based models, such as TransE Bordes et al. (2013), TransH Wang et al. (2014), and TransR Lin et al. (2015), are not suitable for complex multi-hop relation reasoning tasks.

7.2. Reinforcement Learning for KGR

7.2.1. STATES, ACTIONS, AND REWARDS

Actions At each time step, the set of agent's possible actions $A_t \in A$ consists of current entity e_t's outgoing edges. Formally, the action space for each timestamp can defined as $A_t = \{(r', e') \mid (e_t, r', e') \in G\}$. Given a query (e_s, r, e_t), the agent chooses one of the outgoing edges at each time step, until it gets to the destination or the model's time limit is reached.

States Entities and relations in knowledge graphs are naturally in the form of text (shown in Figure 1). Such raw data has to be processed for the model starting with assigning an embedding to each entity and relation. It is crucial to use an approach that keeps the natural semantics of the raw text data, as well as the relations between entities introduced in the KG; accordingly, we take advantage of light-weight embedding-based algorithms such as TransE Bordes et al. (2013) as the front-end feature extraction layer, as introduced in section 7.1. The result of the mentioned preprocessing is a low dimensional (e.g. 50 dimensions) vector for each entity and relation. Given a query triple (e_s, r, e_d), embeddings $\boldsymbol{e_s}$, $\boldsymbol{r}$, and $\boldsymbol{e_d}$ are calculated through the preprocessing layer. The model then defines the current state of the agent as $\boldsymbol{s_t} = (\boldsymbol{e_t}, \boldsymbol{e_d} - \boldsymbol{e_t})$, which includes both current entity's embedding vector $\boldsymbol{e_t}$, and the vector representing current entity's distance from destination entity $\boldsymbol{e_d} - \boldsymbol{e_t}$.

Rewards There are two types of rewards in RL agent training: global reward and efficiency reward. Global reward r_{Global} is the positive reward the agent receives when it reaches the final target before the time limit is passed. r_{Global} is defined as

$$r_{Global} = \begin{cases} +1 & \text{if } e_T = e_d \\ -1 & \text{otherwise} \end{cases} \tag{4}$$

where e_T is the entity that the agent ends up at and e_d is the target entity. The efficiency reward is also calculated as

$$r_{Efficiency} = \frac{1}{length(p)} \tag{5}$$

where p is the path that the agent finds during each episode. It is shown that short paths tend to provide more reliable reasoning evidence and improve the efficiency of reasoning Xiong et al. (2017).

Table 2: Compare RL task between KGR and OpenAI Gym.

	OpenAI Gym	**KGR**
Action space	Small	Large
State dimension	Low∼High	High
Action-state sparsity	Low	High
Network size	Small	Large

7.2.2. SUPERVISED POLICY LEARNING

The number of different relations in a knowledge graph could sometimes be rather large. As an example, the total number of different relations in FB15K-237 Nguyen et al. (2017) is 237, which is the size of action space for the model. To resolve this problem, previous work Xiong et al. (2017) suggests performing a supervised training on agent, to provide it with some initial trajectories with positive rewards; this imitation learning Hussein et al. (2017) style of training significantly increases agent's learning speed. As a result, our algorithm first conducts a bi-directional breadth first search (BFS) to generate several sample paths to guide the agent as a teacher. After the mentioned step, the agent will interact with the environment by itself and choose the next time step action based on its policy.

7.3. Comparison Between KGR with Traditional RL

In practice, one big challenge of KGR is that the relation set can be quite large. For a typical KG, the RL agent is often faced with hundreds (thousands) of possible actions. Table 2 compares the RL task of KGR with traditional OpenAI Gym tasks. In other words, the output layer of the policy network of ten has a large dimension. Due to the complexity of the relation graph and the large action space, conventional RL training through trial and error would lead to very poor convergence properties for the RL model. After a long training, the agent fails to find any valuable path. Directly training the agent to pick actions from the original action space can be a difficult task. AlphaGo first trains a supervised policy network using expert moves. In our case, the supervised policy is trained with a randomized breadth-first search (BFS).

For each relation, we use a subset of all the positive samples (entity pairs) to learn the supervised policy. For each positive sample (e_{source}, e_{target}), a two-side BFS is conducted to find same correct paths between the entities. We update the parameters for each path to maximize the expected cumulative reward using Monte-Carlo policy gradient.

7.4. KGAccel **network structure and resource utilization**

The policy network (PolicyNN) we choose to deploy on FPGA is a three-layer multi-layer connection(MLP). The network structure is shown in Table 4. Specifically, **A** represents the total number of relations in the knowledge graph, which is also action space size. The data precision that we choose for each layer is 16-bits fixed-point number. Table 3 summarizes KGAccel resource utilization on two different FPGA platforms. There are several points in Table 3 that need to be noticed. The first is that, as is shown in Figure 1.(b), for the host CPU part, technically speaking ARM Cortex is directly integrated into Zynq FPGA. The host CPU for ZCU104 means the ARM core will load the entity embedding vector into the FPGA kernel. The second point is the definition of Parallelism

Table 3: FPGA Resource Utilization on Xilinx U280 and ZCU104

	Alveo U280					**UltraScale+ ZCU104**				
Host CPU	Intel Xeon 6226					ARM Cortex-A53				
Batch Size	8					2				
Parallelism	4x4					2x2				
Component	LUT	FF	BRAM	URAM	DSP	LUT	FF	BRAM	URAM	DSP
PEs	736.8K	405.1K	812	0	4023	164.2K	114.7K	213	18	501
KGE	171.9K	106.6K	0	850	0	20.9K	14.6K	64	72	0
PCIe DMA	74.1K	68.5K	94	0	0	-				
Optimizer	56.9K	75.7K	0	0	279	15.2K	17.9K	0	0	47
MPSoC	-					14.5K	16.1K	0	0	0
HBM	4.2K	3.4K	2	0	0	-				
Others	7.9K	3.2K	0	0	0	5.6K	3.1K	0	0	0
Total	1051.8(80.6%)	662.5(25.4)	906(44.9%)	850(88.5%)	4302(47.6%)	220.4K(95%)	179.5K(39.0%)	277(88.7%)	90(93.5%)	548(31.7%)
Frequency	200 MHz					180 MHz				
Latency	2953 cycle					8711 cycle				
Power	28.8 Watt					5.4 Watt				

in Table 3. As is shown in Figure 3, each processing element (PE) IP includes **T** systolic array IPs, and a single layer includes **T** PE IP. Here 8×8 means that there will be 8 PE IPs for a single layer, and each PE IP includes 8 systolic array IPs. The third point is that it is obvious to notice that some component is unique to specific FPGA. For example, ZCU104 have MPSoC IP but not HBM. Also, since for ZCU104, the communication between ARM CPU and FPGA is via ARM smart interconnect inside MPSoC IP, therefore the PCIe DMA IP is not necessary. The last but not least point that we want to clarify is that the latency in Table 3 just represents a single PolicyNN passing time, not the whole KGR time.

7.5. Detailed KGAccel Reasoning Results Analysis

To evaluate the reasoning provided by the KGAccel, we explore an standard KGR task, link prediction, which aims at predicting missing relations among entities of a KG and is widely studied in literature, primarily tackling the problem of KG incompleteness. The model is trained on NELL-995 and FB15K-237, which are the main available datasets for link prediction. We perform reasoning tasks on a number of relations within each dataset, while making sure that the selected tasks include various domains of entities and relations. For each task, all of the instances of its corresponding relation and its inverse are removed from the dataset, and are treated as a queries for the task. The

Table 4: Policy Network (PolicyNN) Structure

	Layer Type	**# of parameter**	**# of output features**
0	Input		192
1.0	Fully-connected(FC1)	192x512	512
1.1		ReLU activation	
2.0	Fully-connected(FC2)	512x512	512
2.1		ReLU activation	
3.0	Fully-connected(FC3)	512xA	A
3.1		Softmax	

Table 5: Sample reasoning paths found by the RL model

Relation	**Sample reasoning paths**
organizationHiredPerson	worksForInverse personLeadsOrganizationInverse mutualproxyfor
athleteHomeStadium	athletePlaysForTeam → teamHomeStadium athleteledSportsTeam → teamPlaysInCity → stadiumLocatedInCityInverse athletePlaysForTeam → teamPlaysSport → sportUsesStadium
filmLanguage	filmIsInCountry → countryLanguage filmProducedBy → personLanguage filmGenre → tvProgramGenreInverse → tvProgramLanguage
personBornInLocation	personBornInCity personGraduatedFromUniversity → atLocation
filmDirectedBy	awardWonByInverse filmWrittenBy → filmCrewRole → filmCrewRoleInverse

training is separated into 3 stages: supervised policy learning, retraining with rewards and testing with updates. During the first stage, a supervised policy is trained using bi-directional BFS. The RL agent is then retrained during the second stage, with rewards given based on its performance on each episode, with the previously learned supervised policy used as a teacher to help if agent fails. During the last stage, the agent is tested and updated, without having access to the teacher anymore. The training is conducted for a maximum of 500, 300 and 500 episodes for each stage, respectively, with each episode containing one task.

Figure 5 depicts success rate and moving success rate of the agent for its training. Success rate is determined by ratio of current number of successful episodes to the total number of episodes, while moving success rate is calculated as the ratio of current number of successful episodes to current number of episodes. Figure 6 shows the mean average precision (MAP) results, used as an evaluation metric to compare our model with two main embedding methods, TransE and TransR. We present more detailed analysis of reasoning accuracy in Appendix Section **??**.

Our approach has a significant improvement in performance comparing to embedding method baselines on both NELL-995 and FB15K-237, validating the capability of our RL model in reasoning. Difference between performances of the models depends on the task. Specifically, the number of paths available between entities for each task can affect the learning proficiency of our model; the less paths available between entities, the less our model can find evidence in KG to reason with. Table 5 contains several paths found by the model for various tasks from NELL-995 and FB15K-237. It can be seen that the reasoned paths match with their respective relations, for which each task is designed and performed.

It is also worth noting that the accuracy results that we achieve on NELL-995 tasks are for the most part higher than FB15K-237 tasks. This is a result of the fact that paths in FB15K-237 are in most cases consisted of several relations, whereas paths in NELL-995 are much shorter, with many including no more than one relation.

7.6. Related Works

7.6.1. HARDWARE ACCELERATION OF GRAPH APPLICATION

Graph data-related applications have drawn considerable attention recently. Domain-specific accelerator (DSA) targeting graph applications has shown great success in the top system and architecture conferences in recent years. Previous works focus on accelerating graph analysis applications, such as GNN Yan et al. (2020); Li et al. (2021); You et al. (2022); Huang et al. (2022); Li et al. (2022); Geng et al. (2020); Zhou et al. (2022); Zhang et al. (2022); Lin et al. (2022), and graph analysis tools, such as graph mining Zhuo et al. (2019); Song et al. (2018); Chen et al. (2021); Talati et al. (2022); Yao et al. (2020). Specifically, work Yan et al. (2020) first proposes the design of DSA targeting GNN training. Geng et al. (2020); You et al. (2022) focus on accelerating GNN on the FPGA platform with load balance and sparsity reduction. Huang et al. (2022) proposes using process in memory (PIM) to accelerate GNN by mapping vertex into crossbar-based architecture. The latest work Li et al. (2022) also explores graph partitioning and acceleration on cloud-based FPGA platforms. Mentioned GNN-related accelerators mainly focus on optimizing vertex to vertex combination and aggregation operations, setting aside the vertex to relation combination. Acceleration of knowledge graph learning application based on CPU-FPGA heterogeneous platform is first proposed by Zhou et al. (2022). These works target graph learning applications such as vertex clustering or classification. KGAccel on the other hand, is the first FPGA acceleration platform targeting the reasoning task, which is inferring not-existing knowledge from existing datasets.

7.6.2. REINFORCEMENT LEARNING ACCELERATION

Integrating reinforcement learning (RL) into system and architecture designs has been the focus of many studies recently Rothmann and Porrmann (2022); Li et al. (2019); Lin et al. (2020). Specifically, conducting hardware-software co-design trying to map different RL algorithms on FPGA shows large potential to increase the speed of agent's training process. Both on-policy RL algorithms, such as PPO Schulman et al. (2017) and A3C Mnih et al. (2016), and off-policy algorithms, such as DDPG Lillicrap et al. (2015), have been accelerated on CPU-FPGA heterogeneous computing platform Cho et al. (2019); Meng et al. (2020); Yang et al. (2021). However, compared to the knowledge graph pathfinding problem that we discussed in this work, such OpenAI Gym tasks have very limited action space, making the policy network training process much easier. Besides, previous works Rothmann and Porrmann (2022) tend to maintain the RL environment on the CPU, which results in CPU-FPGA communication overhead. Since KGR involves frequent interactions between the agent and environment, it is sensible to maintain the RL environment on the FPGA kernel to reduce the data movement overhead. Consequently, our work KGAccel tries to maintain then knowledge graph triples (i.e., environment) on-chip and add an imitation learning teacher to enhance the RL agent's training process.

7.6.3. KNOWLEDGE GRAPH REASONING

Previous KGR acceleration works mostly work on graph embedding-based algorithms such as TransE Bordes et al. (2013) and RotatE Sun et al. (2019). Both CPU Zhu et al. (2019); Boschin (2020); Lerer et al. (2019) and GPU Zheng et al. (2020) acceleration works of graph embedding have been previously proposed. Recently RL RL-based path finding algorithms Xiong et al. (2017); Das et al. (2017); Lin et al. (2018); Shen et al. (2018) have shown great success over multi-hop KGR tasks. The mentioned designs try to train an agent with a deep neural network (DNN) based policy to find the potential relation path based on the existing knowledge. Although such algorithms show significant improvements in accuracy, their training execution time overhead is very high. Accordingly, KGAccel proposes the first domain-specific accelerator targeting RL-based KGR algorithms.