NGDB-Zoo: Towards Efficient and Scalable Neural Graph Databases Training

Zhongwei XIE[1] **Jiaxin BAI**[2] **Shujie LIU**[3] **Haoyu HUANG**[1]
Yufei LI[1] **Yisen GAO**[1] **Hong Ting TSANG**[1] **Yangqiu SONG**[1*]
[1]The Hong Kong University of Science and Technology
[2]Hong Kong Baptist University [3]Microsoft Research Asia

Abstract

Neural Graph Databases (NGDBs) support complex logical reasoning over incomplete knowledge structures, yet their training efficiency and expressivity are constrained by rigid query-level batching and structure-only embeddings. We present NGDB-Zoo, a unified framework that resolves these bottlenecks by synergizing operator-level training with semantic augmentation. By decoupling logical operators from query topologies, NGDB-Zoo transforms training loops into dynamically scheduled data-flow executions, enabling multi-stream parallelism and achieving a $1.8\times$-$6.8\times$ average throughput compared to baselines. Furthermore, we formalize a decoupled architecture to integrate semantic priors from pre-trained text embeddings without triggering I/O stalls or memory overflows. Experiments on six benchmarks, including *ogbl-wikikg2* and *ATLAS-Wiki*, show that NGDB-Zoo scales dense query-embedding training to million-entity graphs while maintaining competitive filtered MRR.

1 Introduction

Neural Graph Databases (NGDBs) have emerged as a powerful paradigm for complex logical reasoning over large-scale, incomplete knowledge structures [44, 7, 46, 39]. By embedding existential first-order logic queries into continuous geometric spaces, these systems enable approximate retrieval and inductive inference beyond symbolic traversal [42, 3]. Despite their promise, the transition from theoretical models to production-ready query engines is hindered by a fundamental tension between training efficiency and representational expressivity.

On the training efficiency front, the primary challenge is **computational inefficiency arising from topology-homogeneous batching.** Existing training frameworks batch queries at the structural level, constraining each mini-batch to queries that share isomorphic logical structures (e.g., 2i) [41, 3, 43]. This constraint causes severe hardware under-utilization when processing diverse query mixtures, as GPU kernels are fragmented by the high entropy of real-world query workloads. As link density drops in large-scale graphs, this efficiency bottleneck becomes a critical barrier to convergence. Table 1 further details the shortcomings of existing systems in this regard.

The efficiency challenge is exacerbated when one seeks to incorporate semantic priors, namely, the embedding features of nodes and relations, into the training and inference pipeline. While most prior work places strong emphasis on the structural and relational aspects of knowledge graphs, we examine how semantic entity features complement structural information for neural graph database modeling on real-world KGs [55, 53, 29, 11, 16, 49]. Treating entities as atomic tokens may suffice for dense knowledge graphs, but richer semantic representations can provide complementary signals for KG reasoning. Yet existing KG reasoning frameworks remain largely structure-only, leaving a significant

*Correspondence to: yqsong@cse.ust.hk.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

gap. Moreover, latent embedding representations can encode not only textual and numerical features but also images, audio, and video, suggesting a possible direction for future multimodal NGDB systems. These observations collectively motivate us to build a fast and scalable system capable of **integrating semantic priors without incurring prohibitive system-level overhead.**

To address these challenges, we present **NGDB-Zoo**, a unified framework designed for high-throughput, semantically-aware neural graph reasoning. Our key contribution is a training runtime that makes operator-level execution practical for dense query-embedding models by preserving gradient dependencies, batch-level loss alignment, and memory safety under dynamically scheduled query DAGs. Concretely, NGDB-Zoo decomposes complex queries into a Directed Acyclic Graph (DAG) of typed operators, enabling cross-query operator fusion, dependency-aware scheduling, and eager reclamation of intermediate tensors. To incorporate semantic priors without sacrificing this efficiency, we introduce a decoupled integration mechanism that fuses frozen, pre-trained text embedding derived semantic anchors with trainable structural embeddings through a lightweight projection layer, backed by a memory-hierarchy-aware loading strategy that adapts to available VRAM. We validate NGDB-Zoo across six benchmarks, including massive knowledge graphs such as *ogbl-wikikg2* and *ATLAS-Wiki*, demonstrating $1.8\times$ average throughput over SMORE ($6.8\times$ over KGReasoning) alongside competitive reasoning accuracy. Our main contributions are:

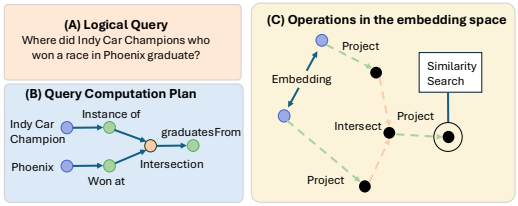


Figure 1: **Task Illustration.** Query embedding methods aim to answer multi-hop logical queries (A) by avoiding explicit graph traversal and executing the query directly in the embedding space by following the query computation plan (B). Operators are executed in the embedding space (C).

Contribution 1: Autograd-Safe Operator-Level Training Runtime. We shift batching granularity from query-level isomorphism to individual logical operators while preserving training semantics through dependency-aware scheduling, cross-query fusion, and eager tensor reclamation.

Contribution 2: Decoupled Structural Semantic Integration. We combine frozen text embeddings with trainable structural representations using a lightweight projection and storage-aware access, and evaluate their accuracy and training costs on standard KG benchmarks.

Contribution 3: Large-Scale Neuro-Symbolic Benchmarking. We provide rigorous evaluation across six benchmarks at varying scales, characterizing the efficiency expressivity trade-off and offering a technical roadmap for adaptive handling of non-stationary query distributions.

2 Related Work

Neural Graph Databases Reasoning. Complex Query Answering (CQA) extends relational link prediction to multi-hop logical operations. Early methods [64, 50, 47] focused on simple relational patterns. More recent geometric embedding models [62] represent queries as points [22], bounded regions [42, 69, 56], or probabilistic distributions [41, 65] to capture topological constraints and uncertainty. Complementary approaches utilize hyperbolic projections [14] or sequential encoding [3]. Recently, standalone models have evolved into integrated Neural Graph Databases (NGDBs) [7, 44, 25, 24, 70, 54, 13], supporting neuro-symbolic retrieval at scale and providing the state-space foundation for autonomous planning in Agentic NGDBs [5, 31, 59, 61]. NGDBench evaluates neural graph data management under noise, incomplete information, and graph updates [34]. Despite their promise, the large, structurally diverse query space motivates efficient online sampling and operator-level training.

Training Optimization for Graph Embedding Models. GNNs [57, 19, 23, 17, 33, 6] use neighbor sampling or subgraph batching, sequence models [38, 63] group sequences by length, and dense QE systems such as KGR [40], SMORE [43], and SQE [3] batch link-prediction triples or preserve structure-level execution boundaries. Several works relax this rigidity. MPQE [15] constructs block-diagonal graphs but remains template-specific; GNN-QE [72] batches operators but uses full-graph projection. CQD [1] relies on continuous optimization, while ULTRAQ [18] utilizes execution-based message passing, both of which have fundamentally different scheduling needs. ATOM [71] introduced operator-level batching for stateless KG *serving*, where no gradient graph

must be preserved. Compiler frameworks optimize captured tensor programs; KGCompiler [35] specializes such optimization to logical-query graphs. These compiler optimizations complement NGDB-Zoo's stateful training runtime, which combines cross-query batching with autograd-aware memory management, online sampling, and loss alignment (Table 1; Appendix A).

Pretrained Embeddings for Structured Knowledge. Besides dense-retrieval-based methods, recent work has explored combining pretrained language models with knowledge graphs [28, 66]: KEPLER [53] leverages KG embeddings as supervision signals to train language models. SimKGC [52] uses contrastive learning to align entity descriptions with KG structure. Some works [20, 18, 67, 36, 27, 45, 9, 10, 30] build pre-trained GNN foundation model for reasoning tasks on KGs. KGPFN uses relation-specific examples for in-context prediction across knowledge graphs [21]. Recent state-of-the-art encoders [51] like Qwen3-Embedding [68] and BGE-Embedding [58] provide rich semantic similarity signals. However, integrating high-dimensional PLM outputs into complex multi-hop reasoning often triggers severe I/O stalls and memory bottlenecks. Fusing these semantic representations with multi-hop reasoning models efficiently remains an open challenge.

Table 1: Capability comparison between NGDB-Zoo and related systems. *Phase* — Training (Tr), Serving (Sv) or Inference (Inf); *Large Scale* — evaluated on KGs with $|\mathcal{E}| \geq 400$K. ✓ = fully supported, △ = partially supported, × = not supported. For MPQE, △ in Cross-Str. Batching indicates that batching is restricted to a single query template per batch. For SMORE, △ in Autograd Mem. indicates that CPU-GPU offloading is supported but autograd-aware tensor reclamation is not.

System	Phase	Cross-Str. Batching	Autograd Mem.	Online Samp.	Sem. Aug.	Large Scale
ATOM [71]	Sv	✓	×	×	×	✓
MPQE [15]	Tr	△	×	×	×	×
GNN-QE [72]	Tr	✓	×	×	×	×
CQD [1]	Inf	×	×	×	×	×
ULTRAQ [18]	Tr	×	×	×	×	×
KGR [40]	Tr	×	×	×	×	×
SMORE [43]	Tr	×	△	✓	×	✓
SQE [3]	Tr	×	×	×	×	×
NGDB-Zoo	**Tr**	✓	✓	✓	✓	✓

3 Problem Formulation

3.1 Complex Logical Queries

Let $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$ be a knowledge graph, where $\mathcal{E}$ is the set of entities, $\mathcal{R}$ is the set of relations, and $\mathcal{T} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ represents the observed fact triples. We consider the task of answering complex logical queries q that can be expressed in Existential First-Order (EFO) logic, following the form:

$$q[V_?] = \exists V_1, V_2, \ldots, V_k : \phi(V_?, V_1, \ldots, V_k) \tag{1}$$

where $V_?$ is the target variable, $\{V_1, \ldots, V_k\}$ are existentially quantified bound variables, and ϕ combines atomic formulas with conjunction, disjunction, and negation.

Neural Graph Databases (NGDBs) execute complex logical queries over these structures. We evaluate 14 distinct query patterns involving projection (p), intersection (i), union (u), and negation (n). These patterns ($1p$, $2p$, $3p$, $2i$, $3i$, pi, ip, $2u$, up, $2in$, $3in$, pin, pni, inp) represent different topological constraints in the reasoning subgraph, challenging the model to perform multi-hop reasoning.

3.2 The Task: Predictive Query Answering

Unlike standard database queries that rely on a complete $\mathcal{G}$, the **Predictive Query Answering** task requires identifying answers from an incomplete graph where many true triples are missing during the training and inference time.

Answer Set. For a query q, its denotation set $\mathcal{A}_q \subseteq \mathcal{E}$ contains all entities satisfying q under the ground-truth graph $\mathcal{G}_{full}$. We distinguish *Direct Answers* ($\mathcal{A}_q^{obs}$), obtained by symbolic execution on the observed training graph $\mathcal{G}_{train}$, from *Predictive Answers* ($\mathcal{A}_q^{miss} = \mathcal{A}_q \setminus \mathcal{A}_q^{obs}$), which are valid under $\mathcal{G}_{full}$ but remain latent in $\mathcal{G}_{train}$ due to missing triples. This distinction separates answers observed during training from answers requiring prediction of missing graph facts.

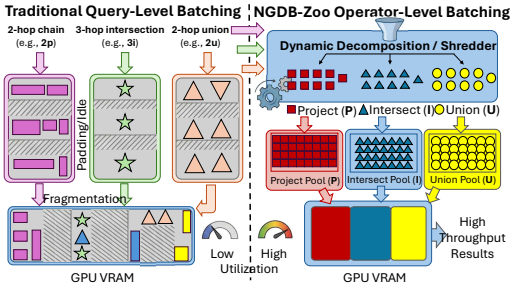


Figure 2: **Overcoming Topological Rigidity. (Left):** Query-level batching incurs high fragmentation overheads by segregating distinct query structures. **(Right):** NGDB-Zoo aggregates atomic operators into homogeneous Operator Pools for unified execution, eliminating structural constraints and saturating GPU cores.

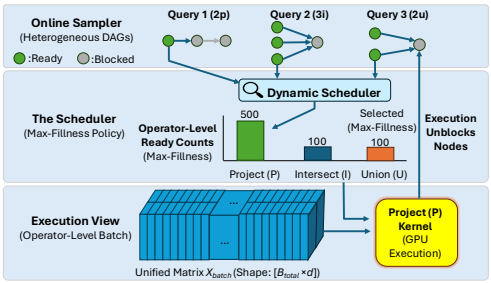


Figure 3: **Max-Fillness Dynamic Scheduling.** By prioritizing operator pools with the highest ready workload, the scheduler improves effective throughput (Appendix K validates this choice against FIFO and Random baselines). This dynamic selection unblocks dependent nodes and produces a continuous training stream.

Objective Function. The objective is to learn a scoring function $f : \mathcal{Q} \times \mathcal{E} \to \mathbb{R}$, where $\mathcal{Q}$ is the query space. For each candidate entity $e \in \mathcal{E}$, $f(q, e)$ measures the likelihood that e is a valid answer to q. We therefore require valid answers to be scored above invalid entities; equivalently, the ranking constraint is:

$$\forall e \in \mathcal{A}_q, e' \notin \mathcal{A}_q \implies f(q, e) > f(q, e') \tag{2}$$

The model must generalize beyond memorized paths by mapping joint entity representations into a target logical manifold $\mathcal{D}$ via Ψ_θ, and computing proximity between the final query representation $\mathbf{q} \in \mathcal{D}$ and entity representations in the same space.

Joint Structural-Semantic Representation. We define a joint representation setting where the entity embedding space is a composite manifold $\mathcal{H}_{joint} = \mathcal{H}_{str} \times \mathcal{H}_{sem}$. The structural subspace $\mathcal{H}_{str} \subseteq \mathbb{R}^{d_s}$ captures endogenous graph topology through trainable embeddings $\mathbf{h}_i$, while the semantic subspace $\mathcal{H}_{sem} \subseteq \mathbb{R}^{d_m}$ encapsulates exogenous linguistic priors via a frozen pre-trained text encoder, Φ. For each entity $e_i \in \mathcal{E}$, its joint representation is $\mathbf{x}_i = [\mathbf{h}_i \oplus \Phi(d_i)] \in \mathcal{H}_{joint}$.

To integrate these features into diverse reasoning frameworks, we interpret the joint signals as the initialization parameters defining the entity representation in the target logical manifold $\mathcal{D}$. We employ a trainable mapping network $\Psi_\theta : \mathcal{H}_{joint} \to \mathcal{D}$ to project the joint embeddings into the manifold $\mathcal{D}$, i.e., $\mathbf{z}_i^0 = \Psi_\theta(\mathbf{x}_i)$, where $\mathbf{z}_i^0$ denotes the model-specific initial entity representation. These representations then serve as the input to the subsequent sequence of logical operators, incorporating both learned topology and static semantic knowledge into query execution.

4 Method

To transition NGDBs from theoretical models to a feasible training system, we must overcome two system-level bottlenecks: (i) the structural fragmentation caused by query-level batching, and (ii) the I/O stalls introduced by integrating high-dimensional semantic priors. NGDB-Zoo addresses both by the following design and implementation. The complete training procedure is given in Appendix D.

4.1 Operator-Level Batching

Standard query-level batching groups queries with identical topological structure, such as batching all 2-hop chains together. This is efficient for homogeneous workloads, but it fragments realistic NGDB training streams that mix paths, intersections, unions, and negations. Under a uniform mixture over $|\mathcal{T}|$ query templates, the effective batch size seen by any one operator kernel is reduced to approximately $|\mathcal{T}|^{-1}$, leaving many launches under-filled. To resolve this bottleneck, NGDB-Zoo instead moves the batching boundary from whole queries to typed logical operators as shown in Figure 2. The training runtime must still satisfy three invariants: intermediate tensors remain available to autograd, final query embeddings are aligned with positive and negative entity samples before the loss is formed, and scheduler decisions do not change the mathematical objective. Appendix J details the batching comparison.

Graph Decomposition and Representation. We define a logical query q as a directed acyclic graph (DAG) $G_q = (V, E)$, where V is the set of atomic operators (e.g., Project, Intersect, Union) and E is the set of data dependencies. Formally, a complex query such as a 2-way intersection $q = \phi_\cap(\phi_p(e_1, r_1), \phi_p(e_2, r_2))$ is decomposed into projection nodes feeding a central intersection node. This decomposition allows us to aggregate operators v across query structures into global execution pools $\mathcal{P}_\tau$, where τ denotes the operator type.

Dynamic Scheduling. The core of our training engine is a dynamic scheduler that optimizes GPU occupancy. At any simulation step t, let $\mathcal{R}_t$ be the set of all ready operators (those whose dependencies are satisfied). The scheduler groups $\mathcal{R}_t$ by operator type τ and selects a target type τ^* to execute. We employ a *Max-Fillness* policy to maximize hardware utilization. We define the fillness ratio $\rho(\tau)$ as:

$$\rho(\tau) = \frac{|\{o \in \mathcal{R}_t : \text{type}(o) = \tau\}|}{B_{\max,\tau}} \tag{3}$$

where $B_{\max,\tau}$ is the maximum efficient hardware batch size for operator type τ. The scheduler selects $\tau^* = \arg\max_\tau \rho(\tau)$. Appendix K compares this policy with FIFO and Random.

Batched Execution. Upon selecting the target type τ^*, the execution engine performs a *Cross-Query Operator Fusion*. Let $\{o_1, o_2, \ldots, o_k\}$ be the set of operators in $\mathcal{R}_t$ with $\text{type}(o_i) = \tau^*$. Regardless of their original query contexts, their input embeddings $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ are coalesced into a contiguous memory block to form a unified batch matrix $\mathbf{X}_{batch} \in \mathbb{R}^{k \times d}$. This unified representation allows the engine to invoke a single high-performance GPU kernel: $\mathbf{Y}_{batch} = \text{Kernel}_{\tau^*}(\mathbf{X}_{batch}; \theta_{\tau^*})$, where θ_{τ^*} represents the shared parameters for the operator type. This mechanism effectively mitigates kernel launch overhead and branch divergence across heterogeneous query structures, ensuring that the hardware operates near its peak performance.

4.2 Vectorized Objective Optimization

Building on the QueryDAG abstraction, we implement a multi-phase optimization strategy that restructures the entire training lifecycle to maximize system throughput.

Vectorized Logit Formulation. Beyond structural scheduling, the computation of the objective function often suffers from high latency due to iterative embedding lookups. After all root query embeddings are produced, the scheduler restores their mini-batch order before aligning them with positive and negative samples. For a mini-batch of B queries, let $\mathbf{Q} \in \mathbb{R}^{B \times d}$ denote the final query embeddings produced by the QueryDAG, and let $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times d}$ denote the candidate entity embedding matrix. For query q_i, positives $\mathcal{P}_i$ are entities in the training denotation set obtained by symbolic execution on $\mathcal{G}_{train}$, while negatives $\mathcal{N}_i$ are sampled corrupted entities from $\mathcal{E} \setminus \mathcal{P}_i$. Instead of scoring these candidates one by one, we gather their rows from $\mathbf{E}$ and evaluate the corresponding scoring function in batch. The training objective is:

$$\mathcal{L} = -\sum_{i=1}^{B} \big[\sum_{p \in \mathcal{P}_i} \log \sigma(f(\mathbf{Q}_i, \mathbf{E}_p)) + \sum_{n \in \mathcal{N}_i} \log \sigma(-f(\mathbf{Q}_i, \mathbf{E}_n))\big], \tag{4}$$

where f is the scoring function of the underlying query encoder and $\sigma(\cdot)$ is the sigmoid function.

Efficient Tensor Indexing. To further alleviate CPU-GPU synchronization bottlenecks, we introduce a *Precomputed Indexing* mechanism. By pre-calculating the memory offsets for entity embeddings within the QueryDAG, we employ asynchronous scatter/gather operations. This reduces redundant tensor concatenation and splitting, keeping the hot path on device when embeddings are resident on GPU and minimizing CPU-side indexing overhead.

4.3 Efficient System Architecture and Optimization

To facilitate training on large-scale data with high-throughput requirements, we introduce system optimizations targeting two primary bottlenecks: memory-intensive intermediate activation management and sub-optimal hardware utilization during complex query reasoning.

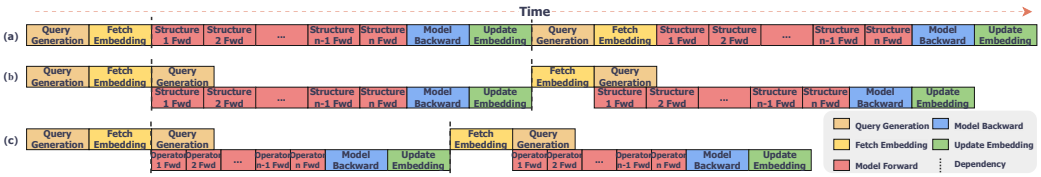


Figure 4: **Evolution toward High-Throughput Asynchronous Pipelining.** Training methods. (a) represents the naive training scheme; (b) shows the pre-sampling and pre-fetching optimization; (c) NGDB-Zoo's operator-level approach breaks these constraints to enable fully overlapping stages, creating a dense execution stream that maximizes hardware saturation.

Proactive Memory Management via Reference Counting. To enable training with larger batch sizes, we implement eager reference counting for fine-grained resource reclamation. For any intermediate tensor T generated within a QueryDAG, let $\mathcal{V}_{\text{desc}}(T)$ denote the set of descendant operators that consume T as input. The scheduler maintains a dynamic reference counter. A scheduler-owned reference to T is released if and only if all dependent forward operations have been dispatched:

$$\text{ReleaseRef}(T) \iff \sum_{v \in \mathcal{V}_{op}} \mathbb{I}(T \in \text{input}(v) \wedge v \text{ not dispatched}) = 0. \tag{5}$$

By strictly enforcing this reclamation policy, our system significantly lowers the memory pressure compared to query-scoped allocation, enabling larger batch sizes training without triggering out-of-memory errors on modern hardware. Appendix L reports measurements of kernel launches, step time, CPU sampling, and peak memory.

Asynchronous Parallelism and Pipelining. The memory efficiency gains from reference counting serve as a foundation for high-concurrency execution. We implement a decoupled execution pipeline characterized by two key parallelization strategies:

Multi-stream Operator Execution: We leverage independent CUDA streams to parallelize disjoint operator types within the QueryDAG. This allows for the simultaneous execution of compute-bound and memory-bound kernels, effectively masking hardware latency. *Heterogeneous Pipelining:* To handle massive graphs exceeding GPU capacity, we adopt an asynchronous CPU-GPU pipelining strategy as illustrated in Figure 4. Feature and structure embeddings are stored in host memory (CPU) and asynchronously fetched to pinned memory before scatter-gathering to the GPU. To mitigate the bottleneck of CPU-side data preparation, we use a consumer-producer pipeline. While the GPU executes current operator batches, the CPU concurrently performs asynchronous sampling for subsequent queries, ensuring a continuous data flow to the accelerators.

Efficiency-oriented Intersect and Union Operator. As shown in Figure 5, unlike structural operators with fixed input degrees (e.g., *Projection* or *Negation*), the *Intersection* ($\cap$) and *Union* ($\cup$) operators encounter variable-cardinality inputs, which typically leads to fragmented memory allocation and sub-optimal GPU utilization. To handle the computational complexity of the Intersect and Union Operator, we propose a batching strategy based on input cardinality. Let $\Phi \in \{\mathcal{O}_{int}, \mathcal{O}_{un}\}$ denote a set-based operator acting on an arbitrary number of input embeddings $\mathcal{S} = \{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$. We define the computation over a query batch as a partition into equivalence classes $\{\mathcal{C}_k\}_{k \in \mathcal{K}}$, where each class $\mathcal{C}_k$ clusters operations with identical input cardinality $k = |\mathcal{S}|$:

$$\mathcal{C}_k = \{\phi_i \in \text{Batch} \mid \text{card}(\text{input}(\phi_i)) = k\}. \tag{6}$$

The global operation is then reformulated as a sequence of vectorized executions: $\mathcal{O}_\Phi = \bigoplus_{k \in \mathcal{K}} \text{VecExec}(\mathcal{C}_k)$, where $\text{VecExec}(\cdot)$ denotes a synchronized operation on a k-depth stacked matrix. This grouping mechanism optimizes GPU memory utilization and maximizes throughput by reducing fragmented memory allocations.

Online Data Sampling and Test Time Optimization. For NGDBs on large-scale graphs, the query space is effectively infinite, making the storage of pre-computed datasets prohibitive. Consequently, we employ an online data sampling system that generates queries dynamically during training. Beyond eliminating storage overhead, this approach exploits temporal locality in query streams. By adjusting the sampling distribution on-the-fly based on query difficulty, the system can optimize for specific query patterns that appear in bursts during multi-run training sessions. This dynamic generation allows for an unlimited stream of diverse samples and enables the curriculum to adaptively prioritize

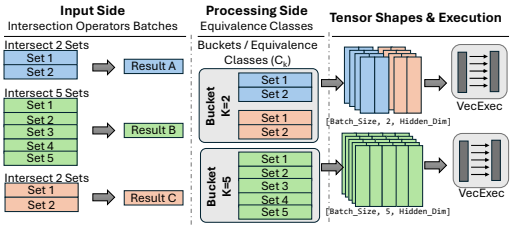


Figure 5: **Vectorized execution for Intersection/Union.** By grouping operations into cardinality-based Equivalence Classes, we eliminate shape misalignment. This enables perfectly aligned **Vectorized Execution**, replacing slow loops with dense matrix operations.

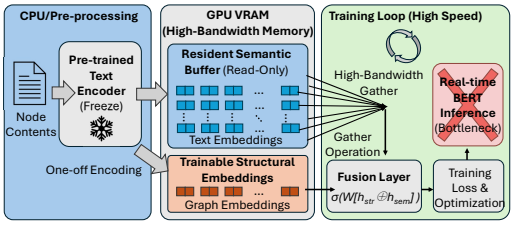


Figure 6: **Decoupled semantic integration strategy.** By relegating the heavy text embedding to an offline pre-computation phase and caching embeddings in GPU memory, NGDB-Zoo achieves minimal-overhead semantic fusion during training.

complex multi-hop structures to stabilize convergence. Appendix H provides the sampling procedure and a controlled diagnostic study on adaptive sampling under non-stationary query distributions.

4.4 Efficient Integration of Semantic Entity Representations

The preceding subsections resolve the structural efficiency bottleneck. We now address the second challenge: enriching entity representations with semantic priors with minimal additional I/O overhead. To augment entities with rich textual knowledge, we integrate high-dimensional semantic priors derived from Pre-trained Text Embeddings. However, joint training with PTEs typically incurs prohibitive overheads, transforming high-throughput graph reasoning into a compute-bound or I/O-bound bottleneck. To resolve this, we propose a *decoupled, GPU-resident integration strategy* that ensures the training loop remains strictly inference-free.

Decoupled Semantic Encoding. Instead of performing real-time feature extraction, which couples the heavy PTE inference with the gradient descent loop, we relegate semantic acquisition to an *offline pre-computation phase*. Given an entity set $\mathcal{E}$, we employ a frozen PTE $\Phi(\cdot)$ to map each entity $e_i \in \mathcal{E}$ to a high-dimensional latent space:

$$\mathcal{H}^{sem} = \{\Phi(T_i) \mid e_i \in \mathcal{E}\} \in \mathbb{R}^{|\mathcal{E}| \times d_l} \tag{7}$$

where T_i denotes the textual description of entity e_i. Crucially, once $\mathcal{H}^{sem}$ is generated, the PTE $\Phi(\cdot)$ is **unloaded** from memory. This decoupling ensures that the subsequent training phase is completely agnostic to the complexity of the text embedding.

Storage-Aware Semantic Access. When $\mathcal{H}^{sem}$ fits within device memory, we register it as a persistent, non-trainable buffer in GPU HBM, reducing semantic access to device-side indexing. For massive KGs where the full semantic table would exceed VRAM, we keep $\mathcal{H}^{sem}$ in CPU memory and reuse the asynchronous pinned-memory staging path described in Section 4.3. For a mini-batch of entity indices $\mathcal{I}$, semantic access is implemented as indexed retrieval from $\mathcal{H}^{sem}$ rather than online PTE inference. In the evaluated GPU-resident setting, this reduces semantic lookup to device-side matrix indexing. To align these high-dimensional priors with the structural embedding space, we employ a lightweight projection $\mathcal{F} : \mathbb{R}^{d_l} \to \mathbb{R}^d$:

$$\mathbf{e}_i^{fused} = \sigma\left(\mathbf{W}_p[\mathbf{h}_i^{str} \oplus \mathcal{F}(\mathbf{h}_i^{sem})] + \mathbf{b}_p\right) \tag{8}$$

where $\oplus$ denotes concatenation and $\mathbf{h}_i^{str}$ is the structural embedding. Structural embeddings and the fusion projection are trainable; the text encoder remains frozen.

5 Experiments

5.1 Experimental Setup

Baselines and Models. Our evaluation follows the training-specific requirements above: preserving ranking accuracy, increasing throughput under heterogeneous query mixtures, controlling memory pressure, and supporting semantic priors. We compare single-hop KG completion runtime against

SQE [3], SMORE [43], Pytorch-Biggraph (PBG) [32], Marius [37], and evaluate end-to-end multi-hop training against KGReasoning [40], SMORE, and SQE. To cover diverse reasoning paradigms, we instantiate NGDB-Zoo with GQE [22] and Q2P [2] as vector-based models, Q2B [42] and BetaE [41] as distribution-based models for probabilistic reasoning, and FuzzQE [12] as a fuzzy-logic model. For semantic experiments, we use Qwen3-Embedding-0.6B [68] and BGE-Base-En-v1.5 [58].
Datasets and Hardware. Experiments use FB15k [8], FB15k-237 [8], NELL [41], FB400k [48, 43], ogbl-wikikg2 [26], and Atlas-Wiki [4]. We use validation/test queries for the first three datasets. For ogbl-wikikg2, queries were randomly sampled using official edge splits. For FB400k, we utilized SPARQL annotations from CWQ as queries, following the data splitting method in SMORE. For ATLAS-Wiki-Triple-4M, queries were sampled using degree-weighted edge sampling. Dataset statistics are listed in Table 5 in Appendix E. Our framework is implemented in PyTorch 2.0. Unless otherwise stated, experiments run on NVIDIA A6000 GPUs (48GB memory) with 40 CPU cores.

5.2 Scalability

GPU memory and end-to-end training speed. We test NGDB-Zoo on all six KGs and report the end-to-end training speed and GPU memory data. As reported in Table 4, NGDB-Zoo achieves substantial throughput gains compared to representative baselines. For instance, on the FB15k dataset using the BetaE model, our system attains 4,477 queries/sec, representing a 7.0× speedup over the SQE framework. While NGDB-Zoo exhibits higher GPU memory footprints compared to SMORE, the memory usage remains well within the capacity of modern accelerators and often stays significantly lower than SQE in complex scenarios like NELL995. The end-to-end gain combines operator batching, vectorization, and asynchronous pipelining; it is not attributable to a single mechanism. Table 3 reports scalability on the evaluated large KGs.

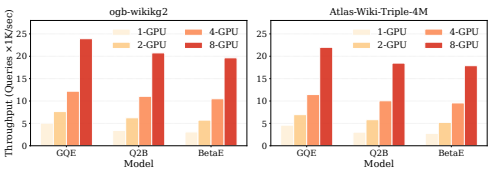


Figure 7: **Multi-GPU Throughput Scalability.** Aggregate training throughput on ogbl-wikikg2 and ATLAS-Wiki-Triple-4M.

Multi-GPU speed up. We report the training speed-up on 1, 2, 4, and 8 GPUs across different methods on the ogbl-wikikg2 and ATLAS-Wiki-Triple-4M in Figure 7. Throughput scaling is sublinear and model-dependent.

Single-hop link prediction 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, PBG and SMORE. 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. For baseline results, we reuse Table 5 from Ren et al. [43]. As shown in Table 2, across all GPU counts, NGDB-Zoo achieves the lowest epoch time, reaching 94s on 8 GPUs versus 121s for SMORE and 419s for PBG.

Table 2: Runtime performance on Freebase (86M entities, 338M edges) on V100. * From Mohoney et al. [37] and Ren et al. [43].

System	Epoch Time (s)			
	1-GPU	2-GPU	4-GPU	8-GPU
Marius [37]*	727	-	-	-
PBG [32]*	3060	1400	515	419
SMORE [43]*	760	411	224	121
NGDB-Zoo (Ours)	**628**	**322**	**181**	**94**

Table 3: **Scalability on Massive KGs.** Filtered MRR (%), aggregate throughput (10^3 Q/sec) on 8×A6000, and peak memory per GPU (GB).

Dataset	Model	MRR	TPut ↑	Mem ↓
FB400k	GQE	35.84	**24.68**	**7.5**
	Q2B	**52.33**	21.55	11.0
	BetaE	50.40	19.97	14.0
ogbl-wikikg2	GQE	32.88	**23.92**	**8.0**
	Q2B	42.01	20.75	11.0
	BetaE	**44.54**	19.65	14.0
ATLAS-Wiki -Triple-4M	GQE	7.31	**22.00**	**10.0**
	Q2B	**9.22**	18.47	12.0
	BetaE	9.01	17.89	15.0

5.3 Predictive performance

Calibration on small KGs. We first evaluate our NGDB-Zoo system for five representational paradigms (BetaE, Q2B, GQE, Q2P, and FuzzQE) on the three small benchmark datasets created in BetaE. For a fair comparison, we test our system against other frameworks under identical online sampler. Although fixed-batch execution is equivalent, mixed-structure versus structure-homogeneous updates can change optimization trajectories; this may explain MRR differences

Table 4: Performance comparison of NGDB-Zoo vs. KGReasoning, SMORE and SQE.

Dataset	Model	MRR (%)				Training queries (Queries/Sec)				GPU Memory (GB)			
		KGR	SMORE	SQE	NGDB-Zoo	KGR	SMORE	SQE	NGDB-Zoo	KGR	SMORE	SQE	NGDB-Zoo
FB15k	BetaE	41.60	40.39	40.82	**43.04**	291	2808	636	**4477**	2.68	**1.16**	8.05	12.95
	Q2B	38.00	41.54	41.08	**42.85**	855	3588	343	**4086**	1.32	**0.96**	6.01	5.07
	GQE	28.00	**30.60**	29.21	30.39	1395	3770	4598	**6271**	**0.76**	1.09	8.03	11.33
	Q2P	-	-	44.30	**46.90**	-	-	832	**1940**	-	-	**4.76**	11.41
	FuzzQE	-	-	42.17	**42.33**	-	-	720	**2973**	-	-	20.14	**8.45**
FB15k-237	BetaE	20.90	19.67	20.92	**21.11**	178	1633	655	**4750**	2.68	**1.14**	6.03	9.81
	Q2B	20.10	**20.42**	19.05	20.19	428	3115	343	**4663**	1.3	**0.93**	6.04	5.09
	GQE	16.30	15.68	16.75	**16.98**	1433	2882	1910	**6034**	**0.75**	1.02	8.01	9.77
	Q2P	-	-	20.12	**22.08**	-	-	842	**1884**	-	-	**4.22**	11.34
	FuzzQE	-	-	21.07	**21.35**	-	-	1350	**2934**	-	-	17.02	**8.54**
NELL995	BetaE	24.60	23.17	23.43	**24.90**	166	1807	154	**4640**	3.42	1.28	35.02	6.81
	Q2B	22.90	21.84	22.21	**23.79**	765	1926	82	**4521**	1.5	**1**	25.04	5.14
	GQE	18.60	17.53	18.02	**18.89**	1224	3691	2959	**6329**	**1.01**	1.17	45.04	7.12
	Q2P	-	-	23.18	**25.65**	-	-	836	**2309**	-	-	9.69	**9.56**
	FuzzQE	-	-	24.31	**25.88**	-	-	2165	**2680**	-	-	**8.07**	8.16
Average Without Q2P/FuzzQE		25.67	25.65	25.72	**26.90**	748	2802	1298	**5086**	1.71	**1.08**	16.36	8.12

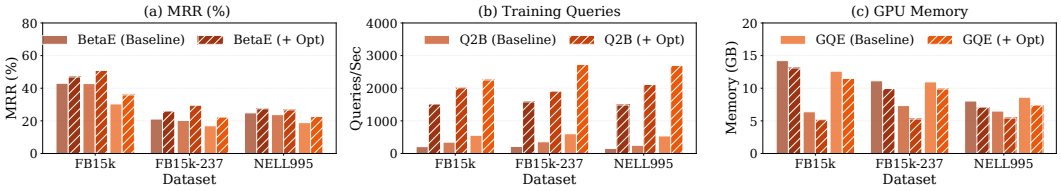


Figure 8: **Impact of Decoupled Semantic Integration. (a)** MRR: structure-only baselines from Table 4 versus semantic-augmented models. **(b, c)** Throughput and peak memory: in-loop frozen encoder inference versus cached semantic access. Offline encoding yields approximately 4×–9× higher throughput and reduces memory by unloading the encoder. Higher is better for (a) and (b); lower is better for (c).

but does not establish causality. To determine the system's throughput, we record both the data loading and sampling durations during this process. As summarized in Table 4,NGDB-Zoo attains MRR comparable to competitive baselines while delivering substantially higher training throughput. Appendix M reports further results on negation queries.

Query answering on large KGs. We further evaluate NGDB-Zoo on three large-scale benchmarks: FB400k, ogbl-wikikg2, and ATLAS-Wiki-4M. As shown in Table 3, NGDB-Zoo successfully scales diverse representational paradigms to graphs with millions of entities. On ogbl-wikikg2, BetaE attains the highest performance with an MRR of 44.54%, while Q2B demonstrates superior expressivity on FB400k, achieving 52.33%. This shows our system maintains robust reasoning even on the massive graph, which contains over 4 million entities.

5.4 Decoupled Semantic Augmentation

Throughput, Memory and Predictive Performance. With GPU-resident semantic tables, we evaluate BetaE, Q2B, and GQE using Qwen3-Embedding (Figure 8) and BGE-Base-En-v1.5 (Appendix Table 8). Encoders receive only entity descriptions, not benchmark queries, answer labels, held-out edges, or split assignments; fusion is trained only on training-graph queries. Pre-training corpus overlap with public descriptions or facts cannot be ruled out. **Throughput:** Offline pre-computation yields approximately 4×–9× higher throughput than repeatedly running the frozen encoder inside the training loop. **Memory:** Relative to in-loop encoding, unloading the text encoder after pre-computation reduces peak GPU memory despite retaining the semantic buffer. **Predictive Performance:** Semantic augmentation improves average MRR in the reported experiments, although gains vary by model and dataset (Appendix N).

6 Conclusion

We presented NGDB-Zoo, a framework resolving the efficiency-expressivity tension in Neural Graph Databases. By adopting operator-level batching, NGDB-Zoo overcomes topological rigidity, achieving competitive training throughput and model-dependent multi-GPU scaling. We also showed that integrating pre-trained text embeddings improves average MRR on the evaluated benchmarks. With model-dependent memory trade-offs, NGDB-Zoo sustains robust reasoning on massive graphs, offering a scalable foundation for next-generation neuro-symbolic systems.

Acknowledgments and Disclosure of Funding

The authors of this paper were supported by the National Key Research and Development Program of China (2025YFE0200500), the ITSP Platform Research Project (ITS/189/23FP) from ITC of Hong Kong, SAR, China, and the AoE (AoE/E-601/24-N), the CRF (No. C6004-25G), the RIF (R6021-20) and the GRF (16205322) from RGC of Hong Kong, SAR, China. We also thank the support of RGC JRFS2526-6S10. Some of the authors are named inventors on a U.S. provisional patent application filed on February 26, 2026, concerning the operator-level batching and decoupled semantic integration methods described in this paper.

References

[1] Erik Arakelyan, Daniel Daza, Pasquale Minervini, and Michael Cochez. Complex query answering with neural link predictors. *arXiv preprint arXiv:2011.03459*, 2020.

[2] Jiaxin Bai, Zihao Wang, Hongming Zhang, and Yangqiu Song. Query2particles: Knowledge graph reasoning with particle embeddings. *arXiv preprint arXiv:2204.12847*, 2022.

[3] Jiaxin Bai, Tianshi Zheng, and Yangqiu Song. Sequential query encoding for complex query answering on knowledge graphs. *arXiv preprint arXiv:2302.13114*, 2023.

[4] Jiaxin Bai, Wei Fan, Qi Hu, Qing Zong, Chunyang Li, Hong Ting Tsang, Hongyu Luo, Yauwai Yim, Haoyu Huang, Xiao Zhou, et al. Autoschemakg: Autonomous knowledge graph construction through dynamic schema induction from web-scale corpora. *arXiv preprint arXiv:2505.23628*, 2025.

[5] Jiaxin Bai, Zihao Wang, Yukun Zhou, Hang Yin, Weizhi Fei, Qi Hu, Zheye Deng, Jiayang Cheng, Tianshi Zheng, Hong Ting Tsang, et al. Top ten challenges towards agentic neural graph databases. *arXiv preprint arXiv:2501.14224*, 2025.

[6] Tianyi Bao, Xinyu Ye, Hang Ruan, Chang Liu, Wenjie Wu, and Junchi Yan. Beyond circuit connections: A non-message passing graph transformer approach for quantum error mitigation. In *The Thirteenth International Conference on Learning Representations*, 2025.

[7] Maciej Besta, Patrick Iff, Florian Scheidl, Kazuki Osawa, Nikoli Dryden, Michal Podstawski, Tiancheng Chen, and Torsten Hoefler. Neural graph databases. In *Learning on Graphs Conference*, pages 31–1. PMLR, 2022.

[8] 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.

[9] Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning. In *International conference on machine learning*, pages 3469–3489. PMLR, 2022.

[10] Dexiong Chen, Markus Krimmel, and Karsten Borgwardt. Flatten graphs as sequences: Transformers are scalable graph generators. *arXiv preprint arXiv:2502.02216*, 2025.

[11] Haibo Chen, Xin Wang, Zeyang Zhang, Haoyang Li, Ling Feng, and Wenwu Zhu. Autogfm: Automated graph foundation model with adaptive architecture customization. In *Forty-second International Conference on Machine Learning*, 2025.

[12] Xuelu Chen, Ziniu Hu, and Yizhou Sun. Fuzzy logic based logical query answering on knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3939–3948, 2022.

[13] Dongwon Choi, Sunwoo Kim, Juyeon Kim, Kyungho Kim, Geon Lee, Shinhwan Kang, Myunghwan Kim, and Kijung Shin. Rdb2g-bench: A comprehensive benchmark for automatic graph modeling of relational databases. *arXiv preprint arXiv:2506.01360*, 2025.

[14] Nurendra Choudhary, Nikhil Rao, Sumeet Katariya, Karthik Subbian, and Chandan K Reddy. Self-supervised hyperboloid representations from logical queries over knowledge graphs. In *Proceedings of the web conference 2021*, pages 1373–1384, 2021.

[15] Daniel Daza and Michael Cochez. Message passing query embedding. *arXiv preprint arXiv:2002.02406*, 2020.

[16] Tao Feng, Yexin Wu, Guanyu Lin, and Jiaxuan You. Graph world model. *arXiv preprint arXiv:2507.10539*, 2025.

[17] Shahaf E Finder, Ron Shapira Weber, Moshe Eliasof, Oren Freifeld, and Eran Treister. Improving the effective receptive field of message-passing neural networks. *arXiv preprint arXiv:2505.23185*, 2025.

[18] Michael Galkin, Jincheng Zhou, Bruno Ribeiro, Jian Tang, and Zhaocheng Zhu. A foundation model for zero-shot logical query reasoning. *Advances in Neural Information Processing Systems*, 37:54137–54160, 2024.

[19] Mikhail Galkin, Priyansh Trivedi, Gaurav Maheshwari, Ricardo Usbeck, and Jens Lehmann. Message passing for hyper-relational knowledge graphs. *arXiv preprint arXiv:2009.10847*, 2020.

[20] Mikhail Galkin, Xinyu Yuan, Hesham Mostafa, Jian Tang, and Zhaocheng Zhu. Towards foundation models for knowledge graph reasoning. *arXiv preprint arXiv:2310.04562*, 2023.

[21] Yisen Gao, Jiaxin Bai, Haoyu Huang, Zhongwei Xie, Yufei Li, Hong Ting Tsang, Sirui Han, and Yangqiu Song. KGPFN: Unlocking the potential of knowledge graph foundation model via in-context learning. *arXiv preprint arXiv:2605.14907*, 2026.

[22] Will Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. Embedding logical queries on knowledge graphs. *Advances in neural information processing systems*, 31, 2018.

[23] Longzhu He, Chaozhuo Li, Peng Tang, and Sen Su. Going deeper into locally differentially private graph neural networks. In *Forty-second International Conference on Machine Learning*, 2025.

[24] Qi Hu, Weifeng Jiang, Haoran Li, Zihao Wang, Jiaxin Bai, Qianren Mao, Yangqiu Song, Lixin Fan, and Jianxin Li. Federated neural graph databases. *arXiv preprint arXiv:2402.14609*, 2024.

[25] Qi Hu, Haoran Li, Jiaxin Bai, Zihao Wang, and Yangqiu Song. Privacy-preserved neural graph databases. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1108–1118, 2024.

[26] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[27] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*, pages 2704–2710, 2020.

[28] Mohamed Amine Ketata, David Lüdke, Leo Schwinn, and Stephan Günnemann. Joint relational database generation via graph-conditional diffusion models. *arXiv preprint arXiv:2505.16527*, 2025.

[29] Jinwoo Kim, Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners. *Advances in Neural Information Processing Systems*, 35:14582–14595, 2022.

[30] Jun Hyeong Kim, Seonghwan Kim, Seokhyun Moon, Hyeongwoo Kim, Jeheon Woo, and Woo Youn Kim. Discrete diffusion schrödinger bridge matching for graph transformation. *arXiv preprint arXiv:2410.01500*, 2024.

[31] Félix Lefebvre and Gaël Varoquaux. Scalable feature learning on huge knowledge graphs for downstream machine learning. *arXiv preprint arXiv:2507.00965*, 2025.

[32] 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.

[33] Yi Li, Zhichun Guo, Guanpeng Li, and Bingzhe Li. You only spectralize once: Taking a spectral detour to accelerate graph neural network. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[34] Yufei Li, Yisen Gao, Jiaxuan Xiong, Jiaxin Bai, Shijie Zhong, Haoyu Huang, Zhongwei Xie, Hong Ting Tsang, and Yangqiu Song. NGDBench: Towards neural graph data management. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, pages 32890–32923, 2026.

[35] Hongyu Lin, Haoran Luo, Hanghang Cao, Yang Liu, Shihao Gao, Kaichun Yao, Libo Zhang, Mingjie Xing, and Yanjun Wu. KGCompiler: Deep learning compilation optimization for knowledge graph complex logical query answering. *arXiv preprint arXiv:2503.02172*, 2025.

[36] Erxue Min, Runfa Chen, Yatao Bian, Tingyang Xu, Kangfei Zhao, Wenbing Huang, Peilin Zhao, Junzhou Huang, Sophia Ananiadou, and Yu Rong. Transformer for graphs: An overview from architecture perspective. *arXiv preprint arXiv:2202.08455*, 2022.

[37] Jason Mohoney, Roger Waleffe, Henry Xu, Theodoros Rekatsinas, and Shivaram Venkataraman. Marius: Learning massive graph embeddings on a single machine. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021.

[38] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*, 2019.

[39] Zihan Pengmei, Zhengyuan Shen, Zichen Wang, Marcus Collins, and Huzefa Rangwala. Pushing the limits of all-atom geometric graph neural networks: Pre-training, scaling and zero-shot transfer. *arXiv preprint arXiv:2410.21683*, 2024.

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

[41] Hongyu Ren and Jure Leskovec. Beta embeddings for multi-hop logical reasoning in knowledge graphs. *Advances in Neural Information Processing Systems*, 33:19716–19726, 2020.

[42] Hongyu Ren, Weihua Hu, and Jure Leskovec. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. *arXiv preprint arXiv:2002.05969*, 2020.

[43] Hongyu Ren, Hanjun Dai, Bo Dai, Xinyun Chen, Denny Zhou, Jure Leskovec, and Dale Schuurmans. 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*, pages 1472–1482, 2022.

[44] Hongyu Ren, Mikhail Galkin, Michael Cochez, Zhaocheng Zhu, and Jure Leskovec. Neural graph reasoning: Complex logical query answering meets graph databases. *arXiv preprint arXiv:2303.14617*, 2023.

[45] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-supervised graph transformer on large-scale molecular data. *Advances in neural information processing systems*, 33:12559–12571, 2020.

[46] Zhihao Shi, Jie Wang, Zhiwei Zhuang, Xize Liang, Bin Li, and Feng Wu. Accurate and scalable graph neural networks via message invariance. *arXiv preprint arXiv:2502.19693*, 2025.

[47] 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.

[48] Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions. *arXiv preprint arXiv:1803.06643*, 2018.

[49] Bohan Tang, Zexi Liu, Keyue Jiang, Siheng Chen, and Xiaowen Dong. Training-free message passing for learning on hypergraphs. *arXiv preprint arXiv:2402.05569*, 2024.

[50] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International conference on machine learning*, pages 2071–2080. PMLR, 2016.

[51] Kexin Wang, Nils Reimers, and Iryna Gurevych. Tsdae: Using transformer-based sequential denoising auto-encoder for unsupervised sentence embedding learning. *arXiv preprint arXiv:2104.06979*, 2021.

[52] Liang Wang, Wei Zhao, Zhuoyu Wei, and Jingming Liu. Simkgc: Simple contrastive knowledge graph completion with pre-trained language models. *arXiv preprint arXiv:2203.02167*, 2022.

[53] Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. Kepler: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*, 9:176–194, 2021.

[54] Yanbo Wang, Xiyuan Wang, Quan Gan, Minjie Wang, Qibin Yang, David Wipf, and Muhan Zhang. Griffin: Towards a graph-centric relational database foundation model. *arXiv preprint arXiv:2505.05568*, 2025.

[55] Qiyao Wei, Edward Morrell, Lea Goetz, and Mihaela van der Schaar. Semantic-kg: Using knowledge graphs to construct benchmarks for measuring semantic similarity. *arXiv preprint arXiv:2511.19925*, 2025.

[56] Yuhan Wu, Yuanyuan Xu, Wenjie Zhang, Xiwei Xu, and Ying Zhang. Query2gmm: Learning representation with gaussian mixture model for reasoning over knowledge graphs. In *Proceedings of the ACM web conference 2024*, pages 2149–2158, 2024.

[57] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[58] Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. C-pack: Packaged resources to advance general chinese embedding, 2023.

[59] Zhongwei Xie, Yiming Du, Jiaxin Bai, Baixuan Xu, Chunyang Li, Xiyu Ren, Wei Fan, Tianshi Zheng, Weizhi Du, Jingqiao Yu, et al. Coworkenv: Synthesizing training environments for long-horizon cowork agents. *arXiv preprint arXiv:2610.24960*, 2026.

[60] Zhongwei Xie, Ruihao Liao, Zimo Wang, Chong Chen, Xian-Sheng Hua, and Xiao Luo. GALA: Geometric data selection with strategic prospecting for large language model self-training. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 10281–10293, 2026.

[61] Zhongwei Xie, Xiyu Ren, Tianshi Zheng, Jiaxin Bai, Wei Fan, Baixuan Xu, Haoran Li, Huihao Jing, and Yangqiu Song. A survey on AI Agent Harness. *ResearchGate Preprint*, 5 2026. DOI:10.13140/RG.2.2.31393.57447.

[62] Bo Xiong, Mojtaba Nayyeri, Ming Jin, Yunjie He, Michael Cochez, Shirui Pan, and Steffen Staab. Geometric relational embeddings: A survey. *arXiv preprint arXiv:2304.11949*, 2023.

[63] Yu Yan, Fei Hu, Jiusheng Chen, Nikhil Bhendawade, Ting Ye, Yeyun Gong, Nan Duan, Desheng Cui, Bingyu Chi, and Ruofei Zhang. Fastseq: Make sequence generation faster. *arXiv preprint arXiv:2106.04718*, 2021.

[64] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

[65] Dong Yang, Peijun Qing, Yang Li, Haonan Lu, and Xiaodong Lin. Gammae: Gamma embeddings for logical queries on knowledge graphs. *arXiv preprint arXiv:2210.15578*, 2022.

[66] Liang Yao, Chengsheng Mao, and Yuan Luo. Kg-bert: Bert for knowledge graph completion. *arXiv preprint arXiv:1909.03193*, 2019.

[67] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

[68] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

[69] Zhanqiu Zhang, Jie Wang, Jiajun Chen, Shuiwang Ji, and Feng Wu. Cone: Cone embeddings for multi-hop reasoning over knowledge graphs. *Advances in Neural Information Processing Systems*, 34:19172–19183, 2021.

[70] Xin Zheng, Wei Huang, Chuan Zhou, Ming Li, and Shirui Pan. Test-time graph neural dataset search with generative projection. In *Forty-second International Conference on Machine Learning*, 2025.

[71] Qihui Zhou, Peiqi Yin, Xiao Yan, Changji Li, Guanxian Jiang, and James Cheng. Atom: An efficient query serving system for embedding-based knowledge graph reasoning with operator-level batching. *Proceedings of the ACM on Management of Data*, 2(4):1–29, 2024.

[72] Zhaocheng Zhu, Mikhail Galkin, Zuobai Zhang, and Jian Tang. Neural-symbolic models for logical queries on knowledge graphs. In *International conference on machine learning*, pages 27454–27478. PMLR, 2022.

Appendix

Contents

A Detailed Comparison with Prior Systems . . . 16

B Limitations and Future Work . . . 17

C Broader Impact . . . 17

D Algorithm . . . 17

E KG Statistics . . . 17

F Implementation Details . . . 18

G Joint Structural-Semantic Representation . . . 19

H Stochastic Online Query Sampling . . . 19

H.1 Formal Definition and Motivation . . . 19

H.2 Sampling Strategy and Efficiency . . . 20

I Throughput Analysis . . . 20

J Batching Granularity Comparison . . . 21

K Dispatch Policy Comparison . . . 21

L Mechanism-Level Systems Breakdown . . . 21

M Additional Results on Negation Queries . . . 21

N Additional Semantic Augmentation Results . . . 21

A Detailed Comparison with Prior Systems

This section clarifies the scope of NGDB-Zoo relative to prior systems and reasoning paradigms. The central distinction is that NGDB-Zoo targets the *stateful training loop* of dense-retrieval query embedding models. In this setting, batching is not only an inference scheduling problem: the system must preserve autograd dependencies, retain or release intermediate tensors at the right time, synchronize optimizer updates, and continuously generate positive and negative training instances. These requirements create bottlenecks that are absent from stateless serving systems and are different from execution-based neural-symbolic methods.

Comparison with ATOM. ATOM [71] introduced operator-level batching for knowledge graph reasoning serving. Its execution is stateless: a query is parsed, decomposed into operators, executed for inference, and then discarded. NGDB-Zoo is designed for training, where the same decomposition must support exact gradient propagation through many heterogeneous queries. This difference introduces three training-specific requirements. First, intermediate tensors cannot be freed immediately after their forward use because they may be needed by backward computation. NGDB-Zoo therefore uses eager reference counting to release scheduler-owned references after all dependent forward operators have been dispatched; tensors needed for backward remain available to autograd. Second, the input workload is actively generated rather than passively served: online query synthesis, answer lookup, and negative sampling must be overlapped with GPU execution. Third, final query embeddings must be synchronized with candidate entity embeddings to compute a global contrastive objective and update model parameters. These requirements make NGDB-Zoo complementary to, rather than a direct serving-time variant of, ATOM.

Comparison with MPQE and GNN-QE. MPQE [15] and GNN-QE [72] are important prior systems that also relax rigid query execution in different ways. MPQE batches queries through graph construction, but its batching procedure is tied to formula templates: a single query structure is selected and replicated within the batch. This is effective graph batching, but it does not create a global operator pool that fuses the same operator type across structurally heterogeneous query DAGs. GNN-QE, by contrast, uses a reverse Polish notation execution plan and can batch compatible operations from different query structures. Our contribution is to build this execution style into a dense-retrieval QE training runtime with autograd-aware memory management, online sampling, and large-KG pipelining.

The key difference is the computational paradigm. GNN-QE performs execution-based message passing over the graph, where projection can require full-graph neural propagation and large intermediate activations. This design is expressive, but it creates irregular memory access and activation storage costs that grow with the number of entities and GNN layers. NGDB-Zoo instead focuses on Skolemization-style query embedding models, where a query is recursively mapped to a dense vector or distribution and answers are retrieved by dense scoring. This dense-operator regime allows cross-query operator fusion, vectorized intersection and union, and predictable memory reclamation on large KGs.

GNN-QE memory profile. On one A6000 at batch size 48, eager reclamation reduced peak GNN-QE memory from 27.6 to 26.2 GB on FB15k and from 34.2 to 29.9 GB on FB15k-237. On NELL995, both configurations ran out of memory at batch size 48 and ran successfully at batch size 12.

Comparison with execution-based and decomposition methods. CQD [1], ULTRAQ [18], and related methods are highly relevant to complex query answering. They differ from NGDB-Zoo in their fundamental reasoning paradigms: CQD answers complex queries by explicit decomposition and continuous optimization over candidate entities, while ULTRAQ extends foundation GNNs to complex queries via execution-based message passing and fuzzy logic. These methods are not currently supported by NGDB-Zoo. Supporting them would require a different scheduler and memory manager for sparse graph traversal, irregular multi-source propagation, and global search states. We therefore position NGDB-Zoo as a scalable training framework for query embedding models rather than a universal runtime for all neural-symbolic query answering methods.

Comparison with KGR, SMORE, and SQE. KGR [40], SMORE [43], and SQE [3] provide strong baselines for training query embedding or sequential query encoding models, but they do not

implement the same combination of cross-structure operator pooling, cardinality-aware vectorized set operators, and asynchronous online sampling. Their training loops largely preserve query-level or structure-level execution boundaries, so a mixed workload of many query templates leads to fragmented kernels and lower effective batch sizes per operator. Building on established asynchronous sampling and pipelining in SMORE, NGDB-Zoo combines three complementary mechanisms: Max-Fillness scheduling aggregates ready operators of the same type across heterogeneous queries; cardinality-based vectorization converts variable-input intersection and union operators into dense tensor execution; and the CPU-GPU pipeline overlaps online sampling and embedding transfer with GPU computation. In the purely structural setting, NGDB-Zoo uses the same model backbones and a single trainable structural embedding table as the baselines; semantic entity representations are introduced only in the separate semantic augmentation experiments.

B Limitations and Future Work

Despite its efficiency, the current architecture relies on a frozen text encoder. Future research could explore lightweight end-to-end co-optimization of the semantic and structural subspaces. System Trade-offs. To scale to massive graphs with limited VRAM, our current implementation on 4M-node graphs relies on asynchronous CPU-offloading. While effective, this introduces a dependency on PCIe bandwidth, which may become a bottleneck for even larger graphs or larger batch sizes compared to a fully GPU-resident approach. On an A6000, random host-row gathering limited staging to 2.59–4.45 GB/s, with 1.7–$3.2\times$ estimated headroom at the measured query rate across configurations up to $16\mathrm{M} \times 1024$ or $4\mathrm{M} \times 2048$ FP16 values. Furthermore, the reliance on a frozen text encoder decouples semantic understanding from structural learning, potentially limiting the model's ability to adapt linguistic priors to specific graph topologies. Additionally, extending the operator-level paradigm to streaming knowledge bases, where the topology evolves in real-time, remains a promising direction for achieving truly autonomous graph reasoning.

C Broader Impact

NGDB-Zoo is a systems framework for scalable neural graph reasoning and is not tied to a specific deployment domain. Its positive impact is to reduce the computational cost of training multi-hop reasoning models, making large-scale KG reasoning more accessible to researchers and practitioners with limited compute resources. By providing an efficient path for structural-semantic integration, it can also support further methods that combine symbolic graph structure with external semantic priors. Potential biases arise when the framework is applied in domains such as medicine, finance, and law. We therefore recommend domain-specific evaluation and human oversight before deployment in high-stakes settings.

D Algorithm

In Algorithm 1, we presented our NGDB-Zoo in the form of pseudo-code to illustrate how it works.

Logical view vs. runtime execution. Algorithm 1 shows the logical scheduling order of the dispatcher; at runtime, batches selected in successive iterations for disjoint operator types may be launched on independent CUDA streams and overlap on the device, as described in Appendix F.

E KG statistics

In Table 5, we show the statistics of all six KGs we use with number of entities, relations, training, validation and test edges. For ATLAS-Wiki-Triple-4M, we construct a 6.4% subgraph from ATLAS-Wiki-Triple using degree-weighted edge sampling. Edge sampling probability is proportional to the sum of endpoint node degrees, prioritizing connections between high-degree (hub) nodes to preserve the graph's core topological structure.

Algorithm 1 Operator-Level Dynamic Scheduling Training Loop

Require: Batch of Queries $\mathcal{Q}$, Model Parameters Θ, Per-type Max Batch Sizes $\{B_{\max,\tau}\}_\tau$
$\mathcal{G} \leftarrow \bigcup_{q \in \mathcal{Q}} \text{BUILDDAG}(q)$ ▷ Construct fused computation graph
$\mathcal{G} \leftarrow \mathcal{G} \cup \text{ADDGRADIENTNODES}(\mathcal{G})$ ▷ Include backward operators
Initialize operator pools $\mathcal{P}_\tau$ for each operator type τ
Initialize ready set $\mathcal{R} \leftarrow \{v \in \mathcal{G} \mid \text{indegree}(v) = 0\}$
while $\mathcal{R} \neq \emptyset$ or $\exists \tau : \mathcal{P}_\tau \neq \emptyset$ **do**
  Distribute $v \in \mathcal{R}$ into pools $\mathcal{P}_{\text{TYPE}(v)}$
  $\mathcal{R} \leftarrow \emptyset$ ▷ Clear ready set after enqueueing ready operators
  Select $\tau^* \leftarrow \arg\max_{\tau : \mathcal{P}_\tau \neq \emptyset} \frac{|\mathcal{P}_\tau|}{B_{\max,\tau}}$ ▷ Apply **Max-Fillness Policy**
  $\mathcal{B} \leftarrow \text{POPBATCH}(\mathcal{P}_{\tau^*}, B_{\max,\tau^*})$
  $\mathbf{O} \leftarrow \text{EXECUTEKERNEL}(\mathcal{B}, \Theta)$ ▷ Fused GPU Kernel
  for $o_i \in \mathcal{B}$ **do**
    o_i.output $\leftarrow \mathbf{O}_i$
    $\text{DECREMENTREFCOUNTS}(\text{INPUTS}(o_i))$ ▷ **Eager Reclamation**
    for $v_{succ} \in \text{SUCCESSORS}(o_i)$ **do**
      if $\text{DEPENDENCIESMET}(v_{succ})$ **then**
        $\mathcal{R} \leftarrow \mathcal{R} \cup \{v_{succ}\}$ ▷ Update ready set
      end if
    end for
  end for
end while
$\text{OPTIMIZERSTEP}(\Theta)$ ▷ Apply gradients computed in the loop

Table 5: KG statistics with number of entities, relations, training, validation and test edges.

Dataset	Entities	Relations	Train Edges	Valid Edges	Test Edges	Total Edges
FB15k	14,951	1,345	483,142	50,000	59,071	592,213
FB15k-237	14,505	237	272,115	17,526	20,438	310,079
NELL995	63,361	200	114,213	14,324	14,267	142,804
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
ATLAS-Wiki-Triple-4M	4,035,238	512,064	23,040,868	2,880,108	2,880,110	28,801,086

F Implementation Details

Asynchronous Parallelism and Pipelining. We implement a decoupled execution pipeline characterized by two key parallelization strategies:

- **Multi-stream Operator Execution:** We leverage independent CUDA streams to parallelize disjoint operator types within the QueryDAG. This allows for the simultaneous execution of compute-bound and memory-bound kernels, effectively masking hardware latency.
- **Heterogeneous Pipelining:** To mitigate the bottleneck of CPU-side data preparation, we introduce a consumer-producer pipeline. While the GPU executes current operator batches, the CPU concurrently performs asynchronous sampling for subsequent queries, ensuring a continuous data flow to the accelerators.

Hyperparameters and Configuration. Table 6 summarizes the unified hyperparameter settings applied across all baseline query encoding models and the configuration for the Qwen3 semantic integration. To ensure a fair comparison with prior work, we maintain a consistent training environment: all baselines are initialized with a latent dimension of 400 and trained using the same negative sampling strategy and optimizer configurations. For the shared configurations in Table 4, BetaE uses 1M optimizer steps and Q2B/GQE use 800K, with matched budgets across frameworks and seeds $\{0, 42, 123\}$. Multi-GPU runs use data parallelism with 512 queries per worker; dense model gradients are synchronized once per optimizer step, while CPU-resident sparse embeddings use asynchronous updates. With 512 queries per worker, aggregate throughput increased 3.13–$3.43\times$ from two to eight GPUs across the configurations in Figure 7.

Table 6: Hyperparameter settings and model configurations.

Parameter	Value
Batch Size	512 queries
Optimizer	Adam (lr 10^{-4})
Margin (γ)	12.0

G Joint Structural-Semantic Representation

We formalize the integration of semantic information not as a mere feature augmentation, but as a mapping into a **composite representation manifold** $\mathcal{H}_{\text{joint}}$. Specifically, we define the problem as finding a joint embedding for each entity $e_i \in \mathcal{E}$ that simultaneously inhabits two distinct latent subspaces.

Subspace Definitions. The representation space is decomposed into:

1. **Structural Subspace ($\mathcal{H}_{\text{str}} \subseteq \mathbb{R}^{d_s}$):** An endogenous, learnable space optimized to satisfy the relational constraints and multi-hop logical dependencies inherent in $\mathcal{G}$.
2. **Semantic Subspace ($\mathcal{H}_{\text{sem}} \subseteq \mathbb{R}^{d_m}$):** An exogenous, fixed manifold derived from a pre-trained linguistic encoder $\Phi : \mathcal{D} \to \mathcal{H}_{\text{sem}}$. For each entity e_i, its semantic prior $\mathbf{s}_i = \Phi(T_i)$ is grounded in its textual description $T_i \in \mathcal{D}$.

Problem Definition: Semantic Grounding. We posit that the complete joint representation of an entity e_i concatenates its structural and semantic signals:

$$\mathbf{x}_i = [\mathbf{h}_i \oplus \mathbf{s}_i], \quad \text{s.t. } \mathbf{h}_i \in \mathcal{H}_{\text{str}}, \mathbf{s}_i \in \mathcal{H}_{\text{sem}}. \tag{9}$$

where $\mathbf{h}_i$ is a set of adaptive parameters and $\mathbf{s}_i$ acts as an invariant semantic anchor. The model-specific initial entity representation is then obtained by $\mathbf{z}_i^0 = \Psi_\theta(\mathbf{x}_i)$.

Parameters for Geometric and Probabilistic Reasoning. For reasoning frameworks that represent queries as geometric regions (e.g., boxes) or probability distributions (e.g., beta distributions), the corresponding representation parameters θ_i are derived from the joint space:

$$\theta_i = \Gamma(\mathbf{z}_i^0) = \Gamma(\Psi_\theta(\mathbf{x}_i)) \tag{10}$$

This formulation incorporates both structural embeddings and fixed linguistic features into the model-specific geometric or distributional parameters.

H Stochastic Online Query Sampling

To handle the combinatorial explosion of query structures in Large-scale Knowledge Graphs (KGs), we implement an **Online Stochastic Sampler** $\mathcal{S}$. Unlike traditional offline methods that rely on pre-computed static query-answer datasets, our approach synthesizes training samples *on-the-fly*. This shifts the cost from persistent query storage to per-batch generation and allows the training pipeline to produce query instances, positive answers, and corrupted negatives according to the current workload. In language model self-training, GALA promotes sample diversity by selecting examples from clusters of sentence embeddings [60].

H.1 Formal Definition and Motivation

Let $\mathcal{G} = (\mathcal{V}, \mathcal{R})$ denote the knowledge graph. A query q can be framed as a subgraph retrieval task. We define the sampling process at training step t as a stochastic mapping:

$$\mathcal{B}_t = \{q_i\}_{i=1}^{|\mathcal{B}|} \sim \mathcal{S}(\mathcal{G}, \mathcal{T}; \pi) \tag{11}$$

where $\mathcal{T}$ represents the set of logical templates and π is the sampling distribution (e.g., uniform or frequency-weighted). The primary motivation for online sampling is to overcome the *data rigidity* and storage cost of static benchmarks, allowing the model to encounter diverse query-answer pairs $(q, \mathcal{A}_q)$ across the training manifold without materializing them in advance.

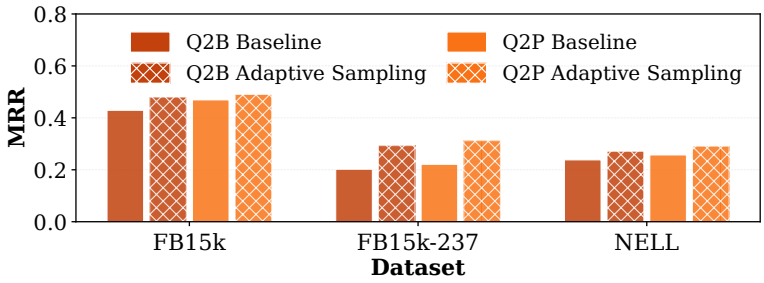


Figure 9: Controlled diagnostic comparison with and without adaptive sampling under a synthetic non-stationary query distribution.

H.2 Sampling Strategy and Efficiency

The sampler employs a hybrid strategy to ensure both efficiency and validity of the generated queries:

- **Dynamic Traversal:** For each batch, the sampler performs a restricted random walk or neighborhood expansion on $\mathcal{G}$ to instantiate logical variables.
- **Constraint Satisfaction:** To maintain query quality, we incorporate a *Rejection Sampling* mechanism. A candidate query q is accepted if it satisfies the structural constraint τ, i.e., $P_{accept}(q) \propto \mathbb{I}(q \in \mathcal{Q}_{valid})$, where $\mathcal{Q}_{valid}$ is the space of queries with non-empty answer sets.
- **Positive and Negative Generation:** Accepted queries are evaluated on $\mathcal{G}_{train}$ to obtain denotation-based positive answers, while negative answers are generated by corrupting target entities outside the positive denotation set.
- **Sampling Cost:** Per-batch runtime depends on query depth, graph degrees, answer-set computation, and rejected candidates. Traditional offline datasets incur storage proportional to the number of generated query-answer pairs, whereas our sampler retains the graph, template set, sampler state, and current batch buffers.

On FB15k-237/BetaE (one A6000), sampling ten templates with Zipf exponents $0, 0.5, 1, 1.5, 2$ yielded 3,610–6,046 sampled queries/s. Switching from projection/intersection-heavy to union/negation-heavy queries produced a mean peak throughput dip of 3.7% over two seeds, while a separate negation-only memory test (2in/3in/inp/pin/pni) peaked at 1.562 GB versus 1.571 GB under uniform sampling.

I Throughput Analysis

Across the shared configurations in Table 4, SMORE delivered about $4\times$ the throughput per GB of NGDB-Zoo (ratio of mean throughput to mean memory). The end-to-end speedup of NGDB-Zoo over query-level baselines arises from three complementary mechanisms:

(i) Max-Fillness Operator Scheduling. Rather than batching entire queries of the same structure, the scheduler aggregates ready operators of the same type across heterogeneous queries. This eliminates the structural fragmentation that limits effective batch sizes in query-level systems (e.g., KGR, SMORE, SQE), where a mixed workload of $|\mathcal{T}|$ query templates reduces the per-kernel batch size by approximately $|\mathcal{T}|^{-1}$.

(ii) Cardinality-Based Vectorization. Variable-arity operators such as Intersection and Union are grouped into equivalence classes by input cardinality (Section 4.3), converting them into dense matrix operations.

(iii) Asynchronous CPU-GPU Pipelining. The CPU-GPU pipeline (Section 4.3) overlaps online query sampling, answer lookup, negative sampling, and embedding transfer with GPU computation. This allows data preparation and GPU execution to overlap rather than executing them sequentially, which is particularly important for large KGs where embedding access involves CPU-to-GPU staging.

These three mechanisms contribute in complementary ways: operator scheduling increases per-kernel batch sizes, vectorization accelerates the kernels themselves, and pipelining hides the latency of data preparation. In the purely structural setting, NGDB-Zoo uses the same model backbones and a single trainable embedding table as the baselines; the comparison evaluates system-level optimization.

J Batching Granularity Comparison

We isolated batching granularity with BetaE over 100 steps using batches of 512 queries evenly drawn from five templates (1p, 2p, 3p, 2i, 3i), with multi-stream execution and cardinality-based vectorization disabled. Grouping operators across queries increased throughput from 229 to 501 queries/s on NELL995 ($2.19\times$) and from 425 to 493 on FB15k-237 ($1.16\times$) relative to batching by query structure. Including inverse relations gives 400 relation types on NELL995 and 474 on FB15k-237, twice the original counts in Table 5.

K Dispatch Policy Comparison

We compared Max-Fillness, FIFO, and Random with identical initialization and sampling on one A6000 (200 steps, 30 warmup steps, batch size 512, one CUDA stream, cardinality-based vectorization disabled). Throughputs in that policy order were 434/422/421 queries/s for BetaE on NELL995, 491/461/456 for BetaE on FB15k-237, and 349/346/334 for Q2B on NELL995. Max-Fillness improved throughput by 3.1–7.7% over Random and exceeded FIFO in all three settings.

L Mechanism-Level Systems Breakdown

In separate BetaE/NELL995 diagnostics on one A6000 (batch size 1,024), operator pooling reduced kernel launches per step by 32.5% and sped up the step by $2.08\times$. Pipelining reduced CPU sampling time on the critical path from 224.7 to 13.0 ms, and eager reference release reduced peak GPU memory from 10.46 to 9.56 GB.

M Additional Results on Negation Queries

This section provides supplementary results for query patterns involving logical negation. The main experiments evaluate the full set of EFO query templates, while Table 7 isolates the subset containing negation operators: `2in`, `3in`, `inp`, `pin`, and `pni`. These results verify that NGDB-Zoo can train BetaE on negation-heavy query workloads without restricting evaluation to positive-only projection, intersection, or union patterns.

Table 7: BetaE performance on EFO query patterns involving negation. We report MRR and Hit@10 (%) across five negation query types and their average.

Dataset	Metrics	2in	3in	inp	pin	pni	avg
FB15k	MRR	13.00	14.97	9.17	6.11	11.88	11.03
	Hit@10	26.36	30.56	18.53	11.97	23.74	22.23
FB15k-237	MRR	3.96	6.95	6.52	3.97	2.96	4.87
	Hit@10	7.75	14.19	13.45	7.45	5.08	9.58
NELL995	MRR	4.06	6.65	8.03	3.25	2.92	4.98
	Hit@10	7.84	14.88	16.05	5.61	5.31	9.94
ogbl-wikikg2	MRR	36.38	50.92	39.07	39.31	38.50	40.84
	Hit@10	56.24	68.48	57.93	58.63	57.82	59.82

N Additional Semantic Augmentation Results

Table 8 reports two related but distinct comparisons for semantic integration with BGE-Base-En-v1.5. The MRR columns compare structure-only versus semantic-augmented models, while the throughput and memory columns compare in-loop frozen PTE inference versus decoupled cached semantic

access. 1. Throughput Enhancement: By decoupling the computationally intensive Pre-trained Text Encoders from the online training loop, we achieve a substantial speedup. With matched cached Qwen3 features on FB15k-237/BetaE, NGDB-Zoo processed 1,581 queries/s versus 1,080 for SMORE ($1.46\times$). 2. Memory Optimization: Our method reduces peak GPU memory usage despite incorporating high-dimensional semantic features. This reduction is attributed to unloading the large transformer after pre-computation and retaining only the embedding buffers needed during training. 3. Predictive Gains: Semantic augmentation improves average MRR by 3.18 percentage points across these configurations; the gain is not uniform, with a small decrease for GQE on FB15k. Randomly reassigning BGE embeddings across entities reduced CQA MRR below the structure-only baseline, and frozen BGE alone yielded near-zero filtered MRR on zero-shot link prediction without KG training.

Table 8: Semantic integration ablation with BGE-Base-En-v1.5 across three datasets and different models. The MRR columns compare structure-only and semantic-augmented variants, while the throughput and memory columns compare in-loop frozen PTE inference against decoupled cached semantic access. Higher is better for MRR and throughput; lower is better for memory.

Dataset	Model	MRR (%)		Training queries (Queries/Sec)		GPU Memory (GB)	
		Structure-only	Semantic-augmented	Online PTE	Decoupled + Opt.	Online PTE	Decoupled + Opt.
FB15k	BetaE	43.04	45.37	187	1402	14.40	13.21
	Q2B	42.85	47.54	340	1886	6.55	5.23
	GQE	30.39	30.17	550	2302	12.74	11.52
FB15k-237	BetaE	21.11	24.81	200	1503	11.28	10.01
	Q2B	20.19	25.72	312	1750	7.46	5.21
	GQE	16.98	22.13	534	2401	11.00	9.84
NELL995	BetaE	24.90	26.41	140	1323	8.16	7.14
	Q2B	23.79	26.20	238	1402	6.70	5.54
	GQE	18.89	22.37	515	2201	8.75	7.48
Average		**26.90**	**30.08**	**335**	**1797**	**9.67**	**8.35**

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer:[Yes]

   Justification: The abstract and Section 1 state three contributions that are validated in Section 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix B discusses limitations.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: This work focuses on system-level design and scalability optimization for training neural graph databases. It does not introduce new formal proofs, or convergence analyses. Therefore, this section is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 5 describes experimental setup; Appendix F provides full hyperparameters, optimizer settings, and hardware configuration; Appendix D gives the complete algorithm. An anonymous code repository is provided.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: An anonymous code repository is provided. Data details are discussed in appendix.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5 and Appendix F specify data splits, hyperparameters, optimizer (Adam), learning rates, batch sizes, and embedding dimensions for all experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: For accuracy-related metrics (MRR, Hits@K), we report the mean value over three runs with different random seeds. Throughput measurements are deterministic given fixed hardware settings.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 5 report detailed compute resources information.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses only publicly available datasets and open-source codebases, does not involve human subjects, and conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Appendix C discusses both positive impacts and negative risks.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The research uses only publicly available datasets and open-source codebases, does not involve human subjects, and conforms to the NeurIPS Code of Ethics.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No]

    Justification: All datasets (FB15k-237, NELL, ogbl-wikikg2, FB400k, Freebase, ATLAS-Wiki-4M) and baseline codebases are publicly available and properly cited. Original papers are referenced for each asset in Section 5 and Appendix E.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The NGDB-Zoo framework is released as an anonymous open-source repository with documentation and usage instructions.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This work does not involve human subjects or IRB-regulated research.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [No]

    Justification: LLMs were used only for writing polish and editing; they are not a component of the core methodology.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.