SINE: Scalable MPE Inference for Probabilistic Graphical Models using Advanced Neural Embeddings

Shivvrat Arya
The University of Texas at Dallas

Tahrima Rahman
The University of Texas at Dallas

Vibhav Gogate
The University of Texas at Dallas

Abstract

Our paper builds on the recent trend of using neural networks trained with self-supervised or supervised learning to solve the Most Probable Explanation (MPE) task in discrete graphical models. At inference time, these networks take an evidence assignment as input and generate the most likely assignment for the remaining variables via a single forward pass. We address two key limitations of existing approaches: (1) the inability to fully exploit the graphical model's structure and parameters, and (2) the suboptimal discretization of continuous neural network outputs. Our approach embeds model structure and parameters into a more expressive feature representation, significantly improving performance. Existing methods rely on standard thresholding, which often yields suboptimal results due to the non-convexity of the loss function. We introduce two methods to overcome discretization challenges: (1) an external oracle-based approach that infers uncertain variables using additional evidence from confidently predicted ones, and (2) a technique that identifies and selects the highest-scoring discrete solutions near the continuous output. Experimental results on various probabilistic models demonstrate the effectiveness and scalability of our approach, highlighting its practical impact.

1 INTRODUCTION

Probabilistic graphical models (PGMs) (Koller and Friedman, 2009), including Bayesian Networks (BN) and Markov Networks (MN), are widely used to model large, multi-dimensional probability distributions. However, as the complexity of these distributions grows, solving NP-hard inference tasks—such as determining the *Most Probable Explanation* (MPE) through exact inference (Otten, 2012; Otten and Dechter, 2012)—becomes computationally infeasible, which limits the scalability of these models in answering complex probabilistic queries. Although several approximate solvers have been developed for the MPE task, they often fail to achieve the accuracy required for real-world applications.

Recent advancements in neural network-based approximate solvers have addressed limitations in existing methods for inference in PGMs. Arya et al. (2024b) introduced a novel technique that combines inference-time optimization with a teacher-student framework to answer MPE queries across various Probabilistic Models (PMs). This technique builds on the earlier work of Arya et al. (2024a), which targeted both MPE and *Marginal Maximum A Posteriori* (MMAP) queries in probabilistic circuits (Choi et al., 2020). By employing a self-supervised loss function, these methods eliminate the need for exact solutions during training. Drawing on the literature of learning to optimize (Li and Malik, 2016; Fioretto et al., 2020; Donti et al., 2020; Zamzam and Baker, 2020; Park and Hentenryck, 2023), they leverage neural models to enable efficient probabilistic reasoning. Furthermore, these neural-based MPE solvers provide two key advantages: they deliver superior solution quality while reducing inference time compared to traditional solvers.

In this paper, we focus on advancing two key aspects of neural network-based models for answering MPE queries over PGMs: improving the quality of input embeddings for inference tasks and developing improved methods for discretization of the neural network's continuous outputs to obtain MPE solutions. Existing approaches typically rely on encoding schemes that focus solely on the probabilistic query, failing to fully exploit the rich information embedded in the PGM. Moreover, these methods often use thresholding for discretization, which only considers the nearest binary solution and may fail to identify the optimal MPE solution due to the non-linearity of the self-supervised loss function. As a result, these techniques are limited to solving simpler problems where the model learns to answer a single predefined query (Arya et al., 2024a), or they necessitate test-time optimization to achieve near-optimal solutions for

Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258. Copyright 2025 by the author(s).

arbitrary queries (Arya et al., 2024b).

In contrast, our approach handles arbitrary queries without requiring test-time optimization by embedding both the structure and parameters of PGMs into a hypergraph representation (Dechter, 2019). We employ neural message-passing algorithms within an attention-based Hypergraph Neural Network (HGNN) (Bai et al., 2019) to efficiently propagate information across the hypergraph. For initializing the HGNN embeddings, we utilize evidence-instantiated factors and initial evidence values, along with partition sets. This process produces more informative representations of the PGM, eliminating the need for additional methods to achieve near-optimal solutions.

To enhance the quality of MPE solutions, we further introduce novel discretization techniques to derive MPE solutions from continuous neural network outputs. The first method utilizes the neural network's outputs to identify variables with high uncertainty in their predictions. By augmenting the evidence with confidently predicted variables, the problem is simplified to a query that an oracle can efficiently solve. While neural networks excel at processing large queries quickly, they may exhibit inaccuracies due to the complexity of the loss function. In contrast, oracles—MPE solvers like those in Otten and Dechter (2012)—provide highly accurate solutions for smaller problems, though with slower performance on larger queries. By combining the strengths of both, this approach balances efficiency and accuracy in solving MPE queries.

The second method enhances solution quality by systematically exploring the neighborhood of binary solutions around the neural network's continuous output. Unlike thresholding, which evaluates only a single solution, this approach examines multiple nearby binary solutions. By leveraging the neural network's predictions as a starting point and examining various nearby configurations, this method effectively navigates the solution space to yield superior solutions compared to the thresholding approach. It also reduces the number of hyperparameters by removing the need to choose a specific threshold value, which may not generalize well across different datasets or problem settings.

We conducted an extensive empirical evaluation, comparing our proposed method against several traditional and neural approaches. Our experiments, performed on a diverse set of benchmark models, demonstrate that neural networks trained using our HGNN embeddings and utilizing advanced discretization techniques significantly outperform existing models in both efficiency and accuracy.

2 BACKGROUND AND MOTIVATION

For simplicity, we use binary variables which take values from the set $\{0, 1\}$. We denote random variables by uppercase letters (e.g., X), their assigned values by lowercase letters (e.g., x), and sets of random variables by bold uppercase letters (e.g., $\mathbf{X}$). For an assignment $\mathbf{x}$ to all variables in $\mathbf{X}$ and a variable $Y \in \mathbf{X}$, let $\mathbf{x}^Y$ represent the projection of the assignment $\mathbf{x}$ onto Y.

2.1 Hypergraphs

Hypergraphs extend the concept of traditional graphs by allowing edges, known as hyperedges, to connect more than two vertices. Formally, a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ consists of a set of vertices $\mathcal{V}$ and a set of hyperedges $\mathcal{E}$, where each hyperedge $e \in \mathcal{E}$ is a subset of $\mathcal{V}$. Unlike standard graphs, where edges are binary relations between two nodes, hypergraphs capture higher-order interactions among groups of vertices, making them suitable for modeling complex relational data in domains such as social networks, biology, and machine learning. Hypergraphs can represent multi-way dependencies, which are challenging to model using simple pairwise connections.

Hypergraph-based approaches have gained widespread recognition in representation learning for their ability to capture higher-order correlations. Methods such as HGNN (Feng et al., 2019), HyperGCN (Yadati et al., 2019), hypergraph convolution and attention operators (Bai et al., 2019), HyperSAGE (Arya et al., 2020), HNHN (Dong et al., 2020) and UniGNN (Huang and Yang, 2021) perform message passing over the hypergraph structure to capture complex relationships, improving the accuracy and robustness of learned representations.

2.2 Probabilistic Graphical Models

In this paper, we focus on Probabilistic Graphical Models (PGMs) (Koller and Friedman, 2009), particularly Markov and Bayesian networks. These models offer a framework for representing and reasoning about complex probabilistic relationships. PGMs enable the computation of the likelihood (or a value proportional to it) of an assignment of values to all variables in linear time in the size of the model. They efficiently represent the joint distribution through factorization, exploiting the conditional independence relationships encoded via a graph.

Bayesian networks (BNs) utilize directed acyclic graphs to represent conditional dependencies, allowing the joint distribution to factorize as:

$$\mathrm{p}_{\mathcal{M}}(\mathbf{X}) = \prod_{i=1}^{n} P(X_i | \mathrm{Pa}(X_i))$$

where $\mathbf{X} = \{X_1, \ldots, X_n\}$ is the set of random variables, and $\mathrm{Pa}(X_i)$ denotes the parents of X_i in the directed graph. $\mathcal{M}$ denotes the Bayesian network (probabilistic model).

Markov networks (MNs) represent conditional dependencies using undirected graphs. The joint distribution in these networks is expressed through potential functions, denoted as ϕ, which are defined over the maximal cliques of the undirected graph:

$$\mathrm{p}_{\mathcal{M}}(\mathbf{X}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{X}_c)$$

Here, $\mathcal{C}$ is the set of maximal cliques in the undirected graph, ϕ_c are non-negative potential functions defined over cliques, and Z is the partition function ensuring normalization. Again, $\mathcal{M}$ represents the probabilistic model. Henceforth, we refer to both conditional probabilities in BNs and potential functions in MNs as *factors*, denoted by $\mathcal{F}_c$, where c corresponds to the clique induced by the factor (in BNs, c refers to the clique formed by a node and its parents).

2.3 MPE Inference in PGMs

We focus on solving the Most Probable Explanation (MPE) task in PGMs, which involves determining the most likely assignment to unobserved (non-evidence) variables, given observations (evidence). Formally, let $\mathcal{M}$ represent a PGM defined over a set of variables $\mathbf{X}$, with the associated distribution $\mathrm{p}_{\mathcal{M}}(\mathbf{X})$. The variables $\mathbf{X}$ are partitioned into evidence $\mathbf{E} \subseteq \mathbf{X}$ and query $\mathbf{Q} \subseteq \mathbf{X}$ sets, such that $\mathbf{E} \cap \mathbf{Q} = \emptyset$ and $\mathbf{E} \cup \mathbf{Q} = \mathbf{X}$. Given an assignment $\mathbf{e}$ to the evidence variables $\mathbf{E}$, the MPE task is formulated as:

$$\texttt{MPE}(\mathbf{Q}, \mathbf{e}) = \operatorname*{argmax}_{\mathbf{q}} \mathrm{p}_{\mathcal{M}}(\mathbf{q}|\mathbf{e}) = \operatorname*{argmax}_{\mathbf{q}} \{\log \mathrm{p}_{\mathcal{M}}(\mathbf{q}, \mathbf{e})\} \tag{1}$$

It is known that the MPE task is NP-hard in general and even hard to approximate (Cooper, 1990; Park and Darwiche, 2004; de Campos, 2011).

2.4 Self-Supervised Loss for NN-based MPE Solver

Arya et al. (2024a) introduced an approach that uses neural networks to answer the MPE queries in Probabilistic Models (PMs). Given data, where each example is an assignment of values to the evidence variables, the main idea in their method is to use a *self-supervised loss* to train the neural network. At inference time, given a test example (an assignment to the evidence variables), the neural network is used to output the MPE assignment via a single forward-pass over the network.

The self-supervised loss utilizes continuous NN outputs, generated via sigmoid activations in the range $[0, 1]$. For an MPE query $\texttt{MPE}(\mathbf{Q}, \mathbf{e})$, let $\mathbf{q}^c \in [0, 1]^{|\mathbf{Q}|}$ represent the predicted continuous assignment. The goal is to maximize $\log \mathrm{p}_{\mathcal{M}}(\mathbf{q}, \mathbf{e})$, or equivalently minimize the loss $\ell(\mathbf{q}, \mathbf{e}) = -\log \mathrm{p}_{\mathcal{M}}(\mathbf{q}, \mathbf{e})$. The NN outputs continuous values, however $\mathrm{p}_{\mathcal{M}}(\mathbf{q}^c, \mathbf{e})$ is not well-defined.

Arya et al. (2024a) leveraged the multilinear nature of $\ell(\mathbf{q}, \mathbf{e})$, defining a continuous extension $\ell^c(\mathbf{q}^c, \mathbf{e})$ that matches the original loss on the discrete domain $\{0, 1\}^n$ but operates over $[0, 1]^n$. This loss is further improved by introducing an entropy-based penalty, parameterized by $\alpha > 0$, encouraging near-binary outputs and optimizing the solution for MPE queries.

Motivation: A key limitation of the approach presented by Arya et al. (2024a) is its reliance on a pre-defined partition of variables into query and evidence subsets. To address this, the authors later introduced the any-MPE task (see Arya et al. (2024b)). In this task, given a probabilistic graphical model (PGM), the neural network takes an assignment to an arbitrary subset of variables (evidence) as input and outputs the most likely assignments for the remaining (query) variables. The goal of this paper is to address and resolve two main limitations of their approach.

First, Arya et al.'s (2024b) approach relies exclusively on the query embedding (evidence values and the subsets of query and evidence variables), without leveraging the structure or parameters of the PGM. As a result, their method requires test-time optimization, with much of its performance gains attributed to this process. Second, their method uses a thresholding procedure to derive the MPE solutions from the continuous outputs of the neural network. This process does not account for the network's confidence or ensure that the thresholded solution is the best discrete solution near the continuous outputs. Given the non-linearity of the loss function, the best solution might exist near the continuous output.

In the following sections, we introduce a novel technique that generates more *informative* embeddings of the MPE-query by incorporating both the structure and parameters of the given PGM with the query. Additionally, we propose two methods that improve the conversion of continuous outputs into discrete near-optimal solutions.

3 AN ADVANCED ENCODING FOR ANY-MPE QUERIES IN PGMs

In this section, we introduce a hypergraph-based embedding to tackle the *any-MPE* task. Given a PGM and evidence—defined as an assignment of values to an arbitrary subset of variables—our goal is to construct an information-rich input encoding that captures the structure and parameters of the PGM and characterizes the subsets of variables representing both the query and evidence, along with the corresponding assignment to the evidence variables. This enriched encoding enhances generalization, reducing the reliance on expensive test-time optimization. Additionally, we propose an output encoding that aligns the neural network's output layer with the given PGM, enabling efficient recovery of the MPE solution from the network's predictions.

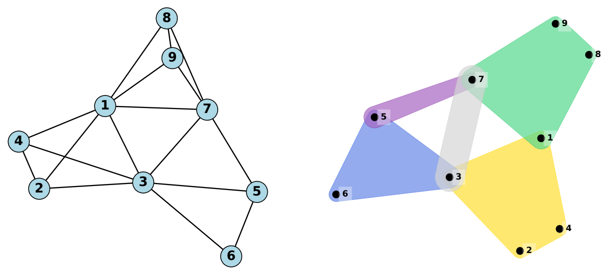
3.1 Integrating PGM Parameters Into the Encodings

Given an assignment to the evidence variables, a straightforward approach to utilize the parameters of the PGM into the input encoding is to instantiate the evidence in each factor/clique of the PGM and then concatenate the resulting instantiated factors. More formally, we apply the following transformation to each clique ϕ_k and each observed variable $E = e$:

$$\phi_k(X_k) = \begin{cases} 0, & \text{if} \quad \mathbf{x_k}^E \neq e \\ \phi_k(X_k), & \text{if} \quad \mathbf{x_k}^E = e \end{cases}$$

A key property of the resulting encoding, which is an *injective* mapping, is that it maintains a consistent length across all possible evidence assignments, making the length independent of the any-MPE query. This consistency is beneficial as it ensures that the neural network receives inputs of the same size. However, this approach has two significant limitations: it fails to incorporate structural information from the PGM, as the concatenation order is arbitrary, and it does not facilitate message passing between cliques. To address these issues, we aim to enhance the expressiveness of these embeddings to achieve a more informative representation.

3.2 Hypergraph Based Encoding



(a) Graph-based representation. Nodes represent variables, and edges capture pairwise dependencies.

(b) Hypergraph-based representation. Hyperedges (colored regions) represent cliques in the PGM.

Figure 1: Comparison of PGM structure representations: (a) graph, and (b) hypergraph.

PGMs can be effectively represented using both hypergraphs (Dechter, 2019) and graphs, with a one-to-one correspondence between the nodes in these structures. Each factor in the PGM corresponds to a clique in the graph, which maps to a hyperedge in the hypergraph. Figure 1 provides an example of these two representations for a PGM with 9 nodes, comprising 2 cliques of size 4, 1 clique of size 3, and 2 pairwise cliques. Each clique represents a factor involving the associated variables in the PGM.

Building on this representation, we explore how a hypergraph framework enables the generation of richer embeddings for any-MPE queries. Utilizing Hypergraph Neural Networks (HGNNs) (Feng et al., 2019), we aggregate embeddings within the hypergraph, enhancing the expressiveness of the resulting representation. Notably, hypergraph-based neural networks exhibit superior performance in capturing complex, higher-order relationships compared to traditional graph-based neural models (Feng et al., 2019; Cai et al., 2022; Chen and Schwaller, 2024).

To utilize HGNNs, three key components are required: (1) A node feature matrix that represents each node's feature vector. (2) A sparse incidence matrix that encodes the connectivity between nodes and hyperedges through the hyperedge indices, (3) Hyperedge feature matrices that allow the model to incorporate edge-specific information.

The incidence matrix is derived directly from the PGM's clique structure. For hyperedge features, we use the embeddings introduced in Section 3.1. Instead of concatenating embeddings from all cliques to form a single representation, we keep them separate, enabling each embedding to serve as the hyperedge attribute for its corresponding clique. Node attributes consist of two values: the first represents the node's value (assigned as the observed value for evidence variables or set to -1 for query variables), while the second indicates whether the node belongs to the query subset (true if it does, false otherwise).

To meet the uniform embedding size requirements of HGNNs, we pad the embeddings with zeros. However, when clique sizes vary significantly, this results in a substantial number of zeros in the embeddings. To mitigate this issue, we reparameterize the PGM so that all cliques have a uniform size. This is accomplished by first grouping variables into clusters of a fixed size, initializing each cluster potential to 1, and then multiplying each factor from the PGM with a cluster potential that includes all variables involved in that factor. This reparameterization preserves the joint distribution of the PGM (Koller and Friedman, 2009) while ensuring uniform clique sizes.

We then apply the hypergraph attention operator (Bai et al., 2019) to aggregate these embeddings. The attention layers process and combine information, producing aggregated node embeddings that serve as the final representation. This approach captures a rich representation incorporating (1) the PGM's structure, (2) its parameters, and (3) the details of the specific query. By accumulating information from neighboring nodes and cliques, it results in more expressive node embeddings. Additionally, this method ensures an *injective* mapping from any-MPE query to its corresponding embedding, preserving the uniqueness of each query's representation.

3.3 Neural Network Training

Given a PGM defined over n variables, we use the encodings from either Section 3.1 or Section 3.2 as inputs to a neural network. For the HGNN-based encoding, we concatenate all node embeddings to form a final representation. The network outputs n values using sigmoid activation, where each output node corresponds to a variable $X_i \in \mathbf{X}$. Outputs corresponding to evidence variables are disregarded, and the self-supervised loss function is applied only to the outputs for the query variables in $\mathbf{Q}$.

We generate the training data following the process described by Arya et al. (2024b). The neural network, along with the encoder parameters (applicable for HGNN based embedding), is trained using a tractable and differentiable self-supervised loss function (see Section 2.4), which allows efficient parameter learning from *unlabeled data*. Once trained, the model can efficiently answer arbitrary MPE queries over the PGM.

4 ADVANCED DISCRETIZATION TECHNIQUES FOR ENHANCED MPE SOLUTIONS

Both theoretical analyses and empirical evidence (Chizat and Bach, 2018; Allen-Zhu et al., 2019; Du et al., 2018; Zhang et al., 2016; Qin et al., 2024; Arora et al., 2019; Jacot et al., 2018; Lee et al., 2020a,b) suggest that overparameterized networks trained with non-convex loss functions (such as the one described in 2.4) tend to converge near the global minimum after repeated gradient updates. Since the true discrete and continuous losses coincide at $\{0, 1\}^n$, the continuous outputs of a trained network are likely to be close to a near-optimal discrete solution.

To obtain MPE solutions, the continuous outputs of the neural network must be discretized. A common approach used in recent methods (Arya et al., 2024a,b) applies thresholding to map outputs to binary values. However, even when the neural network converges near the global minimum, this approach can miss the optimal solution due to the non-linearity of the loss function. Furthermore, it discards valuable information from the continuous outputs by only considering the nearest discrete solution, often leading to suboptimal results, especially for non-linear loss functions.

To address these limitations, we present two novel discretization techniques that provide superior solutions. Unlike thresholding, these methods go beyond selecting a single nearest discrete point by leveraging the output probabilities to assess multiple candidate solutions or employing an oracle for uncertain variables. These approaches effectively leverage the rich information within the continuous outputs, leading to more accurate MPE solutions.

4.1 Oracle-Assisted Uncertainty-Aware Inference

The first proposed method, Oracle-Assisted Uncertainty-Aware Inference (OAUAI), utilizes an Oracle to generate solutions specifically for variables where the neural network exhibits low confidence, reducing the likelihood of incorrect MPE solutions. We first take the continuous output of the neural network and compute confidence scores for each query variable as $cs_i = |\mathbf{q}^c_i - 0.5|$. These scores allow us to divide the query variables into two subsets: (1) confident variables (high cs_i) and (2) uncertain variables (low cs_i).

While confident variables are directly thresholded to produce binary outputs, uncertain variables are handled differently to avoid relying on potentially inaccurate network predictions. For these variables, an MPE Oracle is queried to assign values that optimize the MPE objective. Similar to the classic cutset conditioning method (Pearl and Dechter, 1990), the Oracle processes a modified query where the confident variables serve as additional evidence, thereby reducing the number of variables involved in the query set. The final MPE solution is then constructed by merging Oracle's output for the uncertain variables with the thresholded outputs of the confident variables.

This method provides two key advantages: (1) It improves solution quality by deferring the handling of low-confidence variables to the Oracle for more accurate assignments. (2) Limiting Oracle's queries to only uncertain variables reduces the query set size, which increases Oracle's efficiency and decreases the overall computational complexity.

4.2 Fast Heuristic Search for Closest Binary Solutions

The second discretization method, referred to as Fast Heuristic Search for Closest Binary Solutions (k-Nearest), aims to identify multiple discrete candidates near the neural network's continuous output and select the optimal solution. Given that exhaustively scoring all $2^{|\mathbf{q}|}$ possible discrete solutions is computationally prohibitive, we employ a heuristic search algorithm to efficiently select the k nearest discrete solutions. This approach balances solution quality and computational feasibility, enabling the identification of high-quality discrete outputs without exhaustive enumeration.

First, we define a distance measure between the continuous output $\mathbf{q}^c$ and a binary vector $\mathbf{b}$ of size N as:

$$D(\mathbf{q}^c, \mathbf{b}) = \sum_{i=1}^{N} |q^c_i - b_i|.$$

The following algorithm leverages this distance measure to guide the heuristic search, enabling the efficient approximation of the k-nearest discrete solutions.

The algorithm employs pruning techniques to substantially narrow the search space at each iteration by retaining only the top-k partial assignments. By exploiting the problem's

Algorithm 1 Top-k Closest Binary Assignment Generation

```
1: Input: $\mathbf{q^c}$, $N$, $k$
2: Output: Top $k$ binary assignments $\mathbf{B}^*$ minimizing $D(\mathbf{q^c}, \mathbf{b})$
3: $L \leftarrow [(0, [])]$                    ▷ (distance, partial assignment)
4: for $i = 1$ to $N$ do
5:     $T \leftarrow []$                        ▷ Temporary list
6:     for all $(d, a) \in L$ do
7:         $T \leftarrow T \cup (d + q_i^c, a + [0])$          ▷ For $b_i = 0$
8:         $T \leftarrow T \cup (d + (1 - q_i^c), a + [1])$    ▷ For $b_i = 1$
9:     end for
10:    Sort $T$ by $d$ and keep top $k$
11:    $L \leftarrow T$
12: end for
13: Return top $k$ assignments in $L$
```

optimal substructure, the algorithm iteratively builds upon partial solutions to construct the final set of k-nearest discrete solutions.

Consequently, the algorithms from 4.1 and 4.2 serve as advanced discretization techniques for obtaining near-optimal MPE solutions from the continuous neural network outputs. Both techniques can also be applied simultaneously to the neural network outputs, selecting the optimal solution for each query by leveraging the better-performing approach. This combined strategy, referred to as OAUAI | k-Nearest, further improves overall solution quality by leveraging the strengths of both discretization methods.

5 EXPERIMENTS

In this section, we evaluate the HGNN-based embedding (Section 3.2) along with the two thresholding methods presented in Sections 4.1 and 4.2. We benchmark these against several baselines, including both neural network-based and traditional algorithms that directly operate on probabilistic models. We begin by outlining the experimental framework, which includes the competing methods, evaluation metrics, neural network architectures, and probabilistic graphical models used in the study.

5.1 Graphical Models

We evaluated PGMs using thirty high-treewidth binary models from UAI inference competitions (Elidan and Globerson, 2010; Dechter et al., 2022). Among these models, twenty-five contained cliques larger than size two, with variable counts ranging from 360 to 1444 and a maximum clique size of 12. The remaining five were pairwise networks with variable counts ranging from 400 to 1600. For all models, we merged smaller cliques to ensure all cliques were of size ≥ 4, using the method described in 3.2.

We sampled data from the PGMs using Gibbs Sampling, generating 16,000, 2,000, and 2,000 examples for the training, testing, and validation sets, respectively. The query ratio (qr), defined as the fraction of variables in the query set, was varied across $\{0.1, 0.3, 0.5, 0.7, 0.8, 0.9\}$ for each PGM.

5.2 Baseline Methods and Evaluation Criteria

Baselines - We evaluated our methods against both traditional and neural-based MPE solvers. As a traditional baseline, we employed the Max-Product Belief Propagation (MP) algorithm (Pearl, 1988), implemented by Lowd and Rooshenas (2015). The MP algorithm serves as an approximate inference technique designed to identify the Most Probable Explanation (MPE) state.

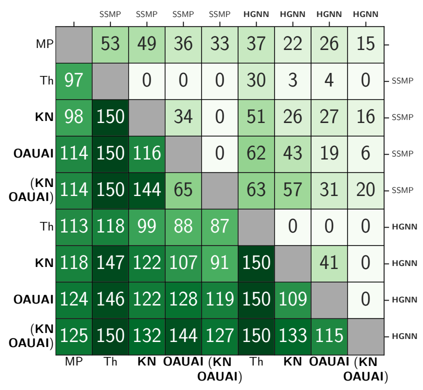
As a neural-based baseline, we employed Self-Supervised learning based MMAP solver for PCs (SSMP) (Arya et al., 2024a), which only encodes the inference query information as input to a neural network and applies thresholding for discretization. This approach enabled comparisons across two dimensions of neural methods: embedding and discretization techniques, which were tested in different combinations during evaluation.

We did not include the inference time optimization scheme (ITSELF) from Arya et al. (2024b) in our comparisons, as it modifies network parameters during inference time to better fit the given query. However, our proposed methods are compatible with ITSELF and could be integrated to benefit from its test time optimization capabilities.

Evaluation Criteria – The competing approaches were assessed based on log-likelihood (LL) scores, calculated as $\ln p_{\mathcal{M}}(\mathbf{e}, \mathbf{q})$, and inference times for a given evidence $\mathbf{e}$ and query output $\mathbf{q}$.

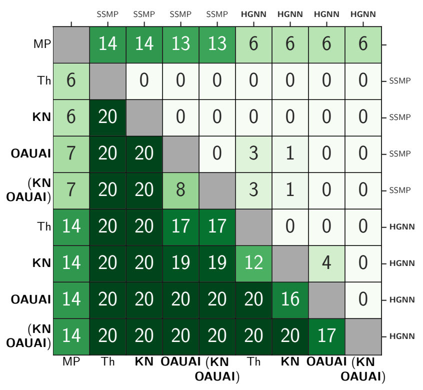
5.3 Advanced Embedding and Discretization Methods

For each PGM and query ratio, we trained a neural network using the embeddings described in 3.2 and the self-supervised loss from 2.4. An attention-based HGNN layer was employed for embedding aggregation, with an input embedding size of 2^k, where k is the number of variables in the largest clique, and an output embedding size of 64. Each model was trained for 20 epochs using the loss function described in 2.4. We standardized the network architecture (for the network that processes the embeddings) across all experiments, using a fully connected NN with three hidden layers (128, 256, and 512 nodes). All hidden layers used ReLU activation functions, while the output layer applied sigmoid functions with dropout regularization (Srivastava et al., 2014). The models were trained using the Adam optimizer (Kingma and Ba, 2015) and implemented in PyTorch (Paszke et al., 2019) and PyTorch Geometric (Fey and Lenssen, 2019). All experiments were conducted on an NVIDIA A40 GPU.



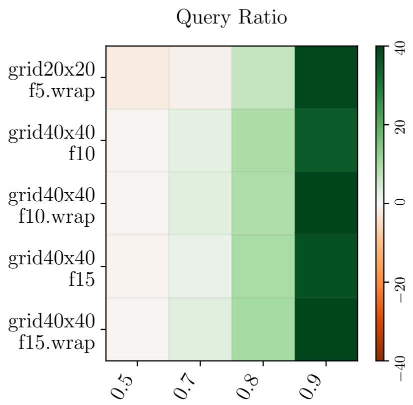
	MP	Th (SSMP)	KN (SSMP)	OAUAI (SSMP)	(KN OAUAI) (SSMP)	Th (HGNN)	KN (HGNN)	OAUAI (HGNN)	(KN OAUAI) (HGNN)
MP		53	49	36	33	37	22	26	15
Th (SSMP)	97		0	0	0	30	3	4	0
KN (SSMP)	98	150		34	0	51	26	27	16
OAUAI (SSMP)	114	150	116		0	62	43	19	6
(KN OAUAI) (SSMP)	114	150	144	65		63	57	31	20
Th (HGNN)	113	118	99	88	87		0	0	0
KN (HGNN)	118	147	122	107	91	150		41	0
OAUAI (HGNN)	124	146	122	128	119	150	109		0
(KN OAUAI) (HGNN)	125	150	132	144	127	150	133	115	

(a) CT: Higher-Order Models



	MP	Th (SSMP)	KN (SSMP)	OAUAI (SSMP)	(KN OAUAI) (SSMP)	Th (HGNN)	KN (HGNN)	OAUAI (HGNN)	(KN OAUAI) (HGNN)
MP		14	14	13	13	6	6	6	6
Th (SSMP)	6		0	0	0	0	0	0	0
KN (SSMP)	6	20		0	0	0	0	0	0
OAUAI (SSMP)	7	20	20		0	3	1	0	0
(KN OAUAI) (SSMP)	7	20	20	8		3	1	0	0
Th (HGNN)	14	20	20	17	17		0	0	0
KN (HGNN)	14	20	20	19	19	12		4	0
OAUAI (HGNN)	14	20	20	20	20	20	16		0
(KN OAUAI) (HGNN)	14	20	20	20	20	20	20	17	

(b) CT: Pairwise Models



(c) HM: Pairwise Models

Figure 2: **Contingency tables** (a) and (b): baseline methods vs. our approach. Each cell: frequency of the row method outperforming the column method. Method names: embedding (right/top) and discretization (left/bottom) techniques. Proposed method names are in bold. **Heatmap** (c) displays % differences in LL scores (color bar) between MP and HGNN: OAUAI | k-Nearest for pairwise models. X-axis: query ratio; Y-axis: dataset names. Green cells: our method is better; Orange cells: MP performs better. Darker shades indicate larger values.

5.4 Hyperparameters for Discretization Methods

We applied four discretization methods: (1) the thresholding (Thresh) technique, which also serves as the baseline discretization method in our experiments, using a threshold of 0.5, as described in Arya et al. (2024a,b), (2) the OAUAI method from 4.1, (3) the k-Nearest method from 4.2, and (4) a combination of the two (OAUAI | k-Nearest). For OAUAI, we limited processing to a maximum of 1/4 of the query variables or 200 (whichever was smaller) using distributed AND/OR Branch and Bound (AOBB) as the chosen oracle, following the method described in Otten and Dechter (2012) method (implementated by Otten (2012)). For k-Nearest, we set k to 1024.

We evaluated the methods on 30 PGMs, including the traditional baseline and neural network-based approaches employing different embedding schemes—query-based baseline embedding (SSMP) and our proposed hypergraph neural network-based embedding (HGNN)—as well as different discretization strategies—thresholding-based baseline (Thresh), our proposed oracle-based method (OAUAI), nearest discrete solutions-based method (k-Nearest), and their combination (OAUAI | k-Nearest).

5.5 Empirical Evaluations

Evaluating Traditional and Neural Baselines vs Our Methods:

Table 2a presents the contingency tables for PGMs with higher-order cliques, with detailed results provided in the supplementary materials. We generated 150 test datasets for the MPE task, utilizing 25 higher-order PGMs across six different query ratios (qr). Each cell (i, j) in the table indicates the number of instances (out of 150) where the method in row i outperformed the method in column j based on average log-likelihood scores. Discrepancies between 150 and the sum of values for cells (i, j) and (j, i) reflect cases where the methods achieved comparable scores.

Similarly, Table 2b displays the contingency tables for pairwise models. Here, we generated 20 test datasets using five PGMs across four query ratios (qr), applying the same interpretative framework as for Table 2a.

Both Tables 2a and 2b use a color scale to illustrate the comparative strength of the methods. Darker green shades indicate that the row method outperformed the column method in a majority of experiments, whereas lighter shades or white (corresponding to a cell value of 0) indicate the opposite, with the column method prevailing more often. Each cell displays its corresponding value. The left and bottom axis labels denote the discretization schemes, while the right and top axis labels indicate the embedding method used.

The analysis of the contingency tables for both higher-order and pairwise models reveals a consistent pattern, where the left side of the diagonal shows higher values, represented by darker green shades. This indicates that methods listed in the lower rows generally outperform those in the earlier columns. This trend underscores the superiority of HGNN-based methods, which are positioned in these lower rows, over alternative approaches. Consequently, HGNN configurations consistently achieve higher log-likelihood scores, surpassing both baseline and SSMP methods in effectiveness. To facilitate a deeper analysis, we will now address several key questions.

Q1. How do HGNN-based methods compare to tra-

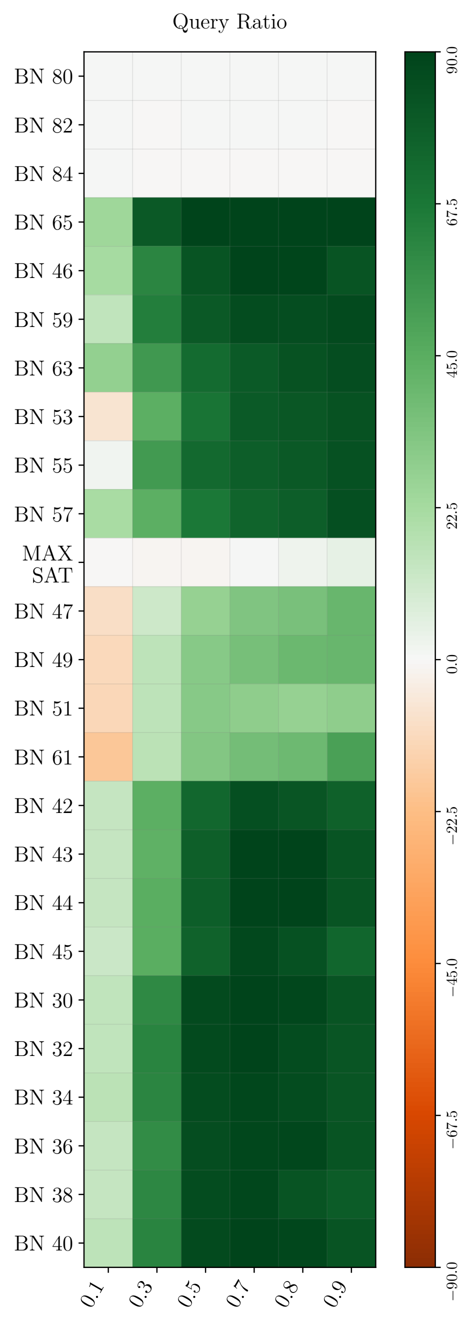


Figure 3: **Heatmap** displaying % differences in LL scores (color bar) between MP and HGNN: OAUAI | k-Nearest for higher-order models. X-axis: query ratio; Y-axis: dataset names. Green cells: our method is better; Orange cells: MP performs better. Darker shades indicate larger values.

ditional baseline MP? The values in the first row and first column provide a comparison between the HGNN-based methods and the traditional MP baseline. Specifically, the last four values in the first column indicate how often the HGNN-based embedding, using different discretization methods, outperformed MP. Among these, HGNN: OAUAI | k-Nearest demonstrated the strongest performance, surpassing the baseline in over 80% of cases for higher-order models and more than 70% for pairwise models. Other methods also displayed competitive performance, indicating that models that use our HGNN-based embedding consistently outperform MP in most scenarios.

Q2. Does a more sophisticated hypergraph-based embedding help? When comparing different methods that use the same discretization technique but differ in their embedding schemes, the HGNN-based embedding consistently outperforms those relying solely on query information. For higher-order networks, the HGNN-based embedding is superior in at least 78% of cases. For pairwise models, it consistently outperforms the alternative method in every instance. This indicates that integrating information from the PGM alongside query data significantly enhances performance compared to embeddings based solely on query information.

Q3. Are advanced discretization techniques more effective than simple thresholding? By examining methods that utilize identical encoding schemes but differ in discretization techniques, we observe OAUAI performs the best following by k-Nearest, and Thresh. Notably, OAUAI | k-Nearest outperforms Thresh in nearly all cases for both embeddings and across higher-order and pairwise models. Furthermore, Thresh, which serves as the baseline for our experiments, does not outperform any of the proposed methods, highlighting the substantial improvements offered by advanced discretization techniques.

Detailed Comparison of Our Best Approach Against MP

To offer a more detailed comparison with the traditional baseline (MP), we use heatmaps in Figures 2c (Pairwise models) and 3 (Higher-order models) to illustrate the performance of HGNN: OAUAI | k-Nearest (our best method) relative to the baseline. Along the y-axis, datasets are arranged by the number of variables, while the x-axis represents different query ratios. Each cell displays the percentage difference in mean log-likelihood (LL) scores between the methods, defined as $\%\text{Diff.} = 100 \times (ll_{nn} - ll_{bp})/|ll_{bp}|$. Green cells indicate where our method outperforms MP, while orange cells denote cases where MP performs better. Darker shades correspond to larger differences, with lighter shades indicating smaller differences; white cells denote no difference.

For pairwise models (Fig. 2c), HGNN: OAUAI | k-Nearest perform comparably to the MP approximation when the query set size is small, as indicated by the light orange cells. However, as the problem complexity increases with larger query set sizes, our method consistently outperforms MP across all datasets, shown by the green color. In these cases, log-likelihood score improvements reach up to 40%.

A similar pattern emerges for higher-order models (Fig.

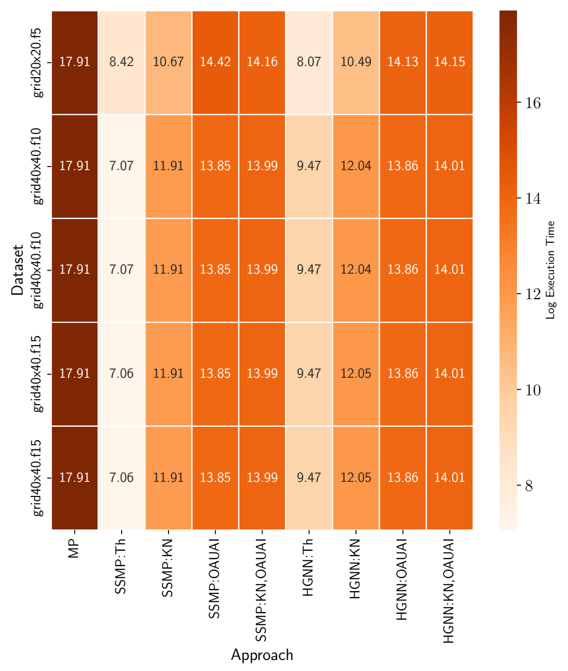


Figure 4: Heatmap of inference times for pairwise PGMs. Color intensity indicates logarithmic microsecond scale, with lighter hues representing faster inference.

3), where our method matches the baseline's performance for the smallest query ratio. However, as the query ratio increases, the performance improvement becomes more evident (up to 90% improvement), as indicated by the prevalence of dark green cells across most datasets. The heatmap features only a few orange and white cells, indicating that our method consistently outperforms the MP approximation in the majority of scenarios.

One key reason for better performance on higher-order networks is that these models inherently encode a large amount of information within each clique, as each clique is represented by 2^k parameters (where k is the size of the clique). This richness enables the HGNN-based model to significantly enhance the quality of MPE solutions. However, when combining cliques (e.g., $k_1 \dots k_m$) to form larger cliques from smaller ones, we transition from $2^{k_1} + 2^{k_2} + \dots + 2^{k_m}$ parameters to $2^{k_1+k_2+\dots+k_m}$ parameters. This increase in parameters can result in embeddings that are less representative. Additionally, the pairwise models used (networks from the grid family) often exhibit similar values in their factors, particularly for the (0,0):(1,1) and (1,0):(0,1) pairs. As a result, when we take the product of cliques, many of the initial embeddings also have similar values, leading to less informative embeddings.

Inference Times

Figures 4 present inference times for all baselines and proposed methods on a logarithmic microsecond scale, where lighter colors indicate faster times. Runtimes for higher-order models appear in the supplement.

For all PGMs, SSMP with Thresh achieves the fastest inference, followed by HGNN with Thresh, with a slight overhead due to evidence instantiation and the attention module. Next are k-Nearest, OAUAI, and HGNN: OAUAI | k-Nearest, maintaining the same ranking across embeddings and trading speed for better solution quality. The MP baseline consistently exhibits the longest inference times.

6 CONCLUSION AND FUTURE WORK

We introduced novel techniques for answering Most Probable Explanation (MPE) queries in probabilistic graphical models, starting with an advanced encoding strategy that employs a HGNN to aggregate information from the query, graphical model structure, and parameters. Additionally, we proposed two novel discretization schemes, k-Nearest and OAUAI, for converting the continuous outputs of the neural network into discrete MPE solutions. These methods eliminates the need for test-time optimization to achieve near-optimal solutions. Furthermore, our methods consistently outperform both traditional baselines and neural network approaches that neither utilize the PGM structure and parameters for embedding computation nor extend beyond thresholding for discretization—a specific case of the k-Nearest method when $k = 1$—as demonstrated by our evaluation on 30 binary high tree-width probabilistic graphical model benchmarks.

Future work includes addressing complex queries in PGMs with constraints; integrating domain-specific prior knowledge during training; boosting performance by developing encoding schemes that achieve HGNN-level accuracy with lower computational requirements; implementing joint training methods that optimize both the encoding network and discretization schemes; etc.

7 ACKNOWLEDGMENTS

This work was supported in part by the DARPA Perceptually-Enabled Task Guidance (PTG) Program under contract number HR00112220005, the DARPA Assured Neuro Symbolic Learning and Reasoning (ANSR) Program under contract number HR001122S0039, the National Science Foundation grant IIS-1652835, and the AFOSR award FA9550-23-1-0239.

References

Allen-Zhu, Z., Li, Y., and Song, Z. (2019). A convergence theory for deep learning via over-parameterization. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine*

Learning, volume 97 of *Proceedings of Machine Learning Research*, pages 242–252. PMLR.

Arora, S., Du, S. S., Hu, W., Li, Z., and Wang, R. (2019). Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 322–332. PMLR.

Arya, D., Gupta, D. K., Rudinac, S., and Worring, M. (2020). Hypersage: Generalizing inductive representation learning on hypergraphs. *CoRR*, abs/2010.04558.

Arya, S., Rahman, T., and Gogate, V. (2024a). Neural Network Approximators for Marginal MAP in Probabilistic Circuits. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(10):10918–10926.

Arya, S., Rahman, T., and Gogate, V. G. (2024b). A neural network approach for efficiently answering most probable explanation queries in probabilistic models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Bai, S., Zhang, F., and Torr, P. H. S. (2019). Hypergraph convolution and hypergraph attention. *Pattern Recognition*.

Cai, D., Song, M., Sun, C., Zhang, B., Hong, S., and Li, H. (2022). Hypergraph structure learning for hypergraph neural networks. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pages 1923–1929, Vienna, Austria. International Joint Conferences on Artificial Intelligence Organization.

Chen, J. and Schwaller, P. (2024). Molecular hypergraph neural networks. *The Journal of Chemical Physics*, 160(14):144307.

Chizat, L. and Bach, F. R. (2018). On the global convergence of gradient descent for over-parameterized models using optimal transport. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 3040–3050.

Choi, Y., Vergari, A., and Van den Broeck, G. (2020). Probabilistic circuits: A unifying framework for tractable probabilistic models. Technical report, University of California, Los Angeles.

Cooper, G. F. (1990). The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405.

de Campos, C. P. (2011). New Complexity Results for MAP in Bayesian Networks. *IJCAI International Joint Conference on Artificial Intelligence*, pages 2100–2106.

Dechter, R. (2019). *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Springer International Publishing, Cham.

Dechter, R., Ihler, A., Gogate, V., Lee, J., Pezeshki, B., Raichev, A., and Cohen, N. (2022). UAI 2022 competition.

Dong, Y., Sawin, W., and Bengio, Y. (2020). HNHN: Hypergraph networks with hyperedge neurons.

Donti, P. L., Rolnick, D., and Kolter, J. Z. (2020). DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*.

Du, S., Zhai, X., Póczos, B., and Singh, A. (2018). Gradient descent provably optimizes over-parameterized neural networks. *International Conference on Learning Representations*.

Elidan, G. and Globerson, A. (2010). The 2010 UAI Approximate inference challenge.

Feng, Y., You, H., Zhang, Z., Ji, R., and Gao, Y. (2019). Hypergraph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):3558–3565.

Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Fioretto, F., Mak, T. W. K., and Hentenryck, P. V. (2020). Predicting AC Optimal Power Flows: Combining Deep Learning and Lagrangian Dual Methods. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):630–637.

Huang, J. and Yang, J. (2021). UniGNN: A unified framework for graph and hypergraph neural networks.

Jacot, A., Hongler, C., and Gabriel, F. (2018). Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 8580–8589.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Lee, J., Schoenholz, S., Pennington, J., Adlam, B., Xiao, L., Novak, R., and Sohl-Dickstein, J. N. (2020a). Finite versus infinite neural networks: an empirical study. *Neural Information Processing Systems*.

Lee, J., Xiao, L., Schoenholz, S. S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. (2020b). Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent. *Journal of Statistical Mechanics: Theory and Experiment*, 2020(12):124002.

Li, K. and Malik, J. (2016). Learning to optimize. *International Conference on Learning Representations*.

Lowd, D. and Rooshenas, A. (2015). The libra toolkit for probabilistic models. *Journal of Machine Learning Research*, 16:2459–2463.

Otten, L. (2012). DAOOPT: Sequential and distributed AND/OR branch and bound for MPE problems.

Otten, L. and Dechter, R. (2012). A case study in complexity estimation: Towards parallel branch-and-bound over graphical models. *Conference on Uncertainty in Artificial Intelligence*.

Park, J. D. and Darwiche, A. (2004). Complexity results and approximation strategies for map explanations. *J. Artif. Int. Res.*, 21(1):101–133.

Park, S. and Hentenryck, P. V. (2023). Self-Supervised Primal-Dual Learning for Constrained Optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(4):4052–4060.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Pearl, J. and Dechter, R. (1990). Identifying independence in bayesian networks. *Networks*, 20(5):507–534.

Qin, T., Etesami, S. R., and Uribe, C. A. (2024). Faster convergence of local SGD for over-parameterized models. *Transactions on Machine Learning Research*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Yadati, N., Nimishakavi, M., Yadav, P., Nitin, V., Louis, A., and Talukdar, P. (2019). HyperGCN: A new method for training graph convolutional networks on hypergraphs. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Zamzam, A. S. and Baker, K. (2020). Learning Optimal Solutions for Extremely Fast AC Optimal Power Flow. In *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pages 1–6.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Not Applicable]
   (b) Complete proofs of all theoretical results. [Not Applicable]
   (c) Clear explanations of any assumptions. [Not Applicable]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Yes]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Yes]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:

(a) The full text of instructions given to participants and screenshots. [Not Applicable]

(b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]

(c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

SINE: Scalable MPE Inference for Probabilistic Graphical Models using Advanced Neural Embeddings: Supplementary Materials

A Experimental Setup

A.1 Probabilistic Graphical Models

Table 1: Dataset Specifications for Higher-Order Probabilistic Graphical Models

Dataset	Number of Variables	Number of Factors	Avg. Clique Size	Max. Clique Size
BN 80	360	360	6.18	12.00
BN 82	360	360	6.18	12.00
BN 84	360	360	6.18	12.00
BN 65	440	440	3.00	5.00
BN 46	499	499	3.40	6.00
BN 59	540	540	3.00	5.00
BN 63	540	540	3.00	5.00
BN 53	561	561	3.00	5.00
BN 55	561	561	3.00	5.00
BN 57	561	561	3.00	5.00
Maxsat aes 64 1 keyfind 1	596	2780	3.00	5.00
BN 47	661	661	3.00	5.00
BN 49	661	661	3.00	5.00
BN 51	661	661	3.00	5.00
BN 61	667	667	3.00	5.00
BN 42	880	880	3.00	5.00
BN 43	880	880	3.00	5.00
BN 44	880	880	3.00	5.00
BN 45	880	880	3.00	5.00
BN 30	1156	1156	2.00	3.00
BN 32	1444	1444	2.00	3.00
BN 34	1444	1444	2.00	3.00
BN 36	1444	1444	2.00	3.00
BN 38	1444	1444	2.00	3.00
BN 40	1444	1444	2.00	3.00

Table 2: Dataset Specifications for Pairwise Probabilistic Graphical Models

Dataset	Number of Variables	Number of Factors	Avg. Clique Size	Max. Clique Size
grid20x20.f5.wrap	400	1201	1.00	2.00
grid40x40.f10	1600	4721	1.00	2.00
grid40x40.f10.wrap	1600	4801	1.00	2.00
grid40x40.f15	1600	4721	1.00	2.00
grid40x40.f15.wrap	1600	4801	1.00	2.00

Tables 1 and 2 summarize the details of the probabilistic graphical models used in the experiments. For higher-order PGMs, the number of variables ranges from 360 to 1444, with an average clique size between 2.00 and 6.18. The maximum clique size varies from 3.00 to 12.00. In contrast, the pairwise PGMs exhibit a more uniform structure across the grid-based datasets, with variables ranging from 400 to 1600 and factors ranging from 1201 to 4801.

A.2 Hyperparameter Configuration and Specifications

We designed our experimental framework to ensure consistency and efficiency across all experiments. For the neural network models, we used a mini-batch size of 512 and applied a learning rate decay of 0.9 whenever the loss plateaued, improving training efficiency. Parameters were initialized using the Kaiming normal distribution to ensure appropriate weight scaling and support effective training.

We implemented the attention-based HGNN using PyTorch Geometric (Fey and Lenssen, 2019), utilizing a single head for the attention module. Optimal hyperparameters were determined through extensive 5-fold cross-validation, while other hyperparameters followed the configurations in Arya et al. (2024a,b).

For the baseline MP method, we utilized the implementation and default parameters from Lowd and Rooshenas (2015). For comparison with optimal solutions, we employed AOBB (Otten and Dechter, 2012), using the implementation from Otten (2012), with default parameters applied. This algorithm also served as the oracle in the OAUAI method.

B Comparative Analysis of Inference Times

We present the inference times for all baselines and proposed methods in Figures 5 and 6. Both figures employ a logarithmic microsecond scale, where color intensity represents inference duration—lighter hues denote shorter times.

For all PGMs, the SSMP method with Thresh and the HGNN method with Thresh exhibit the fastest inference, with SSMP being marginally quicker. This marginal difference arises because the HGNN method requires evidence instantiation to generate richer embeddings. Furthermore, the attention module in HGNN slightly increases inference time.

Following these are the k-Nearest, OAUAI, and HGNN: OAUAI | k-Nearest methods, which maintain the same relative ranking for both embeddings. Although these discretization schemes require more time than the thresholding method (Thresh), they produce significantly better solutions. Reducing the value of the hyperparameters—k for k-Nearest and the number of query variables for OAUAI—can decrease inference time, though at the cost of solution quality. Conversely, increasing the value of these hyperparameters can improve solution quality at the cost of longer inference times. The MP baseline exhibits the longest inference times.

C Comparing MPE Solutions of Proposed Methods Against Near-Optimal Solutions

Table 3: Gap Between AOBB and All Other Methods, Including Baselines and Proposed Methods, for Higher-Order PGMs: Query Ratio 0.5.

PGM	SSMP Th	SSMP KN	SSMP OAUAI	SSMP (KN|OAUAI)	HGNN Th	HGNN KN	HGNN OAUAI	HGNN (KN|OAUAI)	—Worst-Best—
BN 42	0.10524	0.06363	0.06507	0.05166	0.15445	0.08236	0.04370	**0.03969**	0.11476
BN 43	0.21902	0.15704	0.17266	0.14554	0.25150	0.15481	0.04944	**0.04876**	0.20274
BN 44	0.18344	0.13335	0.12052	0.11149	0.10628	0.04523	0.05316	**0.03503**	0.14841
BN 45	0.17915	0.13698	0.11053	0.10570	0.08086	0.03768	0.03712	**0.02808**	0.15107
BN 46	0.28021	0.16378	0.18497	0.14652	0.38591	0.23951	0.16023	**0.14308**	0.24282
BN 65	0.41790	0.37808	0.22508	0.22489	0.11301	0.04416	0.01785	**0.01775**	0.40015
BN 80	0.27192	0.15047	0.08559	0.07923	0.00109	**0.00000**	0.00088	**0.00000**	0.27191
BN 82	0.15444	0.06093	0.08547	0.04847	0.00131	**0.00002**	0.00095	**0.00002**	0.15445
BN 84	0.20071	0.11452	0.15966	0.10912	0.06721	**0.00049**	0.06718	**0.00049**	0.20022
MAX-SAT	0.03481	0.03248	0.02062	0.02062	0.03021	0.02819	**0.01856**	**0.01856**	0.01625

Tables 3 and 4 present the log-likelihood score gaps between AOBB and various neural network techniques, including embedding schemes—query-based (SSMP) and hypergraph neural network-based (HGNN)—as well as discretization schemes: thresholding (Thresh), oracle-based (OAUAI), nearest discrete solutions-based (k-Nearest), and their combination (HGNN: OAUAI | k-Nearest).

For each method M, the gap is calculated as the relative difference between the near-optimal score (determined by AOBB)

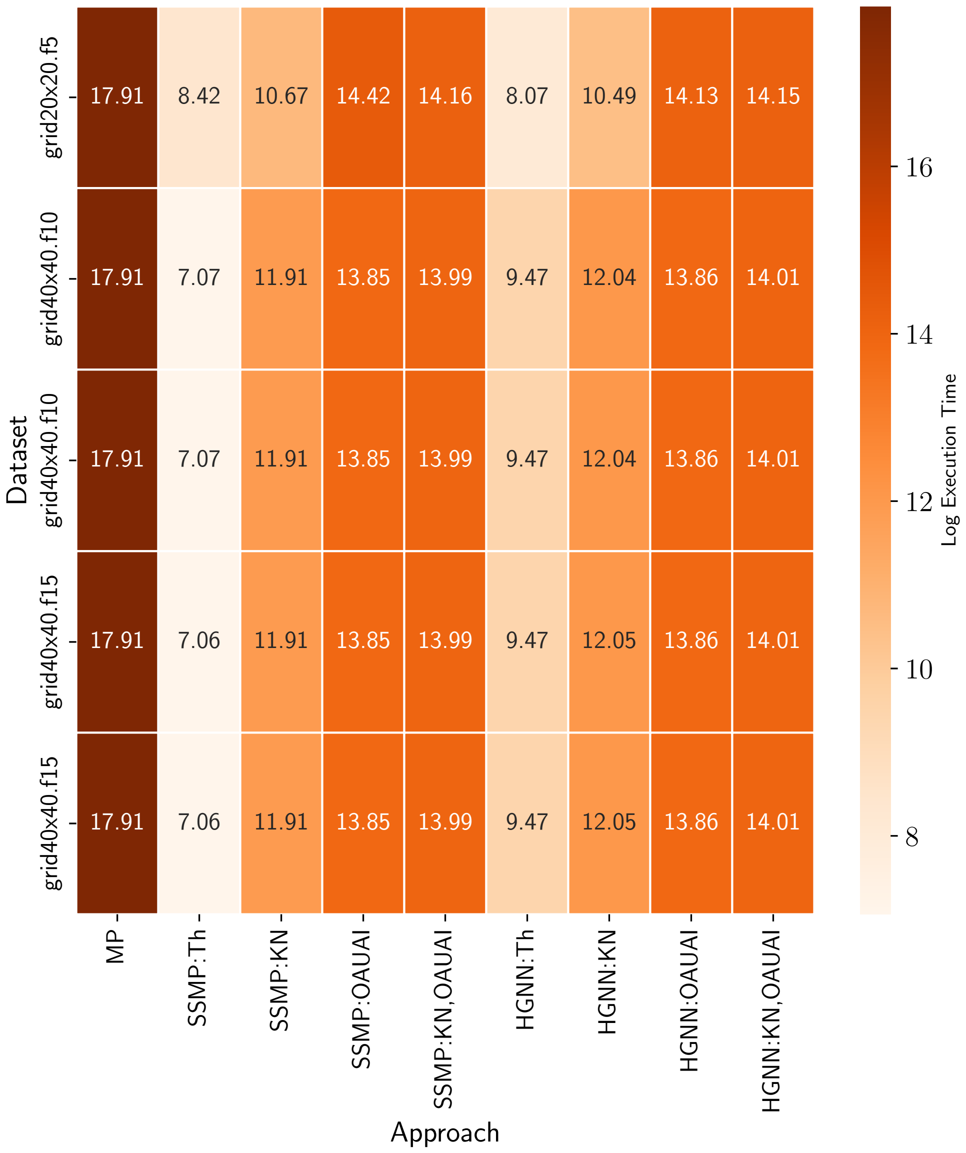
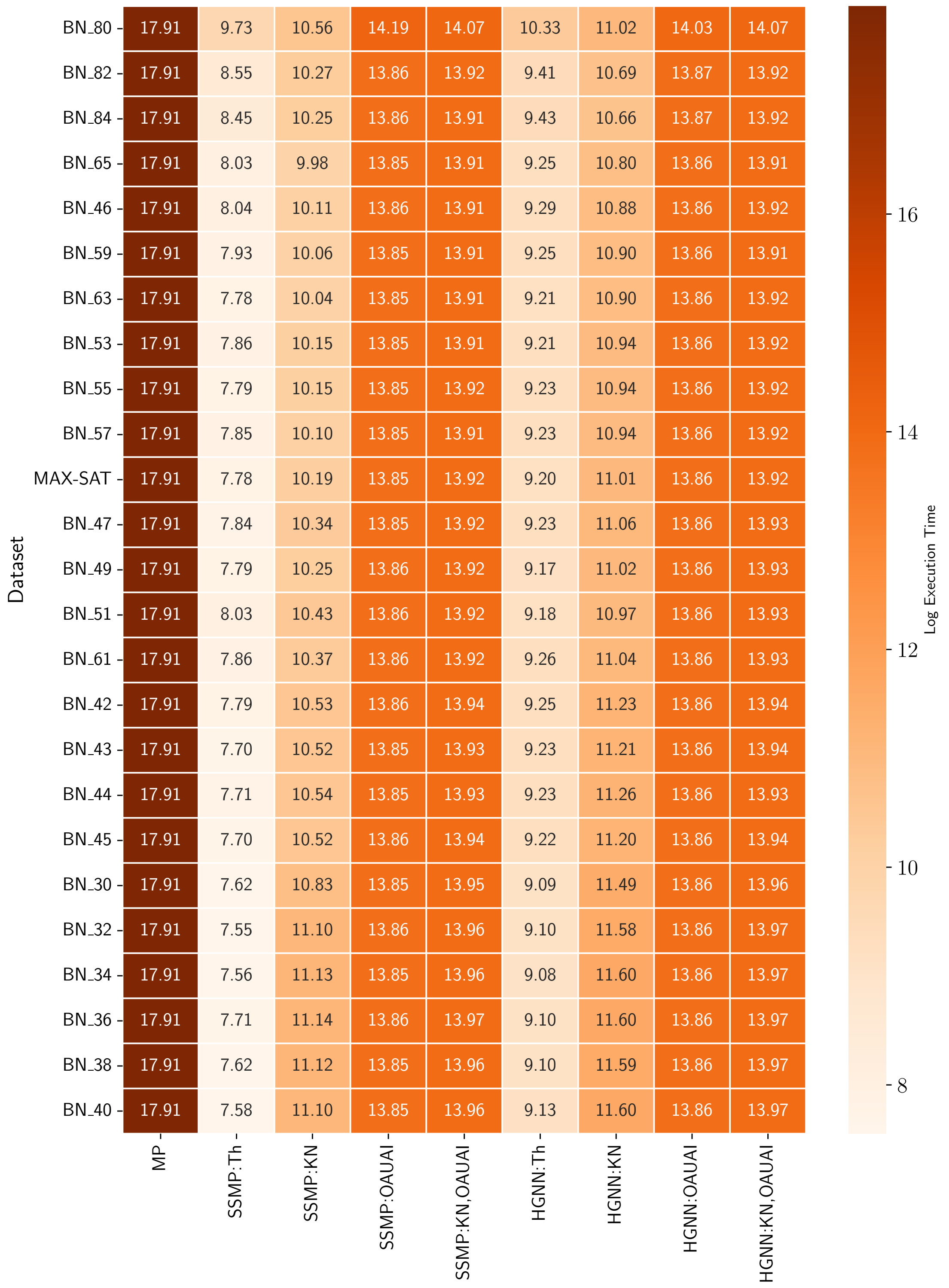


Figure 5: Heatmap of inference times for Pairwise PGMs. The logarithmic microsecond scale is represented by color intensity, with lighter hues indicating shorter, more efficient inference durations.



Dataset	MP	SSMP:Th	SSMP:KN	SSMP:OAUAI	SSMP:KN,OAUAI	HGNN:Th	HGNN:KN	HGNN:OAUAI	HGNN:KN,OAUAI
BN_80	17.91	9.73	10.56	14.19	14.07	10.33	11.02	14.03	14.07
BN_82	17.91	8.55	10.27	13.86	13.92	9.41	10.69	13.87	13.92
BN_84	17.91	8.45	10.25	13.86	13.91	9.43	10.66	13.87	13.92
BN_65	17.91	8.03	9.98	13.85	13.91	9.25	10.80	13.86	13.91
BN_46	17.91	8.04	10.11	13.86	13.91	9.29	10.88	13.86	13.92
BN_59	17.91	7.93	10.06	13.85	13.91	9.25	10.90	13.86	13.91
BN_63	17.91	7.78	10.04	13.85	13.91	9.21	10.90	13.86	13.92
BN_53	17.91	7.86	10.15	13.85	13.91	9.21	10.94	13.86	13.92
BN_55	17.91	7.79	10.15	13.85	13.92	9.23	10.94	13.86	13.92
BN_57	17.91	7.85	10.10	13.85	13.91	9.23	10.94	13.86	13.92
MAX-SAT	17.91	7.78	10.19	13.85	13.92	9.20	11.01	13.86	13.92
BN_47	17.91	7.84	10.34	13.85	13.92	9.23	11.06	13.86	13.93
BN_49	17.91	7.79	10.25	13.86	13.92	9.17	11.02	13.86	13.93
BN_51	17.91	8.03	10.43	13.86	13.92	9.18	10.97	13.86	13.93
BN_61	17.91	7.86	10.37	13.86	13.92	9.26	11.04	13.86	13.93
BN_42	17.91	7.79	10.53	13.86	13.94	9.25	11.23	13.86	13.94
BN_43	17.91	7.70	10.52	13.85	13.93	9.23	11.21	13.86	13.94
BN_44	17.91	7.71	10.54	13.85	13.93	9.23	11.26	13.86	13.93
BN_45	17.91	7.70	10.52	13.86	13.94	9.22	11.20	13.86	13.94
BN_30	17.91	7.62	10.83	13.85	13.95	9.09	11.49	13.86	13.96
BN_32	17.91	7.55	11.10	13.86	13.96	9.10	11.58	13.86	13.97
BN_34	17.91	7.56	11.13	13.85	13.96	9.08	11.60	13.86	13.97
BN_36	17.91	7.71	11.14	13.86	13.97	9.10	11.60	13.86	13.97
BN_38	17.91	7.62	11.12	13.85	13.96	9.10	11.59	13.86	13.97
BN_40	17.91	7.58	11.10	13.85	13.96	9.13	11.60	13.86	13.97

Figure 6: Heatmap of inference times for Higher-Order PGMs. Color intensity represents a logarithmic microsecond scale, with lighter hues indicating shorter inference durations.

Table 4: Gap Between AOBB and All Other Methods, Including Baselines and Proposed Methods, for Pairwise PGMs: Query Ratio 0.5.

PGM	SSMP Th	SSMP KN	SSMP OAUAI	SSMP (KN|OAUAI)	HGNN Th	HGNN KN	HGNN OAUAI	HGNN (KN|OAUAI)	—Worst-Best—
grid20x20.f5.wrap	0.07852	0.06316	0.05627	0.05356	0.05836	0.04185	0.02456	**0.02406**	0.05446
grid40x40.f10	0.01859	0.01642	0.01586	0.01540	0.01312	0.01075	0.01154	**0.01023**	0.00835
grid40x40.f10.wrap	0.01874	0.01645	0.01344	0.01337	0.01124	0.00909	0.00798	**0.00756**	0.01118
grid40x40.f15	0.02260	0.02004	0.01535	0.01533	0.01715	0.01455	**0.01151**	**0.01151**	0.01109
grid40x40.f15.wrap	0.01858	0.01627	0.01323	0.01315	0.01388	0.01388	0.01031	**0.01006**	0.00853

and the score achieved by M. We evaluate the performance on datasets where AOBB can feasibly identify near-optimal or exact solutions. In this experiment, the query ratio is set to 0.5 to improve the quality of the near-optimal solution. Higher query ratios increase problem difficulty, making it more challenging to achieve near-optimal solutions.

The neural-based approach that achieves the best performance for each dataset is highlighted in **bold**. Smaller values indicate better performance, as a smaller gap implies the method is closer to near-optimal solutions. The last column presents the difference between the best (lowest) gap and the worst (highest) gap.

Notably, HGNN: OAUAI | k-Nearest consistently outperforms other neural baselines across most datasets. This method is followed by OAUAI and k-Nearest, both utilizing HGNN for embedding. This analysis offers a comprehensive comparison of the proposed methods with near-optimal solutions, demonstrating that HGNN-based approaches, particularly OAUAI, k-Nearest, and HGNN: OAUAI | k-Nearest, consistently produce solutions with quality close to the optimal solutions.

D Detailed Analysis of Log Likelihood Scores

In this section, we compare the log-likelihood scores across various methods, including the traditional baseline (MP), embedding schemes (query-based embedding (SSMP) and our hypergraph neural network-based embedding (HGNN)), and discretization schemes (thresholding-based (Thresh), oracle-based (OAUAI), nearest discrete solutions-based (k-Nearest), and the combination of the latter two (HGNN: OAUAI | k-Nearest)).

Figures 7 to 31 present the log-likelihood plots for higher-order PGMs, while Figures 32 to 36 show those for pairwise PGMs. Each bar indicates the mean log-likelihood score of the respective method, with tick marks representing the standard deviation. Higher log-likelihood values correspond to better performance.

D.1 Comparison for Higher-Order PGMs

Figures 7 to 31 present the log-likelihood scores for higher-order PGMs, highlighting the performance of our proposed HGNN-based embedding and novel discretization schemes—oracle-based (OAUAI) and nearest discrete solutions-based (k-Nearest)—compared to other baselines. These figures further validate the superior performance of HGNN embedding-based methods over the traditional query-based embedding (SSMP), as demonstrated by the heatmaps and contingency tables in the main paper. Additionally, the results show that OAUAI and k-Nearest outperform the traditional thresholding method (Thresh). These visualizations provide a clear comparison of the performance between baseline methods and our proposed methods across different higher-order PGMs and query ratios.

D.2 Comparison for Pairwise PGMs

Analyzing Figures 32 to 36, which focus on pairwise PGMs, reveals similar trends. The neural-based methods significantly outperform the MP baseline at higher query ratios. Among these, OAUAI and HGNN: OAUAI | k-Nearest outperform all baselines and neural methods across all larger query ratios. Additionally, all HGNN-based embeddings consistently surpass the baseline embedding (SSMP), highlighting the enhanced representational power of HGNN. When comparing traditional discretization methods with k-Nearest, OAUAI, and HGNN: OAUAI | k-Nearest, our proposed methods consistently demonstrate superior performance. Even with the same trained neural network used for inference, advanced discretization methods consistently outperform basic thresholding. These results underscores the superior effectiveness of these advanced techniques in improving model performance.

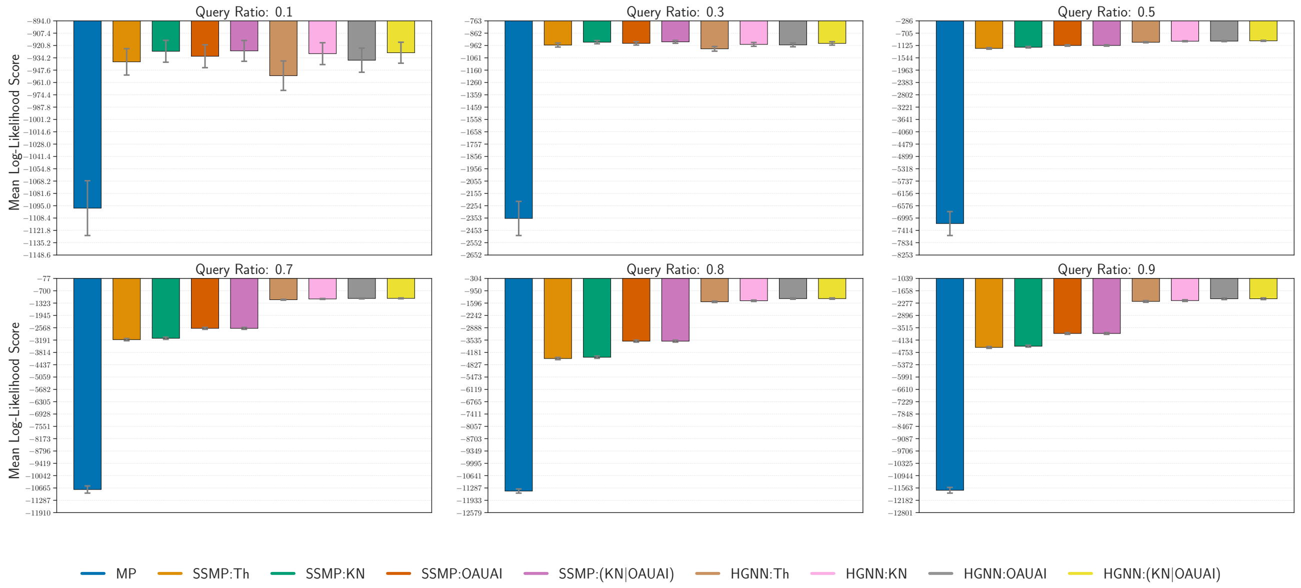


Figure 7: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 30. Higher Scores Reflect Superior Model Performance.

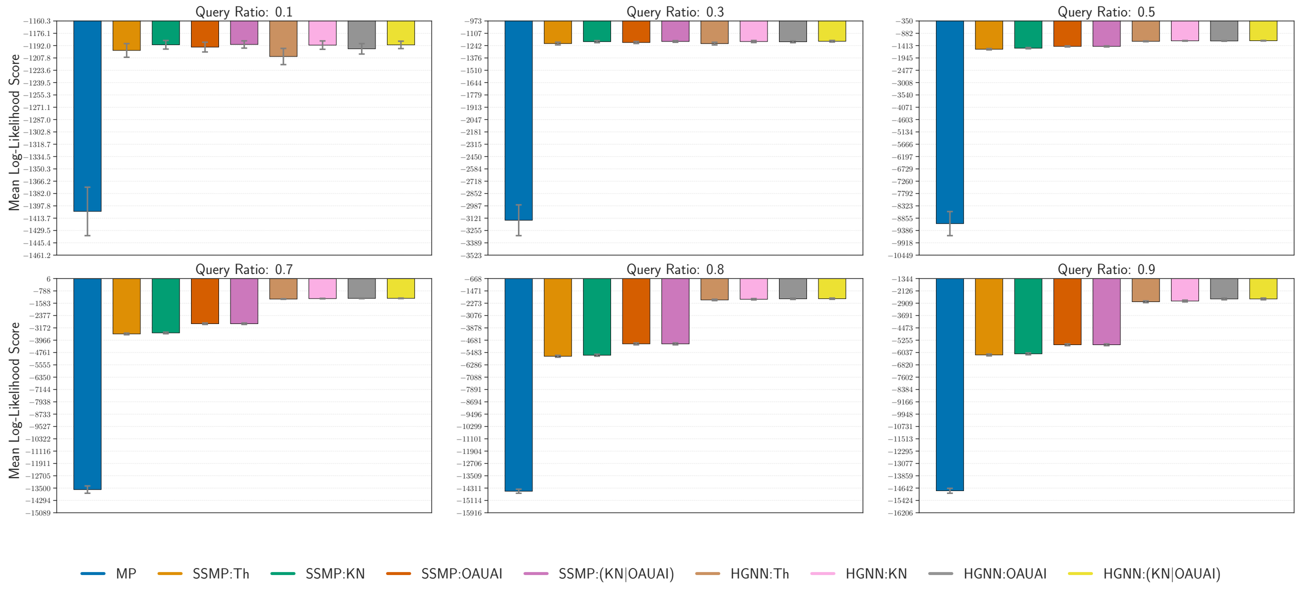


Figure 8: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 32. Higher Scores Reflect Superior Model Performance.

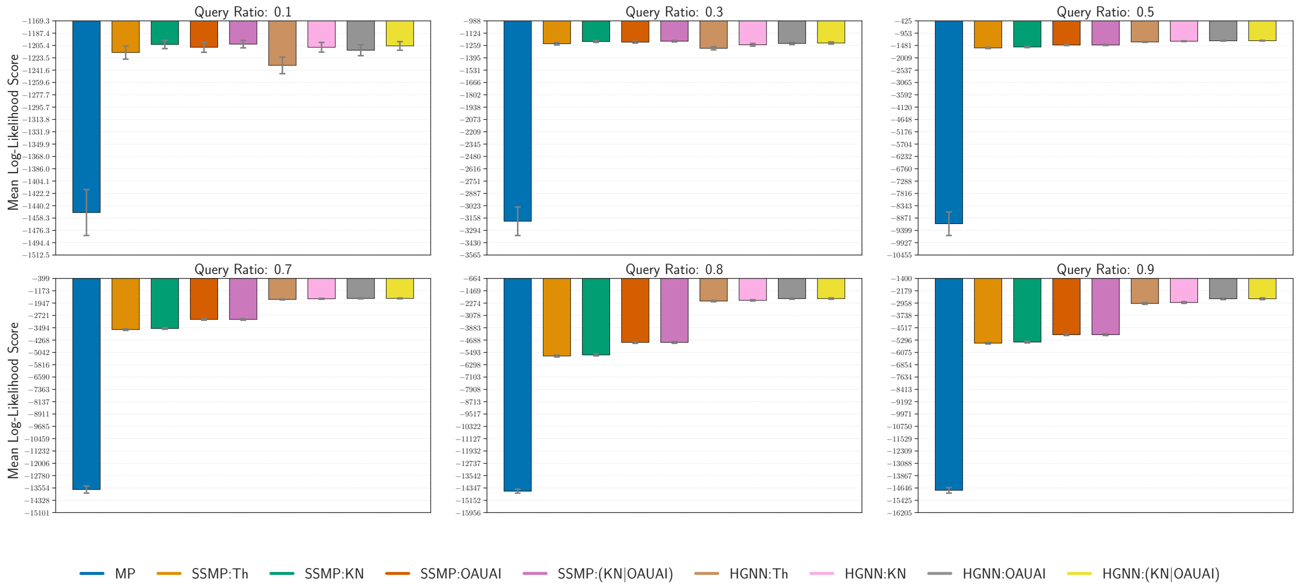


Figure 9: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 34. Higher Scores Reflect Superior Model Performance.

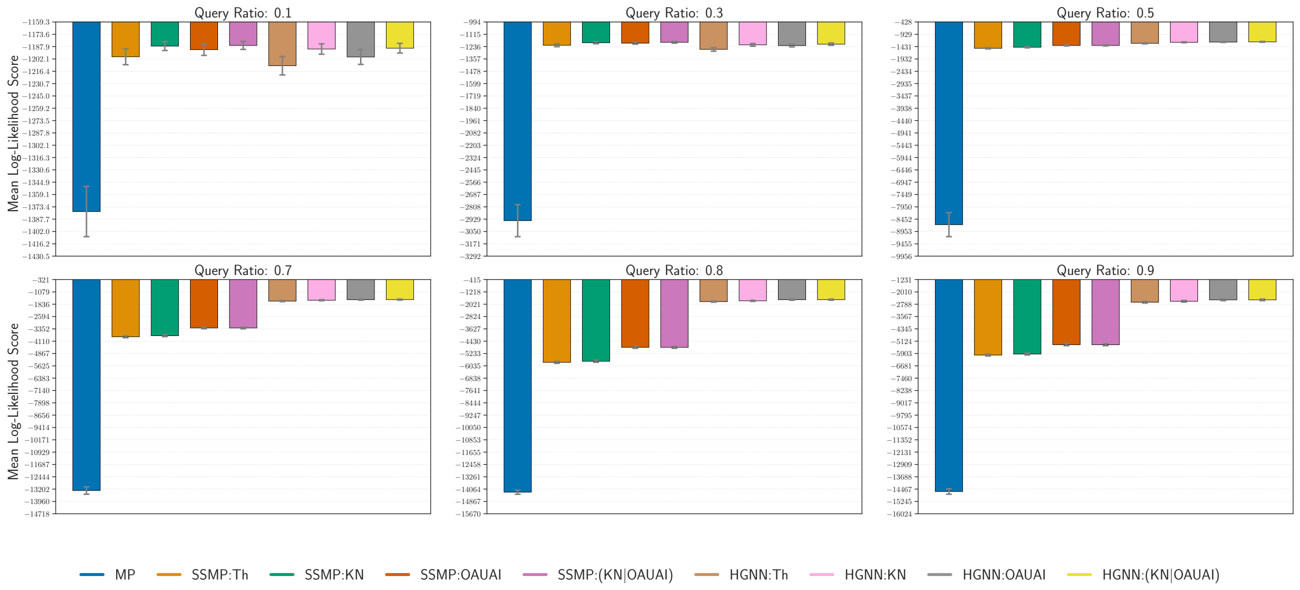


Figure 10: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 36. Higher Scores Reflect Superior Model Performance.

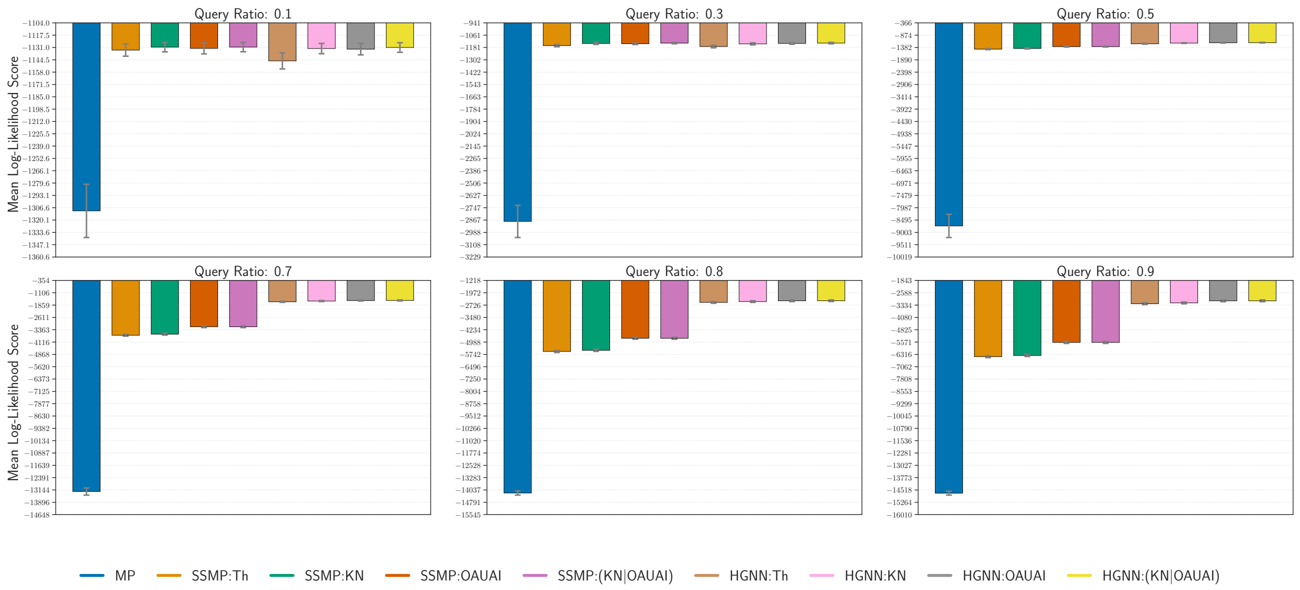


Figure 11: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 38. Higher Scores Reflect Superior Model Performance.

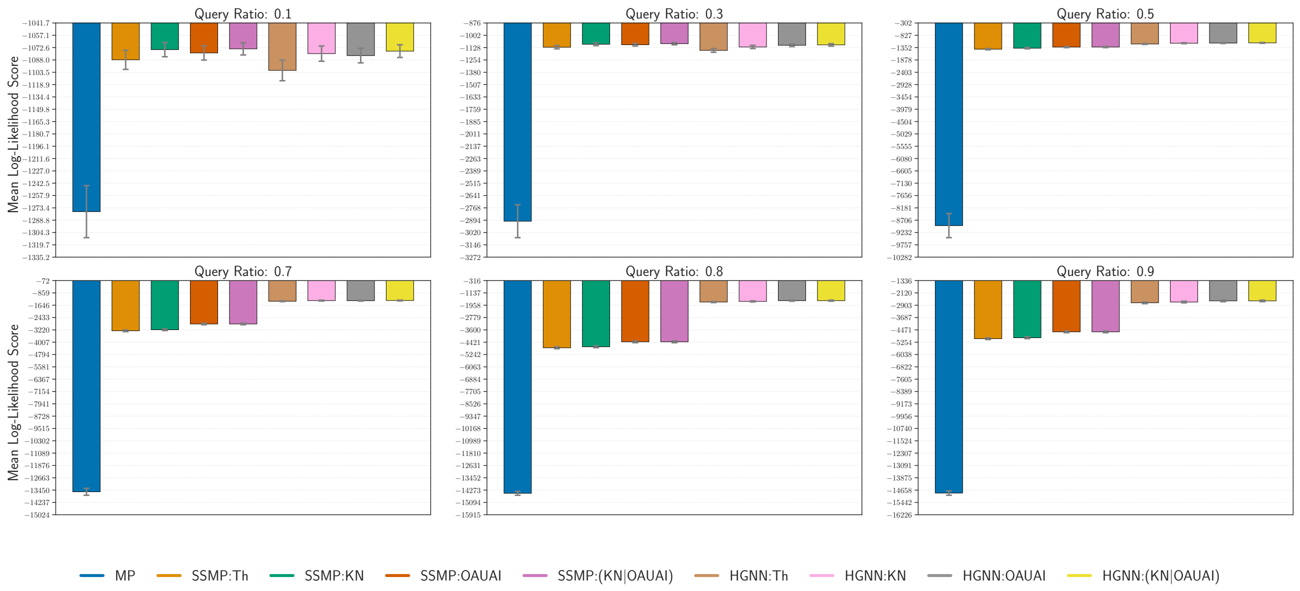


Figure 12: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 40. Higher Scores Reflect Superior Model Performance.

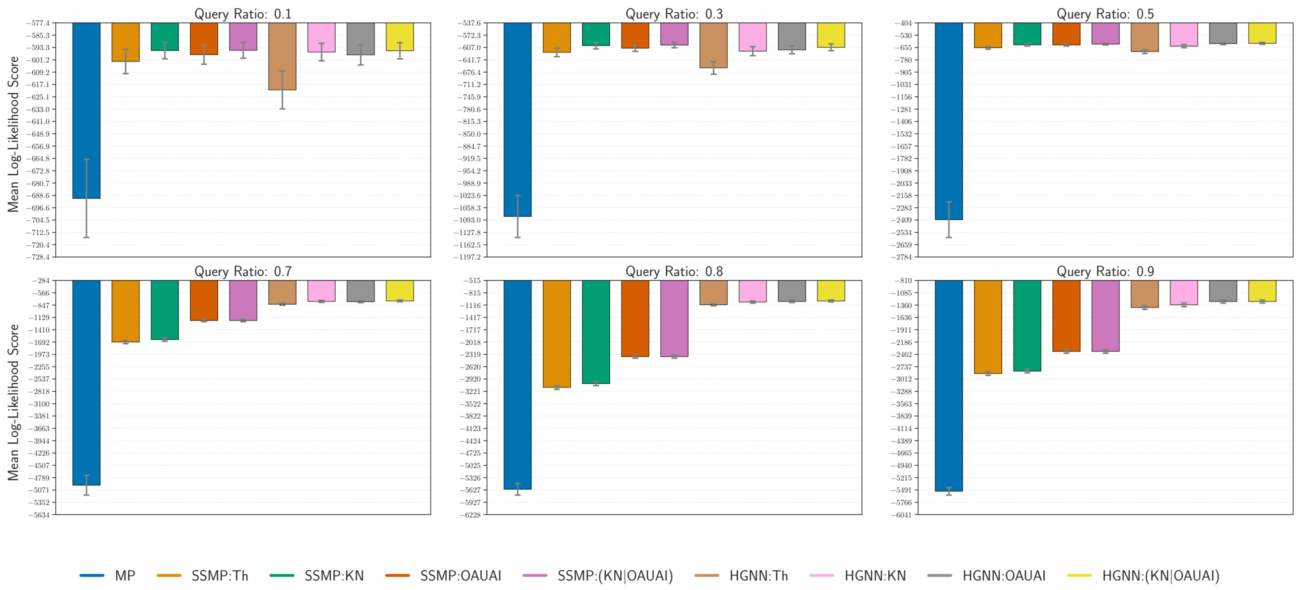


Figure 13: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 42. Higher Scores Reflect Superior Model Performance.

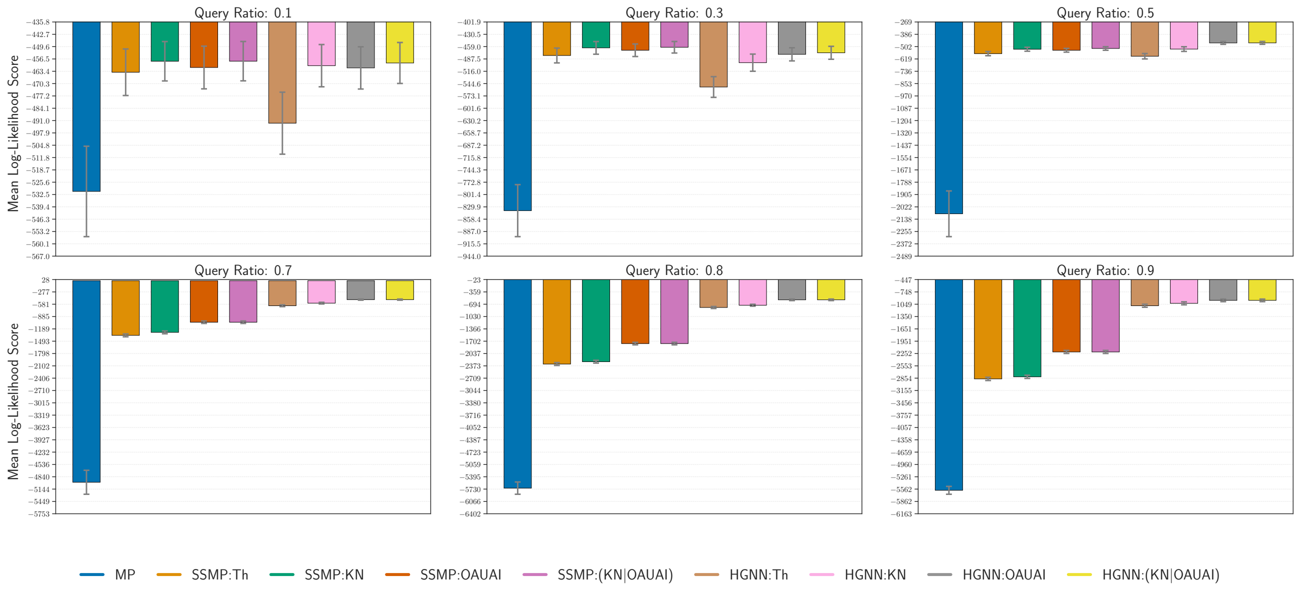


Figure 14: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 43. Higher Scores Reflect Superior Model Performance.

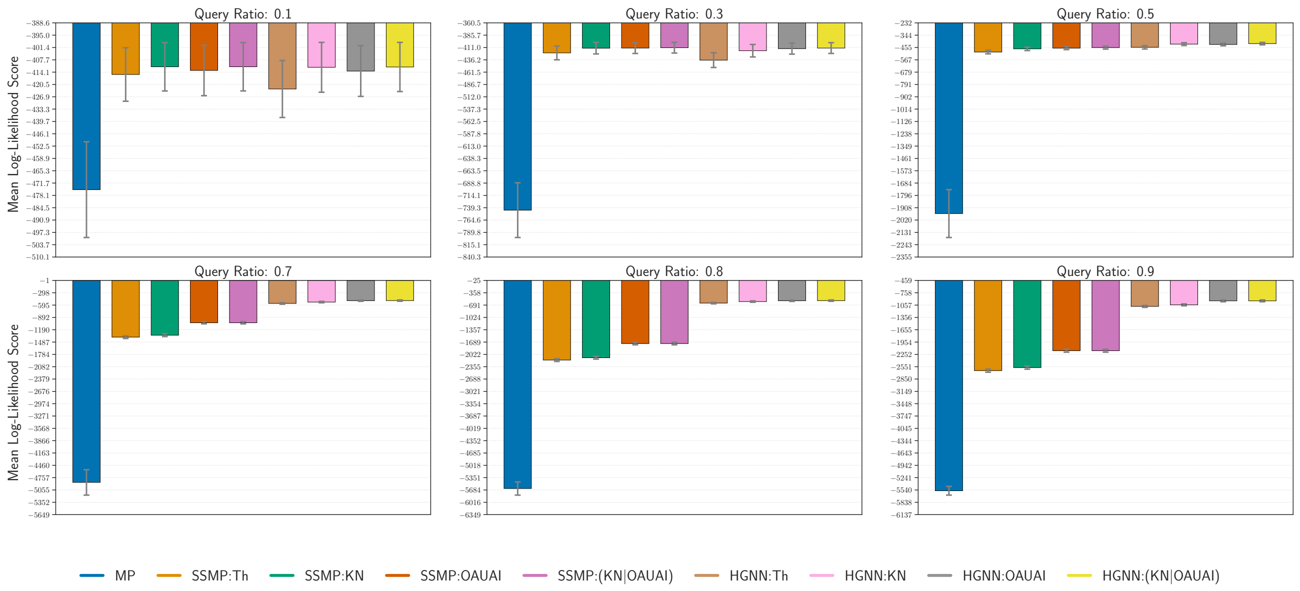


Figure 15: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 44. Higher Scores Reflect Superior Model Performance.

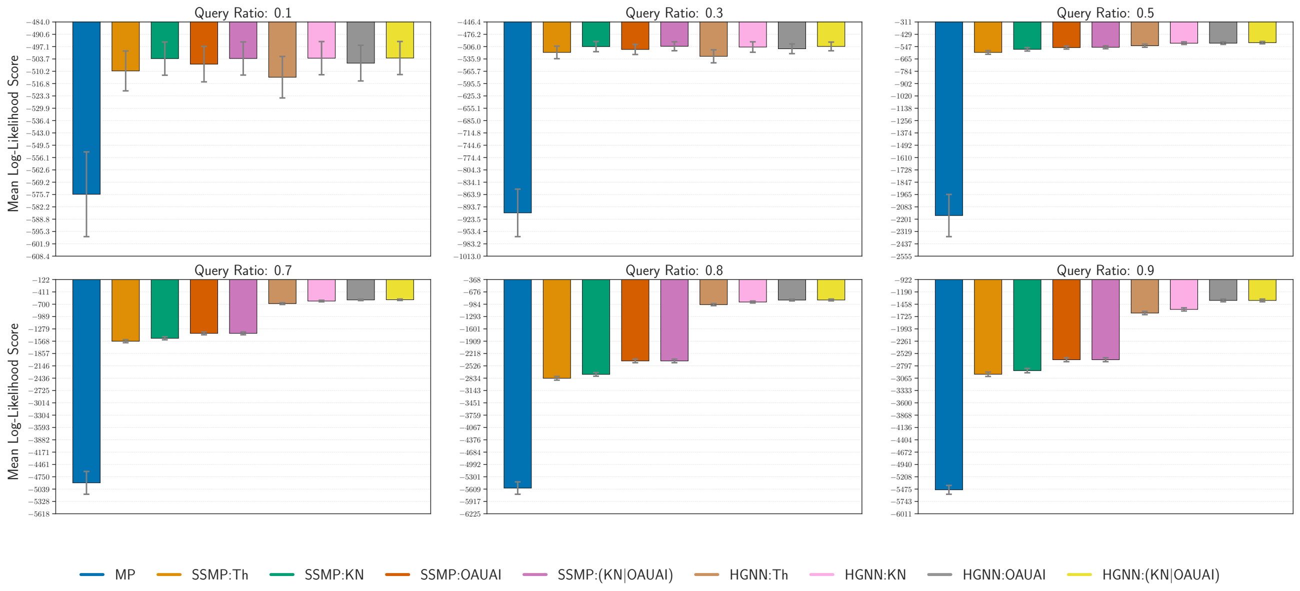


Figure 16: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 45. Higher Scores Reflect Superior Model Performance.

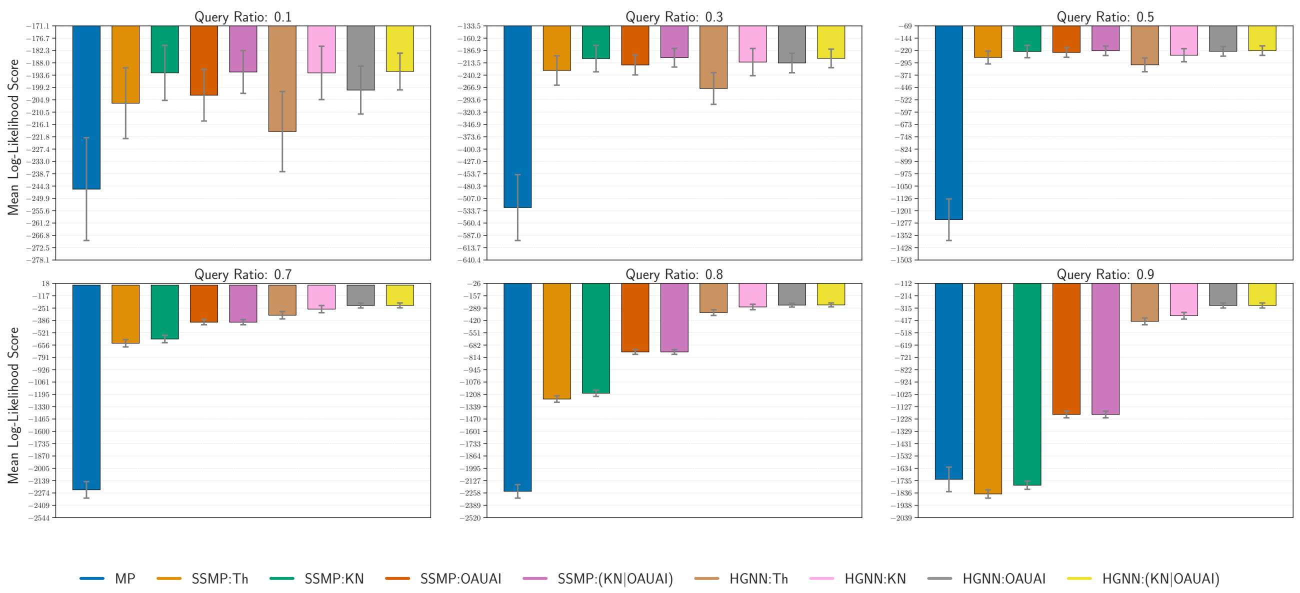


Figure 17: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 46. Higher Scores Reflect Superior Model Performance.

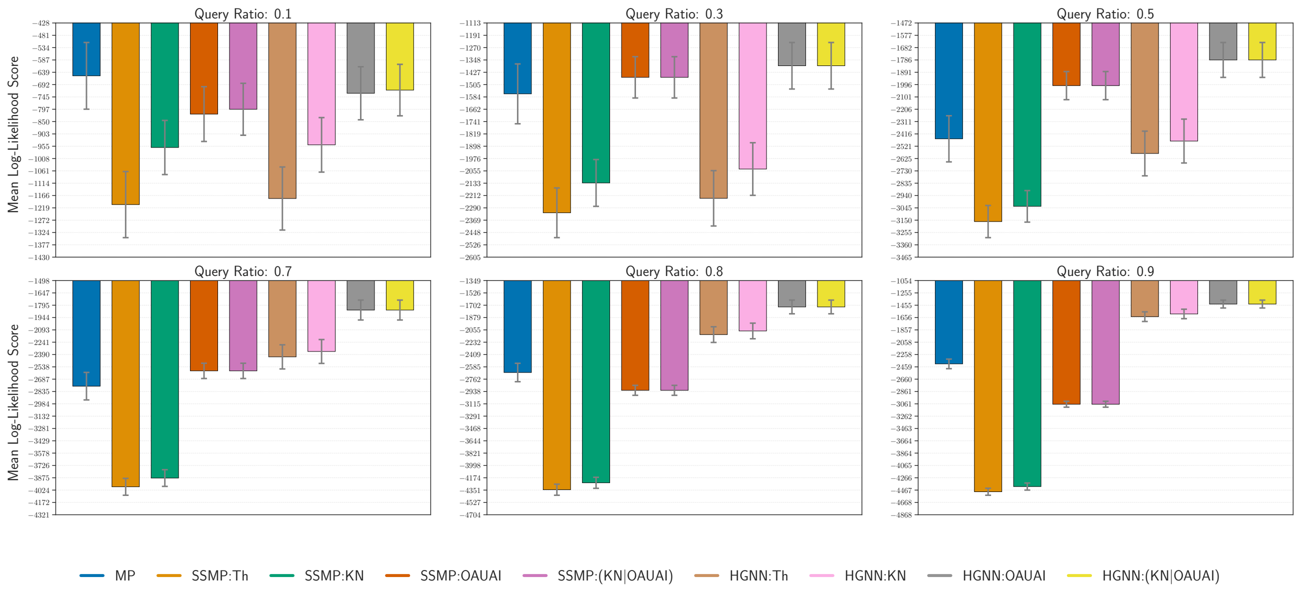


Figure 18: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 47. Higher Scores Reflect Superior Model Performance.

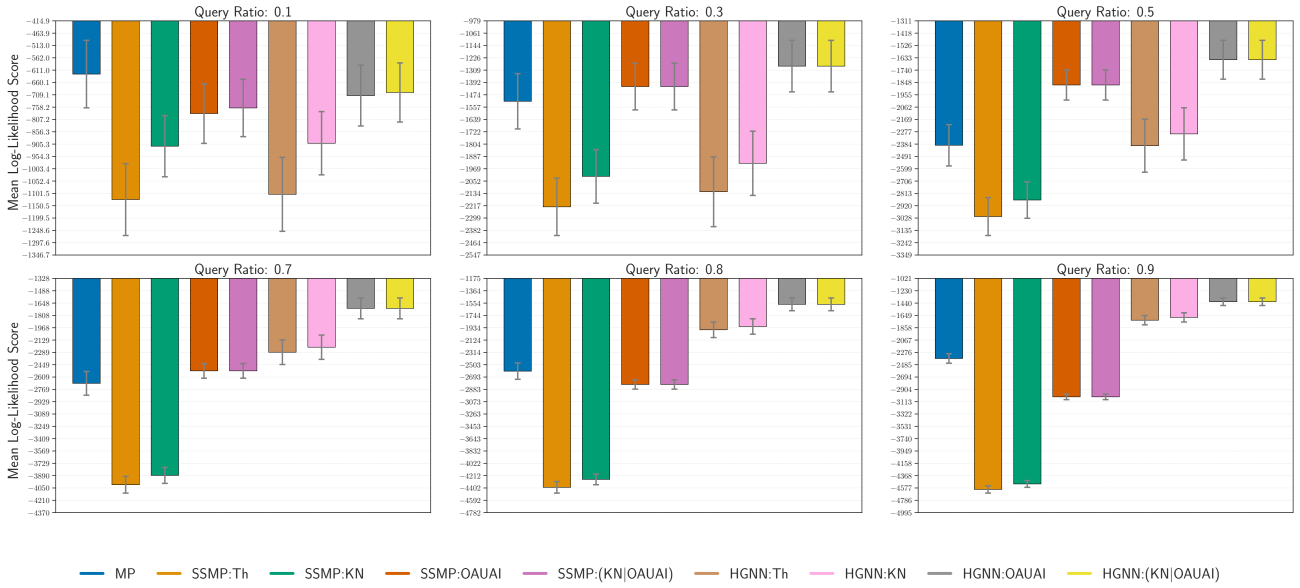


Figure 19: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 49. Higher Scores Reflect Superior Model Performance.

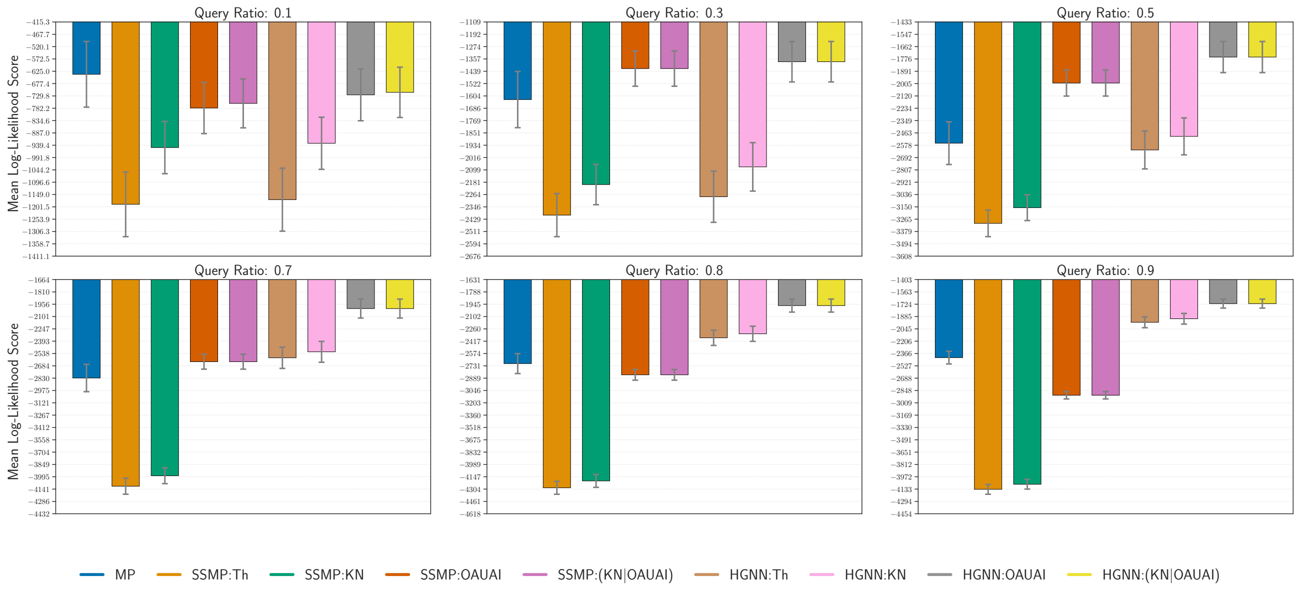


Figure 20: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 51. Higher Scores Reflect Superior Model Performance.

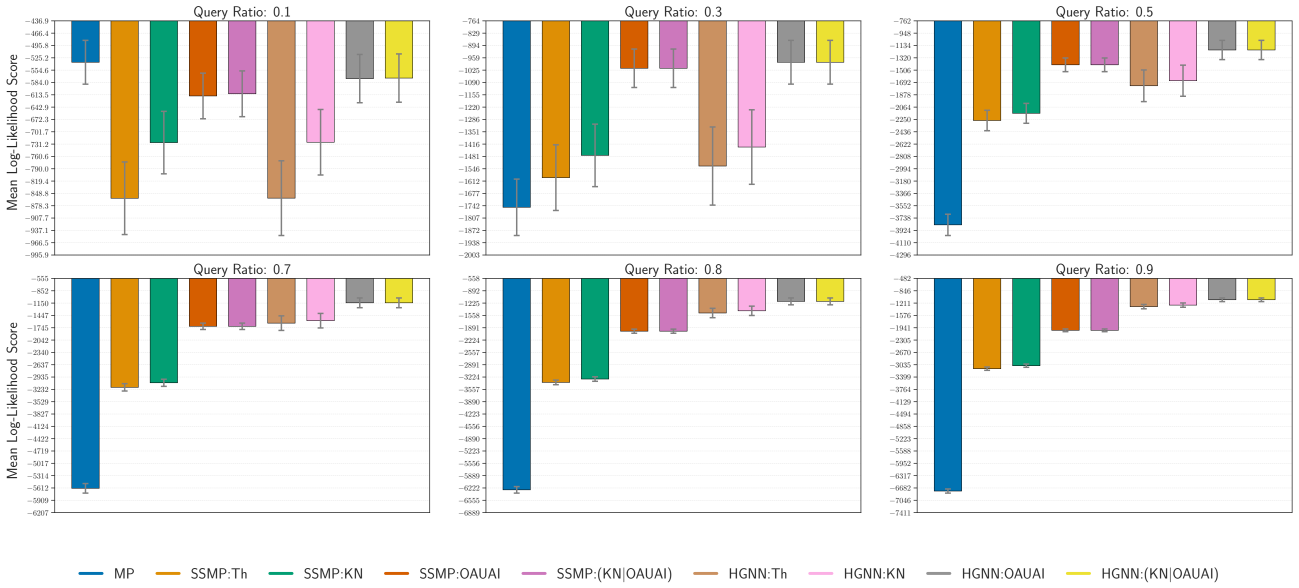


Figure 21: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 53. Higher Scores Reflect Superior Model Performance.

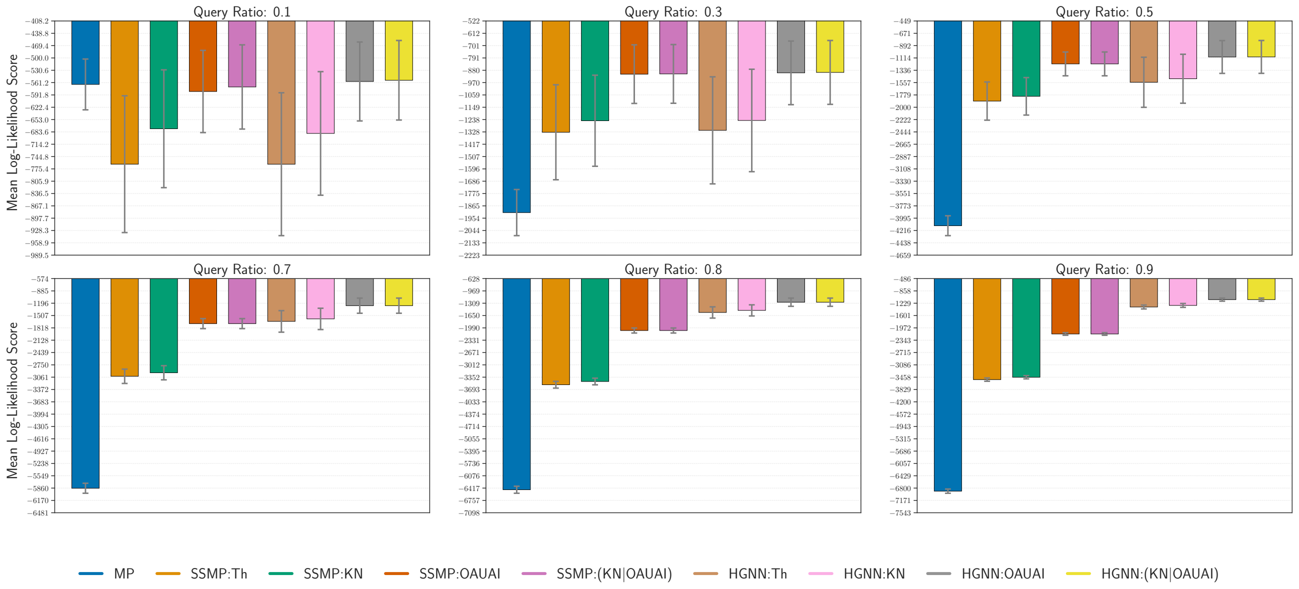


Figure 22: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 55. Higher Scores Reflect Superior Model Performance.

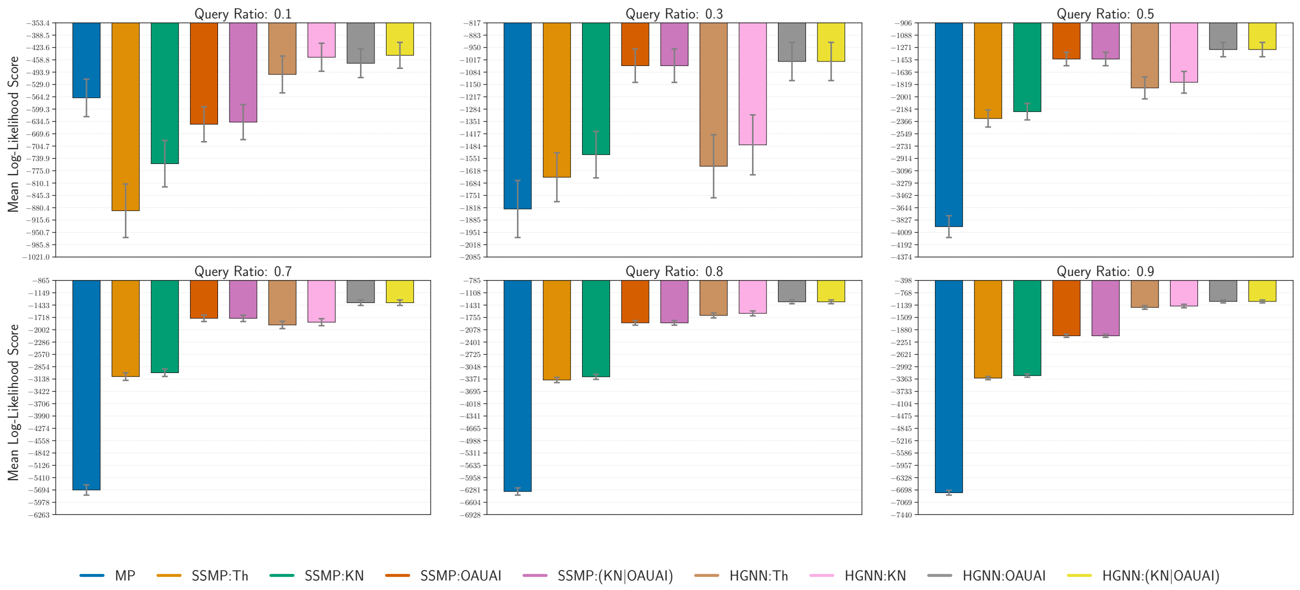


Figure 23: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 57. Higher Scores Reflect Superior Model Performance.

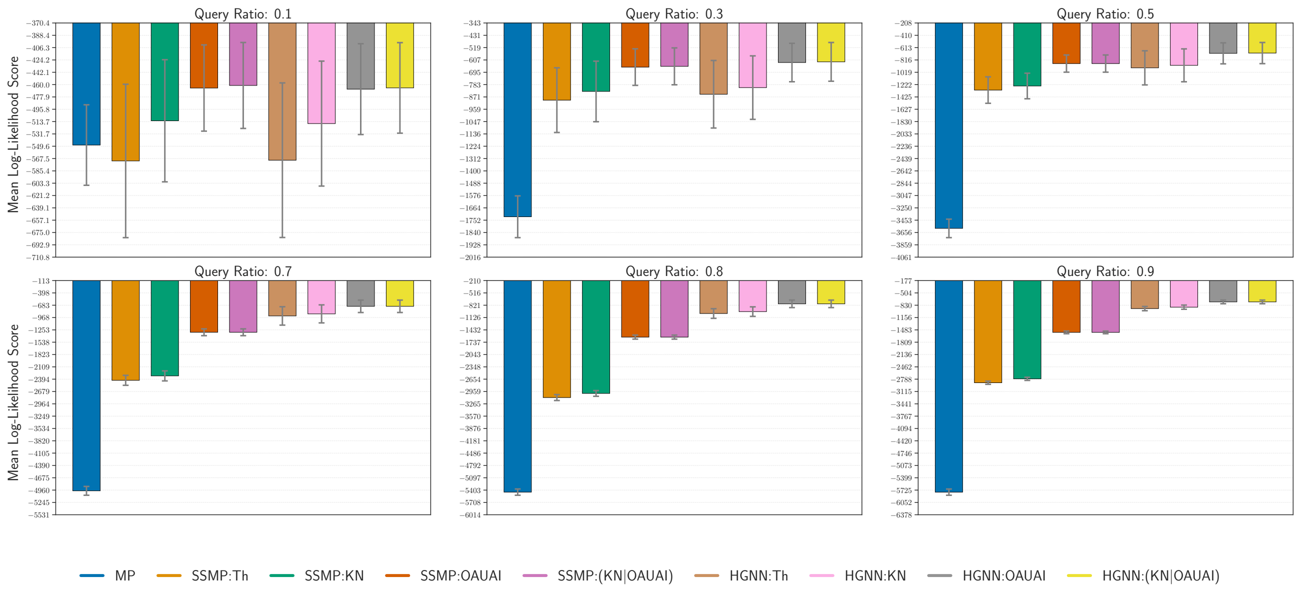


Figure 24: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 59. Higher Scores Reflect Superior Model Performance.

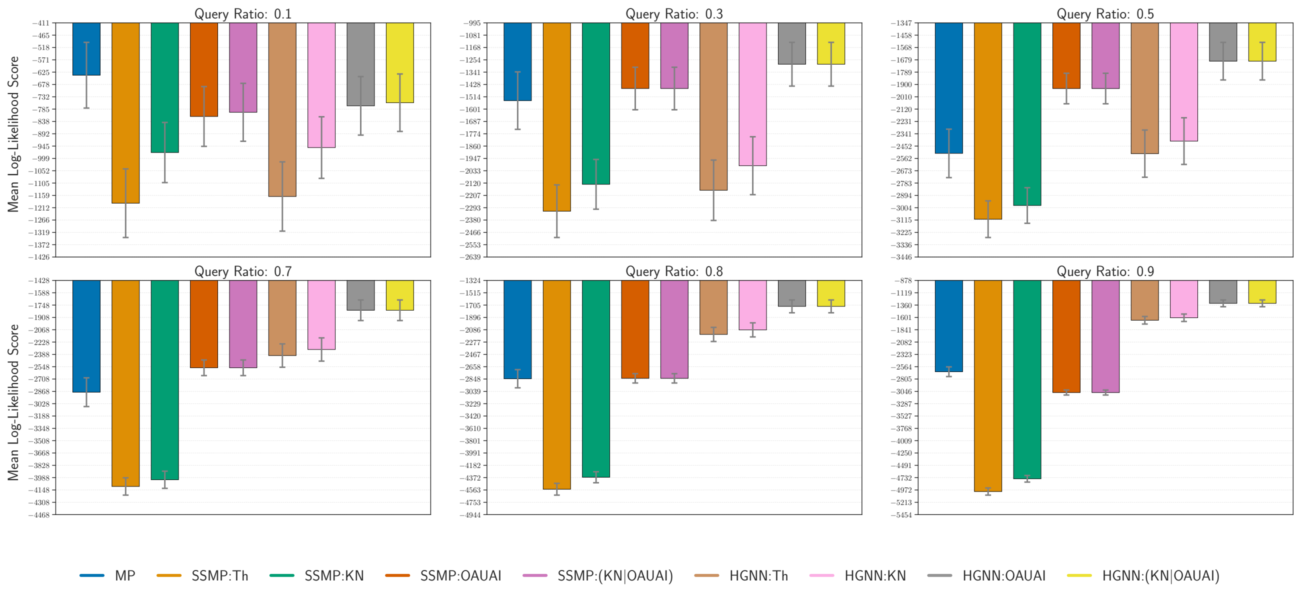


Figure 25: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 61. Higher Scores Reflect Superior Model Performance.

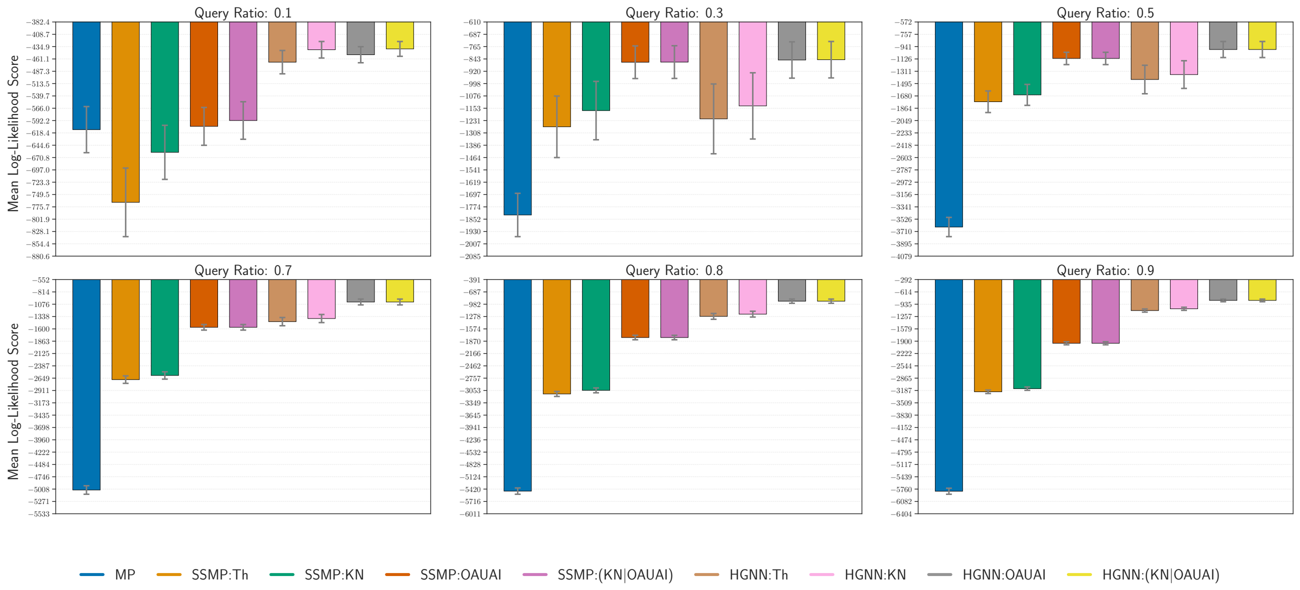


Figure 26: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 63. Higher Scores Reflect Superior Model Performance.

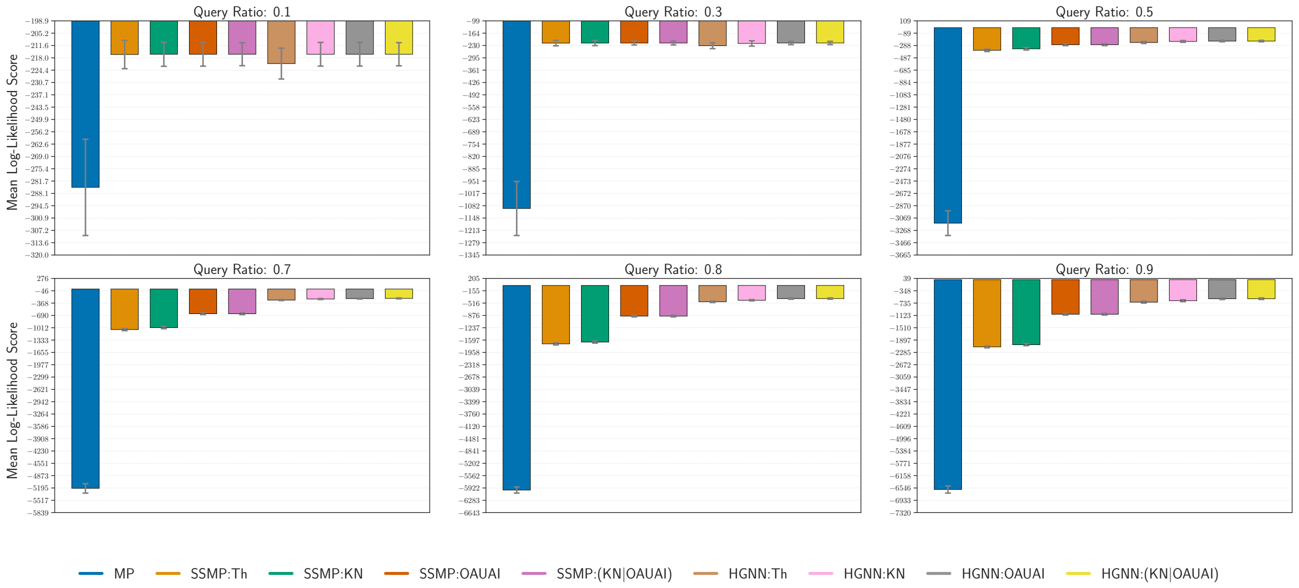


Figure 27: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 65. Higher Scores Reflect Superior Model Performance.

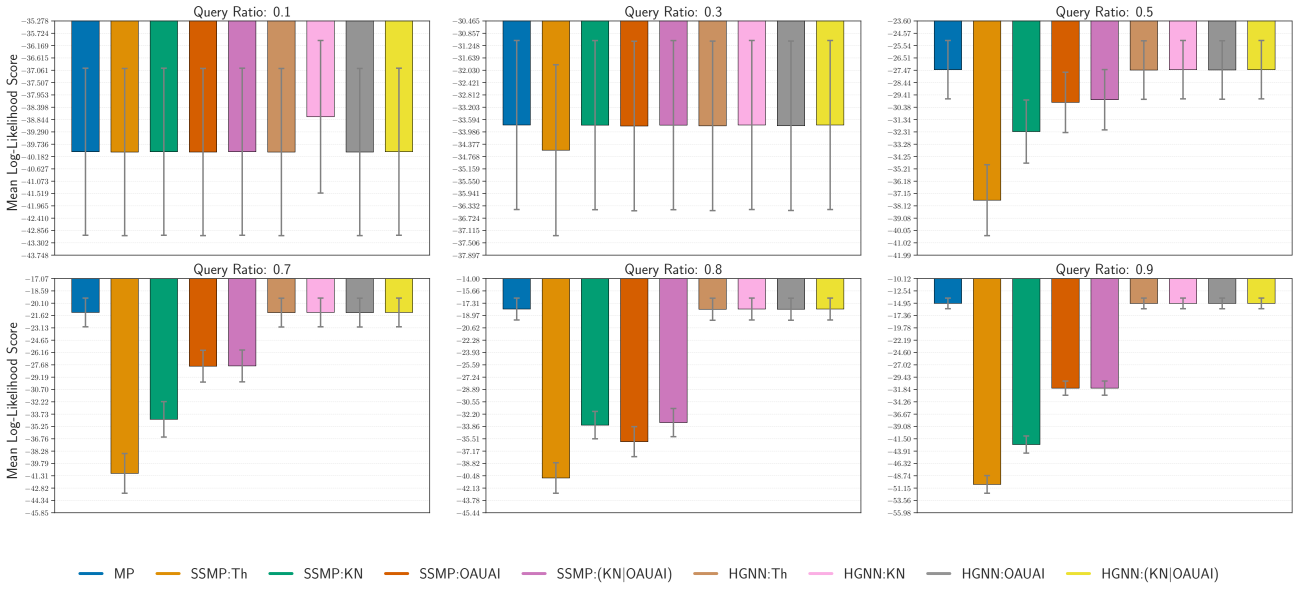


Figure 28: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 80. Higher Scores Reflect Superior Model Performance.

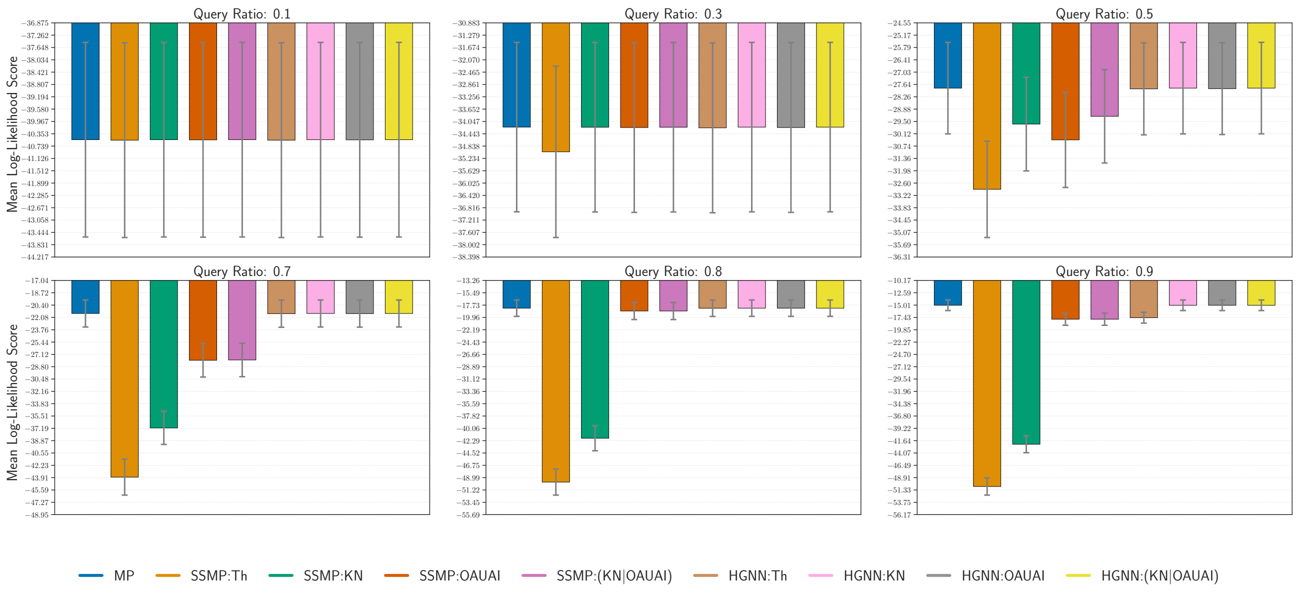


Figure 29: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 82. Higher Scores Reflect Superior Model Performance.

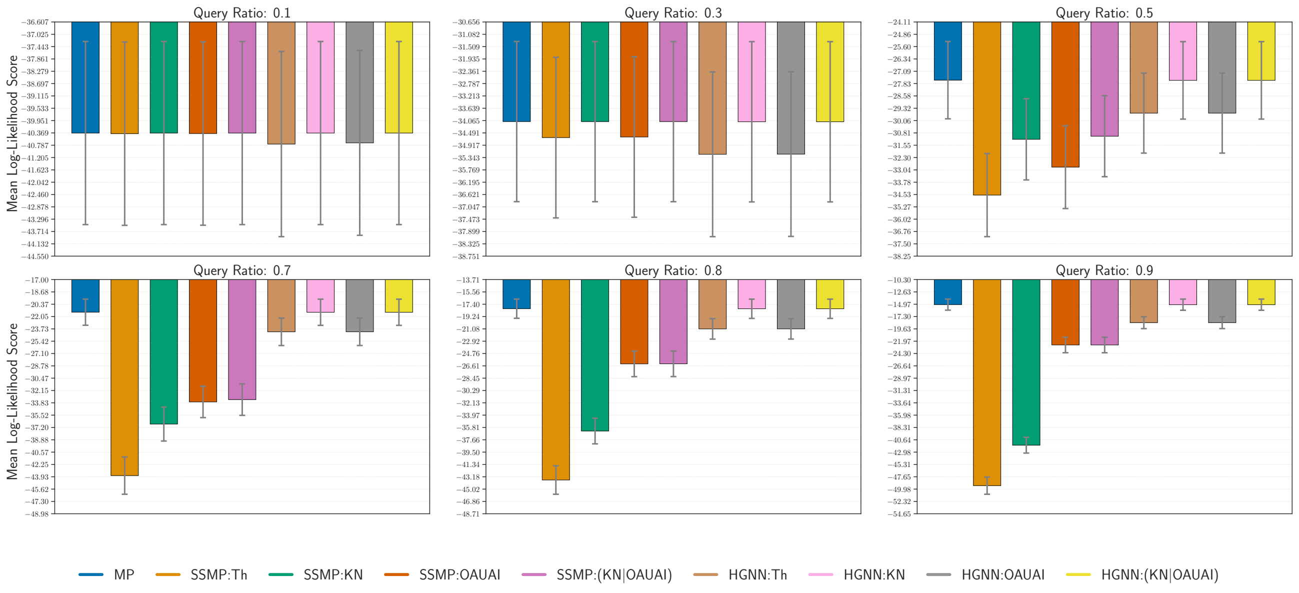


Figure 30: Evaluating Higher-Order PGMs: Log-Likelihood Scores on BN 84. Higher Scores Reflect Superior Model Performance.

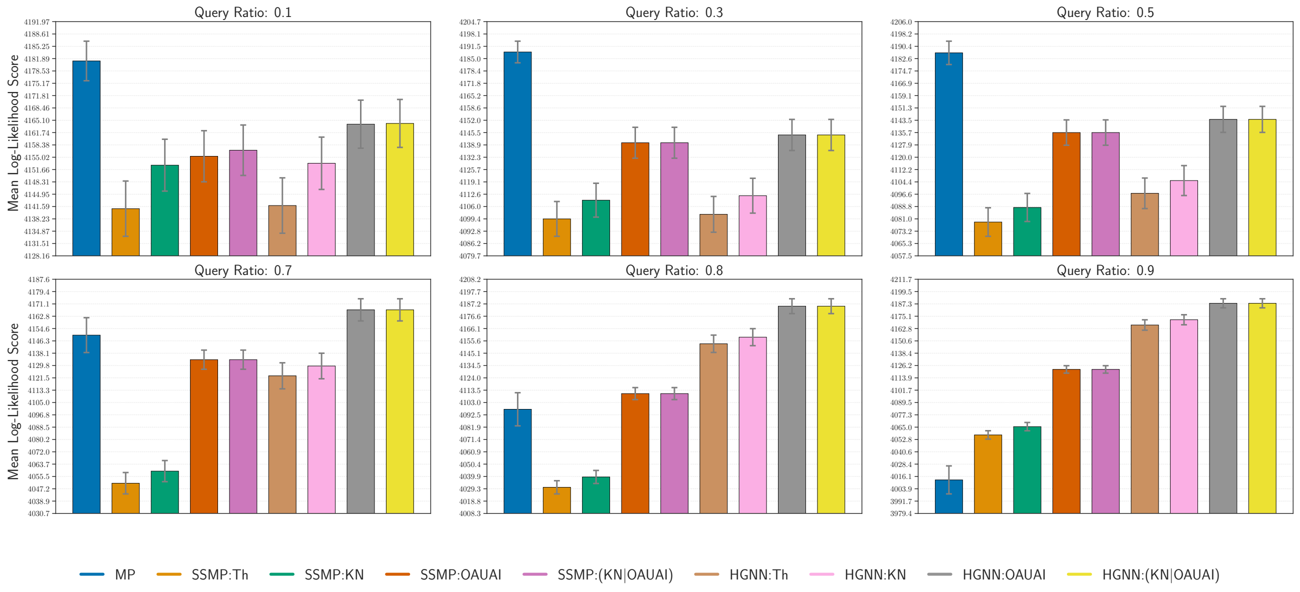


Figure 31: Evaluating Higher-Order PGMs: Log-Likelihood Scores on Maxsat aes 64 1 keyfind 1. Higher Scores Reflect Superior Model Performance.

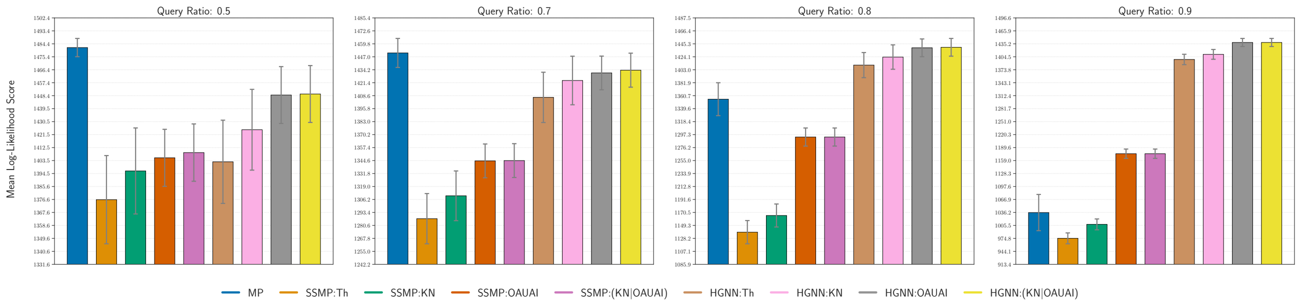


Figure 32: Evaluating Pairwise PGMs: Log-Likelihood Scores on grid20x20.f5.wrap. Higher Scores Reflect Superior Model Performance.

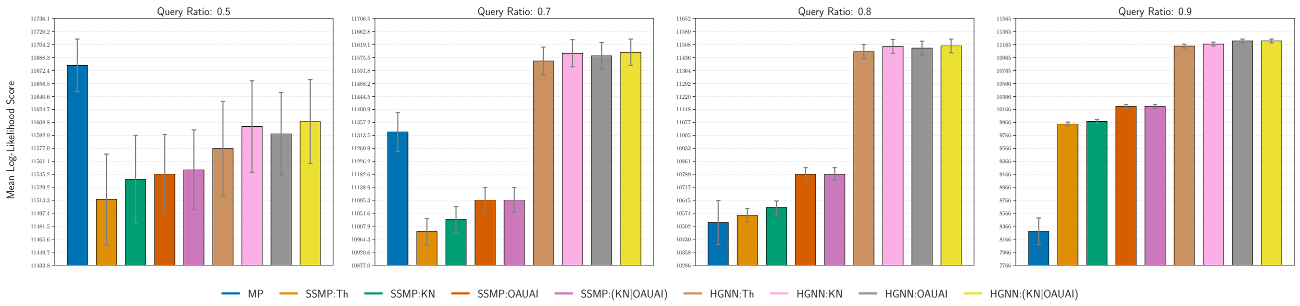


Figure 33: Evaluating Pairwise PGMs: Log-Likelihood Scores on grid40x40.f10. Higher Scores Reflect Superior Model Performance.

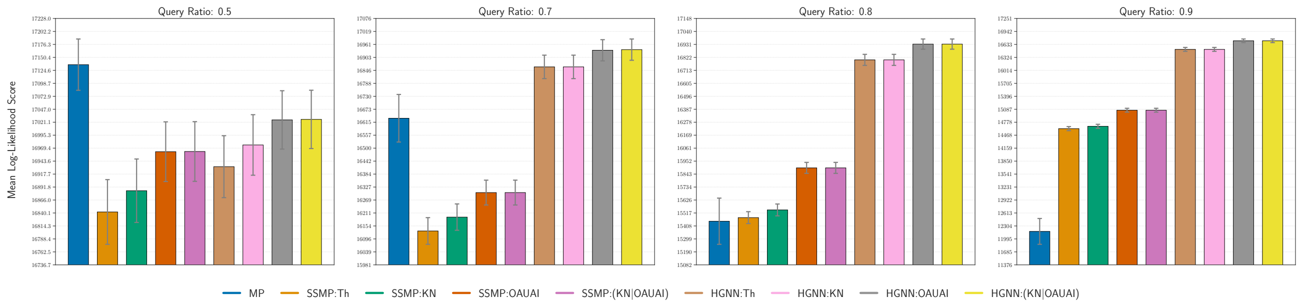


Figure 34: Evaluating Pairwise PGMs: Log-Likelihood Scores on grid40x40.f15. Higher Scores Reflect Superior Model Performance.

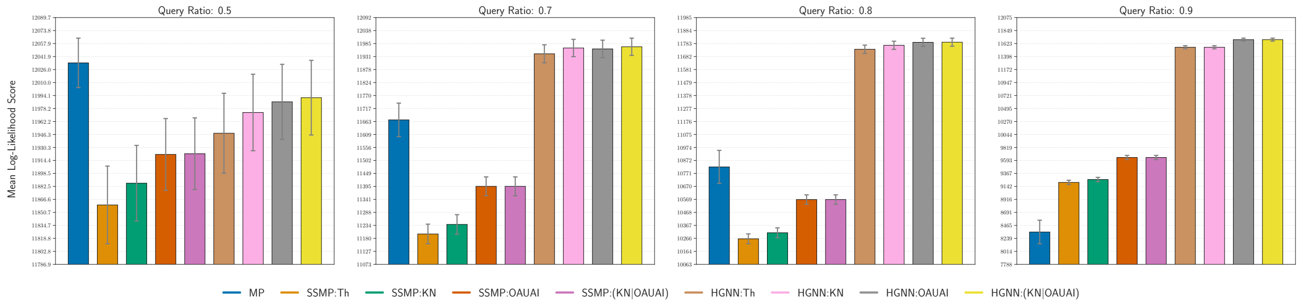


Figure 35: Evaluating Pairwise PGMs: Log-Likelihood Scores on grid40x40.f10.wrap. Higher Scores Reflect Superior Model Performance.

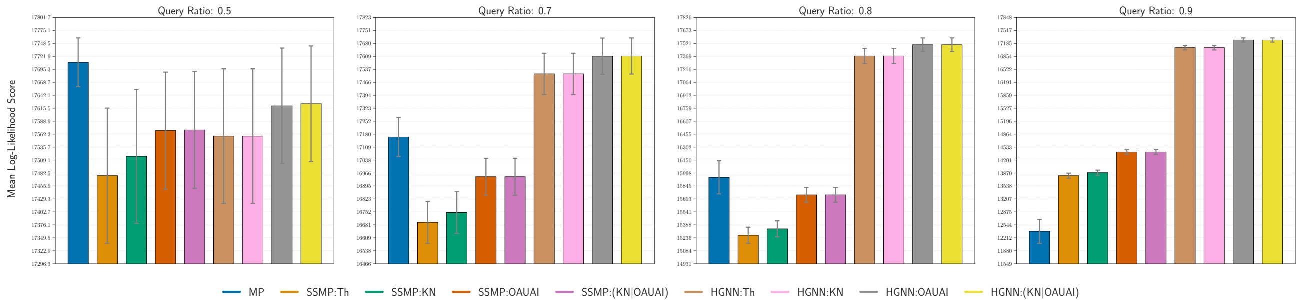


Figure 36: Evaluating Pairwise PGMs: Log-Likelihood Scores on grid40x40.f15.wrap. Higher Scores Reflect Superior Model Performance.