A Neural Network Approach for Efficiently Answering Most Probable Explanation Queries in Probabilistic Models

Shivvrat Arya[1] **Tahrima Rahman**[1] **Vibhav Gogate**[1]

[1]Department of Computer Science, The University of Texas at Dallas, Richardson, Texas, USA

Abstract

We propose a novel neural network-based approach to efficiently answer arbitrary Most Probable Explanation (MPE) queries in large probabilistic models, such as Bayesian and Markov networks, probabilistic circuits, and neural auto-regressive models. These MPE queries are not restricted by predefined partitions of variables into evidence and non-evidence groups. Our key idea is to distill all MPE queries into a neural network, eliminating the need for time-consuming inference algorithms on the probabilistic model itself. We enhance this method by incorporating inference-time optimization with a self-supervised loss to iteratively improve the solutions. Additionally, we use a teacher-student framework to provide a better initial network, reducing the number of necessary inference-time optimization steps. The teacher network, optimized with a self-supervised loss function, seeks the exact MPE solution, while the student network learns from the teacher's near-optimal outputs via supervised loss. We demonstrate the practicality, efficacy and scalability of our approach across various datasets and probabilistic models.

1 INTRODUCTION

Probabilistic models (PMs) such as Probabilistic Circuits (PCs), Bayesian Networks (BNs), Markov Networks (MNs), and Neural Autoregressive Models (NAMs) are widely used for modeling large, multi-dimensional probability distributions. However, solving NP-hard inference tasks like finding the Most Probable Explanation (MPE) is challenging and time-consuming with exact inference techniques [35, 36]. Exact solvers are often impractical due to their slow speed, and approximate solvers usually lack accuracy, particularly in autoregressive models that rely on slow hill-climbing or beam search methods.

Arya et al. [4] proposed using neural networks (NNs) to solve the MPE task in PCs, inspired by learning to optimize literature [12, 15, 29, 41, 54]. Their method involves training a NN to map evidence variables to query variables, using either supervised or self-supervised learning techniques.

We address a more general version of the MPE task, where there is no predefined partition of variables into evidence and query sets, referred to as the **any-MPE** task. This task is complex due to the exponential increase in input configurations and variable divisions. Our method applies to a broad class of PMs, including BNs, MNs, and NAMs, unlike Arya et al.'s method, which is limited to PCs.

Our novel approach uses a NN to solve the any-MPE task in various probabilistic models, advancing in three key areas:

1. Efficient MPE Inference via Encoding Scheme and Loss Function: We introduce a new encoding scheme tailored to the input probabilistic model, delineating NN input and output nodes, and propose a tractable self-supervised loss function for efficient training.

2. Inference Time Optimization with ITSELF: Our new inference technique, ITSELF, iteratively refines the MPE solution during inference using gradient descent and our self-supervised loss, enabling continuous improvement without labeled data.

3. Two-Phase Pre-training with Teacher-Student Architecture: We propose a two-phase pre-training strategy to address self-supervised learning and ITSELF challenges. The teacher network overfits the training data using ITSELF, while the student network learns from the teacher's outputs using supervised loss functions, providing a robust starting point and reducing optimization steps.

Our experimental results demonstrate that our method surpasses existing approaches in both accuracy and speed across various probabilistic models, including PCs, BNs, MNs, and NAMs.

Accepted for the 7th Workshop on Tractable Probabilistic Modeling at UAI (TPM 2024).

2 BACKGROUND

We use binary variables with values from $\{0, 1\}$. An uppercase letter denotes a random variable (e.g., X), and its assigned value is denoted by lowercase letter (e.g., x). A set of random variables is denoted by a bold uppercase letter (e.g., $\mathbf{X}$), and an assignment to all variables in the set by the corresponding bold lowercase letter (e.g., $\mathbf{x}$).

We use the term probabilistic models (PMs) to refer to models where computing the likelihood of an assignment to all variables can be done in polynomial time. This includes Bayesian and Markov networks (probabilistic graphical models or PGMs) [26], smooth and decomposable probabilistic circuits (PCs) [8], and neural autoregressive models (NAMs) like NADE [49] and MADE [17].

We aim to solve the most probable explanation (MPE) task in PMs: finding the most likely assignment to all unobserved variables given observations. Formally, let $\mathcal{M}$ denote a PM defined over variables $\mathbf{X}$, representing the distribution $\mathrm{p}_{\mathcal{M}}(\mathbf{x})$. Variables $\mathbf{X}$ are categorized into evidence $\mathbf{E} \subseteq \mathbf{X}$ and query $\mathbf{Q} \subseteq \mathbf{X}$, with $\mathbf{E} \cap \mathbf{Q} = \emptyset$ and $\mathbf{E} \cup \mathbf{Q} = \mathbf{X}$. Given an assignment $\mathbf{e}$ to $\mathbf{E}$, the MPE task is:

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

$\texttt{MPE}(\mathbf{Q}, \mathbf{e})$ is NP-hard, even to approximate [9, 11, 43].

3 A SELF-SUPERVISED NEURAL APPROXIMATOR FOR ANY-MPE

In this section, we develop a neural network (NN) approach for solving the *any-MPE* task. Given a PM, we create an input encoding (Section 3.1) that sets the number and values of the NN's input nodes for the MPE query. We also design an output encoding to determine the number of output nodes and recover the MPE solution from the outputs. For training, we introduce a tractable self-supervised loss function (Section 3.2), whose global minima align with the MPE solutions, enabling efficient learning from unlabeled data.

3.1 AN ENCODING FOR ANY-MPE INSTANCES

Since NNs require fixed-sized inputs and outputs, we introduce input and output encodings that generate fixed-length input and output vectors for each PM from a given MPE problem instance $\texttt{MPE}(\mathbf{Q}, \mathbf{e})$. To encode the input, for each variable $X_i \in \mathbf{X}$, we associate two input nodes in the NN, denoted by $\hat{X}_i$ and $\bar{X}_i$. Thus for a PM having n (namely, $|\mathbf{X}| = n$) variables, the corresponding NN has $2n$ input nodes. Given a query MPE$(\mathbf{Q}, \mathbf{e})$, we set the values of the input nodes as follows: (1) If $X_i \in \mathbf{E}$ and $X_i = 0$ is in $\mathbf{e}$, then we set $\hat{X}_i = 0$ and $\bar{X}_i = 1$; (2) If $X_i \in \mathbf{E}$ and $X_i = 1$ is in $\mathbf{e}$, then we set $\hat{X}_i = 1$ and $\bar{X}_i = 0$; and (3) If $X_i \in \mathbf{Q}$ then we set $\hat{X}_i = 0$ and $\bar{X}_i = 0$. It is easy to see that the input encoding described above yields an *injective* mapping between the set of all possible MPE queries over the given PM and the set $\{0, 1\}^{2n}$. This means that each unique MPE query $(\mathbf{Q}, \mathbf{e})$ will yield a unique 0-1 input vector of size $2n$.

The neural network's output consists of n nodes with sigmoid activation, each linked to a variable $X_i \in \mathbf{X}$. The MPE solution can be constructed by appropriately thresholding the output nodes corresponding to the query variables.

3.2 SELF-SUPERVISED LOSS FUNCTION

Given an MPE query, the NN's output is continuous, lying in the range $[0, 1]$. Let $\mathbf{q}^c \in [0, 1]^{|\mathbf{Q}|}$ be the predicted MPE assignment. We want to minimize $-\log \mathrm{p}_{\mathcal{M}}(\mathbf{q}, \mathbf{e})$, but $\mathrm{p}_{\mathcal{M}}(\mathbf{q}^c, \mathbf{e})$ is undefined for continuous outputs.

To address this, we use the property that for BNs, MNs, PCs, and NAMs, $\ell(\mathbf{q}, \mathbf{e}) = -\log \mathrm{p}_{\mathcal{M}}(\mathbf{e}, \mathbf{q})$ is either a multilinear polynomial or a neural network, and can be computed in linear time. Specifically, we define a continuous loss function $\ell^c(\mathbf{q}^c, \mathbf{e})$ that coincides with ℓ on $\{0, 1\}^n$. For PGMs and PCs, ℓ^c is obtained by substituting each discrete variable q_i with the corresponding continuous variable $q_i^c \in [0, 1]$. For NAMs, we perform a similar substitution.

Following Arya et al. [4], we improve the loss function using an entropy-based penalty ℓ_E, governed by $\alpha > 0$.

$$\ell_E(\mathbf{q}^c, \alpha) = -\alpha \sum_{j=1}^{|\mathbf{Q}|} \left[q_j^c \log(q_j^c) + (1 - q_j^c) \log(1 - q_j^c) \right]$$

This penalty encourages discrete solutions by preferring q_j^c values close to 0 or 1. Using the theory of Lagrange multipliers, we can show that for any $\alpha > 0$, the use of the entropy penalty yields a tighter lower bound:

Proposition 1.

$$\min_{\mathbf{q}^c \in [0,1]^n} \ell^c(\mathbf{q}^c, \mathbf{e}) \leq \min_{\mathbf{q}^c \in [0,1]^n} \ell^c(\mathbf{q}^c, \mathbf{e}) + \ell_E(\mathbf{q}^c, \alpha) \leq \min_{\mathbf{q} \in \{0,1\}^n} \ell(\mathbf{q}, \mathbf{e})$$

Using the Loss Function: We train a neural network with $2n$ input nodes and n output nodes using the self-supervised loss function $\ell^c(\mathbf{q}^c, \mathbf{e}) + \ell_E(\mathbf{q}^c, \alpha)$. This trained NN can answer any MPE query over the PM. Training data consists of evidence assignments generated by sampling full assignments to all variables, selecting evidence variables randomly, and projecting full assignments onto evidence. This approach eliminates the need for an external MPE solver, using gradient-based training for supervision.

3.3 INFERENCE-TIME NEURAL OPTIMIZATION USING SELF-SUPERVISED LOSS

Assuming the NN is over-parameterized, using the self-supervised loss and repeatedly running gradient updates on

the NN for a given dataset will lead to convergence near the global minimum of the loss function [2, 13, 45, 55]. Thus, the NN will find near-optimal MPE assignments for each training example. This gradient update strategy can also be applied during inference to iteratively improve the MPE solution, maximizing the benefits of self-supervision.

More specifically, at test time, given a test dataset, we initialize the NN either randomly or using a pre-trained model, then iteratively run gradient updates until convergence. The gradient is computed with respect to the self-supervised loss function $\ell^c(\mathbf{q}^c, \mathbf{e}) + \ell_E(\mathbf{q}^c, \alpha)$. We call this algorithm ITSELF (Inference Time Optimization using SELF-Supervised Loss). ITSELF's performance typically improves with each iteration until the loss converges.

ITSELF is related to test-time training approaches in deep learning [1, 10, 19, 30–32, 39, 48, 50, 56]. Unlike these methods, our self-supervised loss's global minima correspond to MPE solutions if the penalty α is sufficiently large.

Algorithm 1 GUided Iterative Dual LEarning with Self-supervised Teacher ($\mathcal{GUIDE}$)

Input: Training data $\mathcal{D}$, teacher $\mathcal{T}$ and student $\mathcal{S}$
Output: Trained student network $\mathcal{S}$
Required: Database DB which stores the best assignment and loss value for each $\mathbf{e} \in \mathcal{D}$
Initialize: Randomly initialize $\mathcal{T}$, $\mathcal{S}$, and DB
repeat
    Sample a mini-batch $\mathcal{D}'$ from $\mathcal{D}$
    Update the parameters of $\mathcal{T}$ using ITSELF and $\mathcal{D}'$
    for each example $\mathbf{e}_i$ in $\mathcal{D}'$ **do**
        Make a forward-pass over $\mathcal{T}$ to yield $\mathbf{q}_i$ for $\mathbf{e}_i$
        Update the entry in DB for $\mathbf{e}_i$ with $\mathbf{q}_i$ if it has a lower loss value than the current entry
    end for
    Update the parameters of $\mathcal{S}$ using the mini-batch $\mathcal{D}'$ and labels from DB and a supervised loss
    $\mathcal{T} \leftarrow \mathcal{S}$ ▷ Initialize $\mathcal{T}$ with $\mathcal{S}$ for the next iteration
until Convergence or max iterations

4 SUPERVISED KNOWLEDGE TRANSFER FROM ITSELF

A drawback of our self-supervised loss function is that it is non-convex function of the NN outputs. As a result, it has larger number of local minima compared to supervised loss functions, but also exponential number of global minima, as an MPE problem can have multiple optimal solutions [37]. Consequently, optimizing and regularizing with self-supervised loss is more challenging for large datasets.

Moreover, our experiments indicate that large datasets require large, over-parameterized NNs to achieve near-optimal MPE solutions. However, with limited training data and sufficiently over-parameterized NNs, our preliminary experiments and prior studies [3, 6, 23, 27, 28] suggest a higher likelihood of approaching global optima. Specifically, with a reasonably sized NN and a small dataset, the algorithm tends to yield near-optimal MPE solutions. A further challenge is that even for small datasets, achieving convergence from random initialization requires many gradient descent iterations, making the training process inefficient.

4.1 TEACHER-STUDENT STRATEGY

To address challenges such as using small datasets with ITSELF, designing better initialization, and training with non-convex loss functions, we propose a two-network teacher-student strategy [7, 16, 20–22, 24, 38, 46, 51–53]. Both networks have the same structure and are trained via mini-batch gradient updates. The teacher network is overfitted to the mini-batch using our self-supervised loss via ITSELF, ensuring it finds near-optimal MPE assignments for all examples in the mini-batch. The student network is then trained with a supervised loss function like binary cross entropy, learning from the teacher's outputs as soft labels. This strategy mitigates optimization difficulties of the non-convex self-supervised loss, allowing the student network to achieve faster convergence and better generalization with a smaller model size. It reduces the need for severe over-parameterization and extensive training iterations for the teacher network, improving initialization for ITSELF.

4.2 TRAINING PROCEDURE

Our proposed training procedure, which we call $\mathcal{GUIDE}$, is detailed in Algorithm 1. The algorithm trains a two-network system comprising a teacher network ($\mathcal{T}$) and a student network ($\mathcal{S}$) with the same structure. The goal is to train the student network using a combination of self-supervised and supervised learning strategies. The algorithm takes as input the training data $\mathcal{D}$, along with the teacher and student networks, $\mathcal{T}$ and $\mathcal{S}$, respectively and outputs a trained network $\mathcal{S}$. A database (DB) is utilized to store the best MPE assignment and corresponding loss value for each example in $\mathcal{D}$. The parameters of $\mathcal{T}$ and $\mathcal{S}$, and the entries in DB, are randomly initialized at the start.

In each epoch, a mini-batch $\mathcal{D}'$ is sampled. $\mathcal{T}$ is updated with ITSELF on $\mathcal{D}'$. For each $\mathbf{e}_i$ in $\mathcal{D}'$, a forward-pass over $\mathcal{T}$ yields $\mathbf{q}_i$. DB is updated if $\mathbf{q}_i$ improves the loss as compared to the current entry for $\mathbf{e}_i$ in DB. Then, $\mathcal{S}$ is updated with $\mathcal{D}'$ and DB labels using a supervised loss. Finally, $\mathcal{T}$ is reinitialized with $\mathcal{S}$ for the next epoch (addressing the initialization issue associated with ITSELF).

Thus, at a high level, Algorithm 1 leverages the strengths of both self-supervised and supervised learning to improve training efficiency and reduce the model complexity, yielding a student network $\mathcal{S}$. Moreover, at test time, the student network can serve as an initialization for ITSELF.

	MAX	ML	Seq	SSMP	$\mathcal{GUIDE}$	SSMP ITSELF	$\mathcal{GUIDE}$ ITSELF
MAX	0	79	94	81	71	39	12
ML	41	0	43	55	49	15	5
Seq	26	64	0	44	43	16	7
SSMP	39	65	76	0	24	4	0
$\mathcal{GUIDE}$	49	71	77	71	0	8	0
SSMP ITSELF	81	105	104	104	99	0	7
$\mathcal{GUIDE}$ ITSELF	108	115	113	107	107	94	0

(a) PCs: Initial Solutions

	MAX	ML	Seq	SSMP	$\mathcal{GUIDE}$	SSMP ITSELF	$\mathcal{GUIDE}$ ITSELF
MAX	0	65	81	66	62	46	21
ML	35	0	49	41	27	22	4
Seq	21	45	0	29	27	15	4
SSMP	36	55	71	0	23	17	1
$\mathcal{GUIDE}$	39	71	73	49	0	24	0
SSMP ITSELF	53	73	82	70	61	0	12
$\mathcal{GUIDE}$ ITSELF	79	88	94	91	89	72	0

(b) PCs: Hill-Climbing

Figure 1: Contingency Tables

5 EXPERIMENTS

For lack of space, we only present experiments for PCs.

Datasets and Models: We used twenty binary datasets extensively used in tractable probabilistic models literature [5, 18, 34, 49]—referred to as TPM datasets—for evaluating PCs. To train Sum Product Networks (SPNs), our choice of Probabilistic Circuits (PC)s, we employed the DeeProb-kit library [33]. The query ratio (qr) is defined as the fraction of variables in the query set. For each PM, we varied qr with values from the set $\{0.1, 0.3, 0.5, 0.7, 0.8, 0.9\}$.

Baseline Methods: We employed three polynomial-time baseline methods from the literature as initial benchmarks [40, 44]. **MAX Approximation (MAX)** [44] transforms sum nodes into max nodes and performs two passes over the PC to compute an MPE solution. **Maximum Likelihood Approximation (ML)** [40] computes the marginal distribution $p_{\mathcal{M}}(Q_i|\mathbf{e})$ for each variable $Q_i \in \mathbf{Q}$, and sets Q_i to its most likely value. **Sequential Approximation (Seq)** [40] iteratively assigns query variables according to an order o. At each step j, it selects the j-th query variable Q_j in o and assigns to it a value q_j such that $\mathrm{p}_{\mathcal{M}}(q_j|\mathbf{e}, \mathbf{y})$ is maximized, where $\mathbf{y}$ is an assignment of values to all query variables from 1 to $j-1$. Our study further assessed the impact of initializing stochastic hill climbing searches using solutions from all baseline approaches and our proposed methods for MPE inference, conducting 60-second searches for each MPE problem (see Park and Darwiche [40]).

Arya et al. [4] introduced Self-Supervised learning based MMAP solver for PCs (SSMP), which trains a NN to handle queries on a fixed partition of variables on PCs. We proposed an extension of this approach for solving the any-MPE task. We use it as another neural baseline. We used **log-likelihood (LL) scores**, $\ln p_{\mathcal{M}}(\mathbf{e}, \mathbf{q})$, and inference times for a given evidence $\mathbf{e}$ and query output $\mathbf{q}$ to compare the algorithms.

Results: For each PM and query ratio, we implemented two neural network training protocols: SSMP and $\mathcal{GUIDE}$, subjecting each model to 20 training epochs as in Arya et al. [4]. Both protocols employed two distinct inference strategies: (1) a single forward pass through the network to estimate query variable values (Arya et al. [4]), (2) our novel test-time optimization-based approach Inference Time Self Supervised Training (**ITSELF**), where we undertook 100 optimization iterations or terminated earlier upon achieving convergence. We standardized network architectures across all experiments (see supplement for more details).

We compare methods—including three polynomial-time baselines, neural network-based SSMP, and our **ITSELF** and $\mathcal{GUIDE}$ methods—on 20 TPM datasets as shown in the contingency table in figure 1a (detailed results in the supplementary materials). We generated 120 test datasets for the *Most Probable Explanation* (MPE) task using 20 PCs across 6 query ratios (qr). Each cell (i, j) in the table represents how often (out of 120) the method in row i outperformed the method in column j based on average log-likelihood scores. Any difference between 120 and the combined frequencies of cells (i, j) and (j, i) indicates cases where the compared methods achieved similar scores.

The contingency table for PC shows that the **ITSELF** methods outperform polynomial-time and traditional neural baselines. Specifically, $\mathcal{GUIDE}$ + **ITSELF** is superior to all the other methods in almost two-thirds of the 120 cases, while SSMP + **ITSELF** is better than both SSMP and $\mathcal{GUIDE}$ without **ITSELF**. However, the polynomial-time baseline MAX is better than both SSMP and $\mathcal{GUIDE}$ (note that these methods were used in Arya et al. [4]), highlighting **ITSELF**'s significant role in boosting model performance for the complex *any-MPE* task.

Further analysis in Figure 1b compares our proposed methods with various baselines for initializing Hill Climbing Search. The goal is to assess if ITSELF and $\mathcal{GUIDE}$ improve *anytime methods* to surpass other heuristic initialization techniques. We observe that methods utilizing **ITSELF** are superior for initializing local-search methods.

Our experiments answer the following two questions *affirmatively*. First, because $\mathcal{GUIDE}$ is superior to SSMP, we conclude that teacher-student training is better than training a single NN with self-supervised loss. Second, Inference-time optimization via ITSELF is superior to performing one pass over the NN, and yields superior initialization for subsequent discrete optimization methods like hill climbing.

Summary. We introduced novel methods for answering MPE queries in probabilistic models using self-supervised loss functions for tractable loss and gradient computations during neural network training. We proposed **ITSELF**, an inference time optimization technique that iteratively improves MPE solutions via gradient updates, and $\mathcal{GUIDE}$, a dual-network strategy combining supervised and unsupervised training for better initialization. Tested on various benchmarks, our method outperformed polytime baselines and other neural methods, and enhanced the effectiveness of stochastic hill climbing search strategies.

ACKNOWLEDGEMENTS

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

References

[1] Ferran Alet, Maria Bauza, Kenji Kawaguchi, Nurullah Giray Kuru, Tomás Lozano-Pérez, and Leslie Kaelbling. Tailoring: Encoding inductive biases by optimizing unsupervised objectives at prediction time. In *Advances in Neural Information Processing Systems*, volume 34, pages 29206–29217. Curran Associates, Inc., 2021.

[2] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 242–252. PMLR, 09–15 Jun 2019.

[3] Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. 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, 2019.

[4] Shivvrat Arya, Tahrima Rahman, and Vibhav Gogate. Neural Network Approximators for Marginal MAP in Probabilistic Circuits. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(10):10918–10926, March 2024. ISSN 2374-3468. doi: 10.1609/aaai.v38i10.28966.

[5] Jessa Bekker, Jesse Davis, Arthur Choi, Adnan Darwiche, and Guy Van den Broeck. Tractable learning for complex probability queries. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[6] Lénaïc Chizat and Francis R. Bach. 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, 2018.

[7] Jang Hyun Cho and Bharath Hariharan. On the Efficacy of Knowledge Distillation. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4793–4801, October 2019. doi: 10.1109/ICCV.2019.00489.

[8] YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. oct 2020.

[9] Gregory F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, March 1990. ISSN 00043702. doi: 10.1016/0004-3702(90)90060-D.

[10] Mohammad Zalbagi Darestani, Jiayu Liu, and Reinhard Heckel. Test-time training can close the natural distribution shift performance gap in deep learning based compressed sensing. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 4754–4776. PMLR, 17–23 Jul 2022.

[11] Cassio P de Campos. New Complexity Results for MAP in Bayesian Networks. *IJCAI International Joint Conference on Artificial Intelligence*, pages 2100–2106, 01 2011. doi: 10.5591/978-1-57735-516-8/IJCAI11-351.

[12] Priya L. Donti, David Rolnick, and J. Zico Kolter. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, October 2020.

[13] S. Du, Xiyu Zhai, B. Póczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. *International Conference on Learning Representations*, 2018.

[14] G. Elidan and A. Globerson. *The 2010 UAI Approximate Inference Challenge*. 2010.

[15] Ferdinando Fioretto, Terrence W. K. Mak, and Pascal Van Hentenryck. Predicting AC Optimal Power Flows: Combining Deep Learning and Lagrangian Dual Methods. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):630–637, April 2020. ISSN 2374-3468. doi: 10.1609/aaai.v34i01.5403.

[16] Tommaso Furlanello, Zachary Chase Lipton, M. Tschannen, L. Itti, and Anima Anandkumar. Born Again Neural Networks. In *International Conference on Machine Learning*, May 2018.

[17] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 881–889, Lille, France, 07–09 Jul 2015. PMLR.

[18] Jan Van Haaren and Jesse Davis. Markov Network Structure Learning: A Randomized Feature Generation Approach. *Proceedings of the AAAI Conference on Artificial Intelligence*, 26(1):1148–1154, 2012. ISSN 2374-3468. doi: 10.1609/aaai.v26i1.8315.

[19] Moritz Hardt and Yu Sun. Test-time training on nearest neighbors for large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

[20] Byeongho Heo, Jeesoo Kim, Sangdoo Yun, Hyojin Park, Nojun Kwak, and Jin Young Choi. A Comprehensive Overhaul of Feature Distillation. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1921–1930, October 2019. doi: 10.1109/ICCV.2019.00201.

[21] Byeongho Heo, Minsik Lee, Sangdoo Yun, and Jin Young Choi. Knowledge Transfer via Distillation of Activation Boundaries Formed by Hidden Neurons. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3779–3787, July 2019. doi: 10.1609/aaai.v33i01.33013779.

[22] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[23] Arthur Jacot, Clément Hongler, and Franck Gabriel. 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, 2018.

[24] Jangho Kim, Seonguk Park, and Nojun Kwak. Paraphrasing Complex Network: Network Compression via Factor Transfer. *ArXiv*, February 2018.

[25] Diederik P. Kingma and Jimmy Ba. 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*, 2015.

[26] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[27] Jaehoon Lee, S. Schoenholz, Jeffrey Pennington, Ben Adlam, Lechao Xiao, Roman Novak, and Jascha Narain Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. *Neural Information Processing Systems*, 2020.

[28] Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent. *Journal of Statistical Mechanics: Theory and Experiment*, 2020(12):124002, December 2020. ISSN 1742-5468. doi: 10.1088/1742-5468/abc62b.

[29] Ke Li and Jitendra Malik. Learning to optimize. *International Conference on Learning Representations*, 2016.

[30] Yushu Li, Xun Xu, Yongyi Su, and Kui Jia. On the Robustness of Open-World Test-Time Training: Self-Training with Dynamic Prototype Expansion. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11802–11812, Paris, France, October 2023. IEEE. ISBN 9798350307184. doi: 10.1109/ICCV51070.2023.01087.

[31] Huan Liu, Zijun Wu, Liangyan Li, Sadaf Salehkalaibar, Jun Chen, and Keyan Wang. Towards Multi-domain Single Image Dehazing via Test-time Training. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5821–5830, New Orleans, LA, USA, June 2022. IEEE. ISBN 978-1-66546-946-3. doi: 10.1109/CVPR52688.2022.00574.

[32] Yuejiang Liu, Parth Kothari, Bastien van Delft, Baptiste Bellot-Gurlet, Taylor Mordan, and Alexandre Alahi. TTT++: When Does Self-Supervised Test-Time Training Fail or Thrive? In *Advances in Neural Information Processing Systems*, volume 34, pages 21808–21820. Curran Associates, Inc., 2021.

[33] Lorenzo Loconte and Gennaro Gala. DeeProbkit: a python library for deep probabilistic modelling, 2022. URL `https://github.com/deeprob-org/deeprob-kit`.

[34] Daniel Lowd and Jesse Davis. Learning Markov Network Structure with Decision Trees. In *2010 IEEE International Conference on Data Mining*, pages 334–343. IEEE, 2010. ISBN 978-1-4244-9131-5. doi: 10.1109/ICDM.2010.128.

[35] Radu Marinescu. Daoopt: Distributed and/or optimization-uai'10 inference competition. 2010. URL `https://github.com/lotten/daoopt`.

[36] Radu Marinescu and Rina Dechter. AND/OR Branch-and-Bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16-17):

1457–1491, November 2009. ISSN 00043702. doi: 10.1016/j.artint.2009.07.003.

[37] Radu Marinescu and Rina Dechter. Counting the optimal solutions in graphical models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/fc2e6a440b94f64831840137698021e1-Paper.pdf.

[38] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved Knowledge Distillation via Teacher Assistant. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5191–5198, April 2020. doi: 10.1609/aaai.v34i04.5963.

[39] David Osowiechi, G. A. V. Hakim, Mehrdad Noori, Milad Cheraghalikhani, Ismail Ben Ayed, and Christian Desrosiers. Tttflow: Unsupervised test-time training with normalizing flow. *IEEE Workshop/Winter Conference on Applications of Computer Vision*, 2022. doi: 10.48550/arXiv.2210.11389.

[40] James D. Park and Adnan Darwiche. Complexity results and approximation strategies for map explanations. *J. Artif. Int. Res.*, 21(1):101–133, feb 2004. ISSN 1076-9757.

[41] Seonho Park and Pascal Van Hentenryck. Self-Supervised Primal-Dual Learning for Constrained Optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(4):4052–4060, June 2023. ISSN 2374-3468. doi: 10.1609/aaai.v37i4.25520.

[42] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[43] Robert Peharz. *Foundations of Sum-Product Networks for Probabilistic Modeling*. PhD thesis, PhD thesis, Medical University of Graz, 2015.

[44] Hoifung Poon and Pedro Domingos. Sum-Product Networks: A New Deep Architecture. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pages 337–346. AUAI Press, 2011.

[45] Tiancheng Qin, S. Rasoul Etesami, and Cesar A Uribe. Faster convergence of local SGD for over-parameterized models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

[46] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, C. Gatta, and Yoshua Bengio. FitNets: Hints for Thin Deep Nets. *CoRR*, December 2014.

[47] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56): 1929–1958, 2014. ISSN 1533-7928.

[48] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-Time Training with Self-Supervision for Generalization under Distribution Shifts. In *Proceedings of the 37th International Conference on Machine Learning*, pages 9229–9248. PMLR, November 2020.

[49] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 17(205):1–37, 2016.

[50] Dequan Wang, Evan Shelhamer, Shaoteng Liu, B. Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. *International Conference on Learning Representations*, 2021.

[51] Chenglin Yang, Lingxi Xie, Chi Su, and Alan L. Yuille. Snapshot Distillation: Teacher-Student Optimization in One Generation. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2854–2863, June 2019. doi: 10.1109/CVPR.2019.00297.

[52] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A Gift from Knowledge Distillation: Fast Optimization, Network Minimization and Transfer Learning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7130–7138, July 2017. doi: 10.1109/CVPR.2017.754.

[53] Sergey Zagoruyko and N. Komodakis. Paying More Attention to Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer. *ArXiv*, November 2016.

[54] Ahmed S. Zamzam and Kyri Baker. 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, November 2020. doi: 10.1109/SmartGridComm47815.2020.9303008.

[55] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[56] Wentao Zhu, Yufang Huang, Daguang Xu, Zhen Qian, Wei Fan, and Xiaohui Xie. Test-time training for deformable multi-scale image registration. *IEEE International Conference on Robotics and Automation*, 2021.

A Neural Network Approach for Efficiently Answering Most Probable Explanation Queries in Probabilistic Models (Supplementary Material)

Shivvrat Arya[1] **Tahrima Rahman**[1] **Vibhav Gogate**[1]

[1]Department of Computer Science, The University of Texas at Dallas, Richardson, Texas, USA

A EXPERIMENTS

This section evaluates the ITSELF method (see section 3.3), the $\mathcal{GUIDE}$ teacher-student training method (see section 4) and the method that uses only self-supervised training, which we call SSMP (see section 3.2). We benchmark these against various baselines, including neural network-based and traditional polynomial-time algorithms that directly operate on the probabilistic model. We begin by detailing our experimental framework, including competing methods, evaluation metrics, neural network architectures, and datasets.

A.1 DATASETS AND GRAPHICAL MODELS

We used twenty binary datasets extensively used in tractable probabilistic models literature [5, 18, 34, 49]—referred to as TPM datasets—for evaluating PCs and Neural Auto-Regressive Model (NAM). For the purpose of evaluating Probabilistic Graphical Models (PGM)s, we utilized high treewidth models from previous UAI inference competitions [14].

To train Sum Product Networks (SPNs), our choice of PCs, we employed the DeeProb-kit library [33], with SPN sizes ranging from 46 to 9666 nodes. For NAM, Masked Autoencoder for Distribution Estimation (MADE) models were trained using PyTorch as described in [17]. In the case of Markov Networks (MNs), a specific category of PGMs, we utilize Gibbs Sampling, generating 8000, 1000, and 1000 examples for the training, testing, and validation sets, respectively. The query ratio (qr) is defined as the fraction of variables in the query set. For each PM, we varied qr with values from the set $\{0.1, 0.3, 0.5, 0.7, 0.8, 0.9\}$.

A.2 BASELINE METHODS AND EVALUATION CRITERIA

PCs - We employed three polynomial-time baseline methods from the PC and PGM literature as initial benchmarks [40, 44]. **MAX** [44] transforms sum nodes into max nodes. During the upward pass, max nodes output the highest weighted value from their children. The downward pass, starting from the root, selects the child with the highest value at each max node and includes all children of product nodes. **ML** [40] computes the marginal distribution $p_{\mathcal{M}}(Q_i|\mathbf{e})$ for each variable $Q_i \in \mathbf{Q}$, setting Q_i to its most likely value. **Seq** [40] iteratively assigns query variables according to an order o. At each step j, it selects the j-th query variable Q_j in o and assigns to it a value q_j such that $\mathrm{p}_{\mathcal{M}}(q_j|\mathbf{e}, \mathbf{y})$ is maximized, where $\mathbf{y}$ is an assignment of values to all query variables from 1 to $j-1$. Our study further assessed the impact of initializing stochastic hill climbing searches using solutions from all baseline approaches and our proposed methods for MPE inference, conducting 60-second searches for each MPE problem in our experiments, as detailed in Park and Darwiche [40].

NAMs - As a baseline, we used the stochastic hill-climbing search (HC). Similarly to the PC procedure, for each test example, we conducted a 60-second hill-climbing search, initializing query variables randomly and setting evidence variables based on the example.

PGMs - As a baseline we utilize the AND/OR Branch-and-Bound (AOBB)[36], employing the implementation described in Marinescu [35]. Given that AOBB is an anytime scheme, for each test example, we designated a 60-second time limit for

Accepted for the 7th Workshop on Tractable Probabilistic Modeling at UAI (TPM 2024).

	MAX	ML	Seq	SSMP	$\mathcal{GUIDE}$	SSMP ITSELF	$\mathcal{GUIDE}$ ITSELF
MAX	0	79	94	81	71	39	12
ML	41	0	43	55	49	15	5
Seq	26	64	0	44	43	16	7
SSMP	39	65	76	0	24	4	0
$\mathcal{GUIDE}$	49	71	77	71	0	8	0
SSMP ITSELF	81	105	104	104	99	0	7
$\mathcal{GUIDE}$ ITSELF	108	115	113	107	107	94	0

(a) PCs: Initial Solutions

	MAX	ML	Seq	SSMP	$\mathcal{GUIDE}$	SSMP ITSELF	$\mathcal{GUIDE}$ ITSELF
MAX	0	65	81	66	62	46	21
ML	35	0	49	41	27	22	4
Seq	21	45	0	29	27	15	4
SSMP	36	55	71	0	23	17	1
$\mathcal{GUIDE}$	39	71	73	49	0	24	0
SSMP ITSELF	53	73	82	70	61	0	12
$\mathcal{GUIDE}$ ITSELF	79	88	94	91	89	72	0

(b) PCs: Hill-Climbing

	MADE	SSMP	$\mathcal{GUIDE}$	SSMP ITSELF	$\mathcal{GUIDE}$ ITSELF
MADE	0	46	38	36	12
SSMP	34	0	7	13	4
$\mathcal{GUIDE}$	42	66	0	34	16
SSMP ITSELF	44	61	40	0	6
$\mathcal{GUIDE}$ ITSELF	68	72	60	70	0

(c) NAMs

	AOBB	SSMP	$\mathcal{GUIDE}$	SSMP ITSELF	$\mathcal{GUIDE}$ ITSELF
AOBB	0	7	6	6	5
SSMP	9	0	3	4	0
$\mathcal{GUIDE}$	9	12	0	15	3
SSMP ITSELF	10	12	0	0	3
$\mathcal{GUIDE}$ ITSELF	11	16	12	13	0

(d) PGMs

Figure 2: Contingency Tables

the inference process.

Neural Baselines - Arya et al. [4] introduced SSMP, which trains a neural network to handle queries on a fixed partition of variables on PCs. We proposed an extension of this approach for solving the any-MPE task in PMs (see section 3.2), where a single neural network is trained to answer any-MPE query. We use it as another neural baseline.

Evaluation Criteria - Competing approaches were compared using 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 $\mathbf{q}$.

A.3 NEURAL NETWORK-BASED APPROACHES

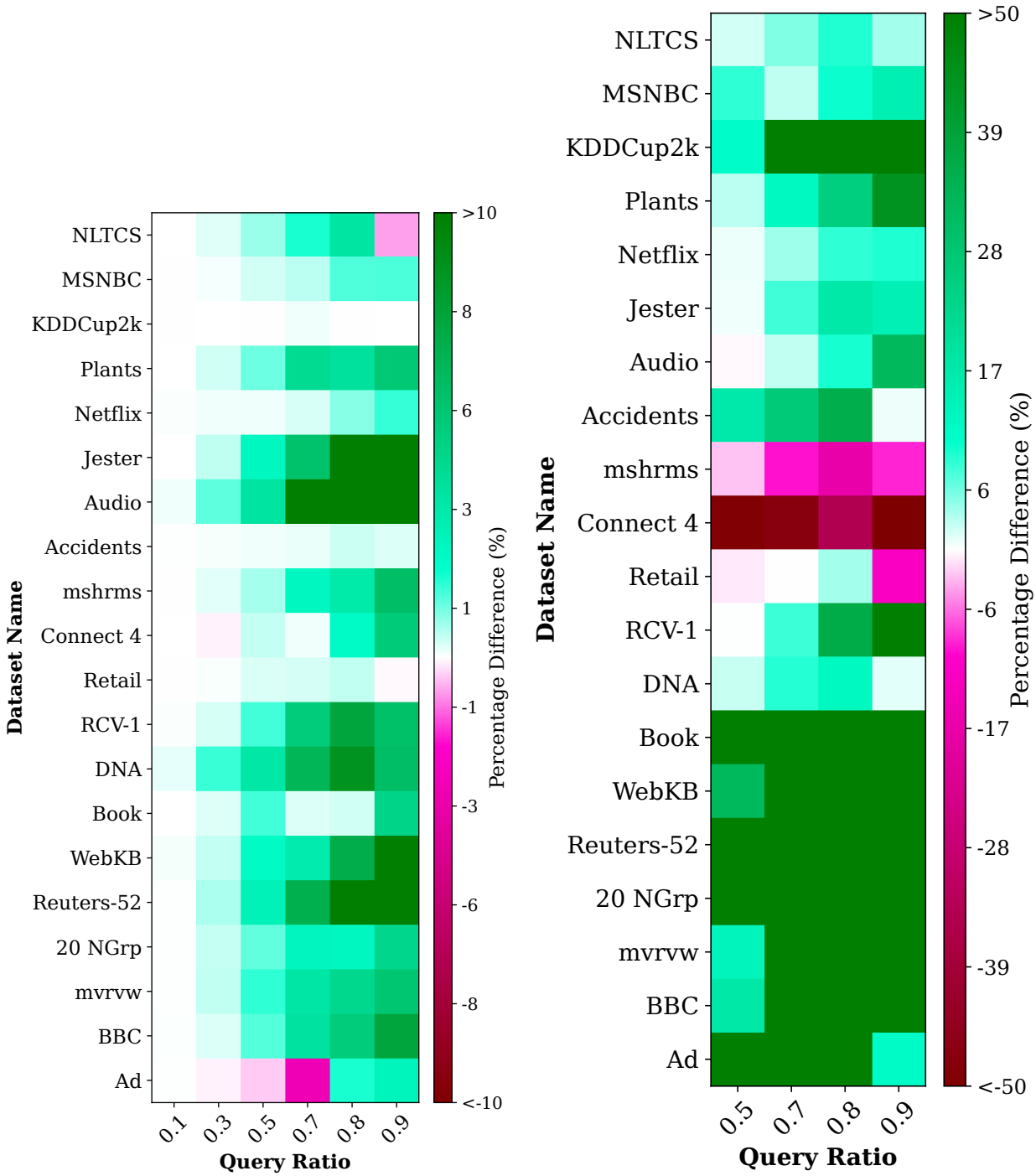
For each PM and query ratio, we implemented two neural network training protocols: SSMP and $\mathcal{GUIDE}$. We subjected each model to 20 training epochs, adhering to the established procedures for SSMP as delineated by Arya et al. [4]. Both protocols employed two distinct inference strategies, thus forming four neural-based variants. The first strategy consisted of a single forward pass through the network to estimate query variable values, as specified by Arya et al. [4]. The second strategy utilized our novel test-time optimization-based **ITSELF** approach for inference. For **ITSELF**, we undertook 100 optimization iterations or terminated earlier upon achieving loss convergence, applicable to both PCs and PGMs. In cases involving NAMs, the optimization iterations extended to 1,000, or concluded upon convergence.

We standardized network architectures for PMs across all experiments. For PCs, we used a fully connected Neural Networks (NN) with three hidden layers (128, 256, 512 nodes). For NAMs and PGMs, a single hidden layer of 512 nodes was employed. All hidden layers featured ReLU activation, while the output layers used sigmoid functions with dropout for regularization [47]. Optimization was performed using the Adam optimizer [25], and models were implemented in PyTorch [42] on an NVIDIA A40 GPU.

Results for PCs: We compare methods—including three polynomial-time baselines, neural network-based SSMP, and our **ITSELF** and $\mathcal{GUIDE}$ methods—on 20 TPM datasets as shown in the contingency table in figure 2a (detailed results in the supplementary materials). We generated 120 test datasets for the MPE task using 20 PCs across 6 query ratios (qr). Each cell (i, j) in the table represents how often (out of 120) the method in row i outperformed the method in column j based on average log-likelihood scores. Any difference between 120 and the combined frequencies of cells (i, j) and (j, i) indicates cases where the compared methods achieved similar scores.

The contingency table for PC shows that the **ITSELF** methods outperform polynomial-time and traditional neural baselines. Specifically, $\mathcal{GUIDE}$ + **ITSELF** is superior to all the other methods in almost two-thirds of the 120 cases, while SSMP + **ITSELF** is better than both SSMP and $\mathcal{GUIDE}$ without **ITSELF**. However, the polynomial-time baseline MAX is better than both SSMP and $\mathcal{GUIDE}$ (note that these methods were used in Arya et al. [4]), highlighting **ITSELF**'s significant role in boosting model performance for the complex *any-MPE* task.

We compare MAX and $\mathcal{GUIDE}$ + **ITSELF** using a heatmap in Figure 3a. The y-axis presents datasets by variable count and the x-axis by query ratio. Each cell displays the percentage difference in mean LL scores between the methods, calculated as $\%\text{Diff.} = 100 \times (ll_{nn} - ll_{max})/|ll_{max}|$. From the heatmap in Figure 3a for PCs, we observe that $\mathcal{GUIDE}$ + **ITSELF** exhibit performance comparable to the MAX approximation when the query set size is small. As the problem complexity increases with an increase in query set size, our new method consistently outperforms MAX across all datasets, except for NLTCS and Tretail. Even when $\mathcal{GUIDE}$ + **ITSELF** underperforms compared to MAX (noted in 12 instances out of 120) the performance gap is very small, evidenced by sparse red cells in the heatmaps.



(a) PC: $\mathcal{GUIDE}$ + **ITSELF** vs. MAX

(b) NAM: $\mathcal{GUIDE}$ + **ITSELF** and HC

Figure 3: Heatmaps showing LL Percentage Differences: Top—PC; Bottom—MADE.

Further analysis in Figure 2b compares the performance of our proposed methods with various baselines as initialization strategies for Hill Climbing Search. The goal is to assess whether ITSELF and $\mathcal{GUIDE}$ enhance *anytime methods* to outperform other heuristic initialization techniques, with methods utilizing **ITSELF** proving superior in initializing local search-based algorithms.

Results for NAMs: Our evaluation, detailed in the contingency table in Figure 2c, examines the performance of several methods for NAM, including HC and two neural network methods, SSMP and $\mathcal{GUIDE}$, each with two inference schemes. We tested four neural-based schemes on 20 TPM datasets, generating 80 test datasets for the MPE task using 20 MADEs across four query ratios (qr).

Similar to the results observed with PC, the $\mathcal{GUIDE}$ + **ITSELF** method exhibits superior performance over the baselines and other neural inference schemes. HC outperforms SSMP, while $\mathcal{GUIDE}$ and SSMP + **ITSELF** are superior to HC.

The heatmaps in Figure 3b highlight the superior performance of $\mathcal{GUIDE}$ + **ITSELF** for NAMs, particularly in larger datasets where it outperforms the HC baseline by over 50% in most cases, as indicated by the dark green cells. The integration of $\mathcal{GUIDE}$-based learning with **ITSELF**-based inference consistently outperforms the baseline across most scenarios, with exceptions only in the Mushrooms, Connect 4, and Retail datasets. Thus, the $\mathcal{GUIDE}$ + **ITSELF** approach significantly enhances MPE query answering in NAM models.

Results for PGMs: In our evaluation, as detailed in the contingency table in 2d, we assessed the efficacy of various methods for PGMs, including AOBB and four neural-network-based methods, across four high-treewidth networks (details of these networks are provided in the supplement). To accomplish this, we constructed 16 test datasets for the MPE task by employing four PGMs across four query ratios (qr).

Similar to the outcomes observed with previous PMs, the methods employing **ITSELF** for inference consistently demonstrate enhanced performance compared to the baseline methods AOBB and SSMP in most scenarios. Both $\mathcal{GUIDE}$ and SSMP outperform AOBB in at least 50 percent of the tests.

Is Teacher Student Training Better than A Single Network Trained with Self-Supervised Loss? (SSMP versus $\mathcal{GUIDE}$): In our comparative analysis between SSMP and $\mathcal{GUIDE}$ across different models, we aim to evaluate the performance of $\mathcal{GUIDE}$ against the traditional neural network training methods used in SSMP. Using traditional inference schemes (i.e., one forward pass through the network), $\mathcal{GUIDE}$ consistently outperforms SSMP, demonstrating its superiority in 60% of scenarios for PCs, more than 80% for NAM models, and 75% for PGM models. When employing **ITSELF** for inference over the trained models, $\mathcal{GUIDE}$ continues to outperform SSMP, achieving better results in more than 75%, 85%, and 80% for PCs, NAMs, and PGMs, respectively.

Does Inference Time Optimization help? (One Pass versus Multiple Passes): Comparative analyses reveal that **ITSELF** consistently outperforms traditional single forward pass inference across various PMs. **ITSELF** with SSMP training outperforms the other methods in over 85% of PC cases, and more than 75% for NAM and PGM models. When trained using $\mathcal{GUIDE}$, **ITSELF** demonstrates even better results, achieving superior performance in nearly 90% of PC cases and 75% for both NAMs and PGMs. $\mathcal{GUIDE}$ with the **ITSELF** inference scheme emerges as the best method across all our experiments.

Finally, we present inference times in the supplement. For all the PMs, the neural based method that utilizes traditional inference are the quickest. For MADE, the method employing a $\mathcal{GUIDE}$-trained model with **ITSELF** is the second fastest. Similarly, in PGMs, the $\mathcal{GUIDE}$ + **ITSELF** method emerges as the third fastest, after SSMP + **ITSELF**; in PCs, MAX leads, slightly ahead of both $\mathcal{GUIDE}$ + **ITSELF** and SSMP + **ITSELF**, while ML and Seq record the longest inference times.

Summary: Our experiments demonstrate that $\mathcal{GUIDE}$ + **ITSELF** outperforms both polynomial-time and neural-based baselines across various PMs, as evidenced by higher log-likelihood scores. Specifically, **ITSELF** exceeds the capabilities of traditional forward pass inference in addressing the challenging task of answering *any-MPE* queries within a probabilistic model, highlighting the necessity of Inference Time Optimization. Additionally, the superiority of models trained with $\mathcal{GUIDE}$ over SSMP underscores the effectiveness of using a dual network approach that utilizes two loss functions to enhance the initial model quality and provide an optimal starting point for **ITSELF**.

B MORE DETAILS ON EXPERIMENTS

B.1 DATASETS AND MODELS

Table 1 summarizes the datasets and the probabilistic circuits trained on them. We use the same datasets for both PCs [8] and NAMs [17, 49]. The selection includes both smaller datasets, such as NLTCS and MSNBC, and larger datasets with over 1000 nodes. We utilize high treewidth grid Markov networks for PGM, specifically grid40x40.f2.wrap, grid40x40.f5.wrap, grid40x40.f10.wrap, and grid40x40.f15.wrap. Each model contains 4800 variables and 1600 factors.

Table 1: Summary of datasets used with their respective numbers of variables and nodes in probabilistic circuits.

Dataset	Number of Variables	Number of Nodes
NLTCS	16	125
MSNBC	17	46
KDDCup2k	64	274
Plants	69	3737
Audio	100	348
Jester	100	274
Netflix	100	400
Accidents	111	1178
Mushrooms	112	902
Connect 4	126	2128
Retail	135	359
RCV-1	150	519
DNA	180	1855
Book	500	1628
WebKB	839	3154
Reuters-52	889	7348
20 NewsGroup	910	2467
Movie reviews	1001	2567
BBC	1058	3399
Ad	1556	9666

B.2 HYPERPARAMETERS DETAILS

Our experimental framework was meticulously designed to ensure consistency and efficiency across all conducted experiments. For NAM's we used MADE, the model was trained with two hidden layers, 512 and 1024 units each, using hyper-parameters from [17].

For neural network models, the mini-batch size was set to 512 instances, and a learning rate decay strategy, reducing the rate by 0.9 upon loss plateauing, was implemented to improve training efficiency. Optimal hyperparameters were identified via extensive 5-fold cross-validation, as detailed in the main text.

In discrete loss scenarios, the hyperparameter α played a pivotal role. We undertook a methodical investigation to ascertain the optimal value of α, examining the range $0.001, 0.01, 0.1, 1, 10, 100, 1000$ for neural based models including **ITSELF** and $\mathcal{GUIDE}$. It is important to note that higher values of α constrain outputs to binary values more closely, aiding in achieving near-optimal results.

C A COMPARATIVE ANALYSIS OF PERFORMANCE OF ITSELF FOR DIFFERENT PRE-TRAINING METHODS

This section evaluates the performance of models initialized through various techniques—random initialization, SSMP, and $\mathcal{GUIDE}$. Each plot represents the loss for a distinct test example, with the x-axis denoting the number of **ITSELF** iterations and the y-axis showing the Negative Log Likelihood (NLL) scores. Lower NLL values signify better solutions. Through this empirical assessment, we compare the impact of different pre-training methods on model performance.

Figures 4 to 27 present the plots for NAMs. The plots for PCs are shown in Figures 28 to 66. Figures 67 to 77 illustrate the plots for PGMs. We selected the following datasets for PCs and NAMs: DNA, RCV-1, Reuters-52, Netflix, WebKB, Audio, Moviereview, and Jester. For PGMs, we used all the datasets presented in the main paper. Each plot consists of two sections. The left section presents the Negative Log-Likelihood Loss for 1000 iterations for all methods. The right section contains two sub-plots: the top sub-plot displays the zoomed-in losses for the first 200 iterations, while the bottom sub-plot shows the zoomed-in losses for the last 200 iterations.

We randomly initialize the parameters for the random model and perform 1000 iterations of **ITSELF** for inference. For the two pre-trained models (SSMP and $\mathcal{GUIDE}$), we update the top N layers, where N is the number of layers corresponding to that loss curve, and fix the remaining bottom layers. We extract features by passing the input through these fixed layers and then train the parameters of the top N layers. We again perform 1000 iterations of **ITSELF** for inference. For NAMs and PGMs, we use neural networks with up to one hidden layer, while for PCs, we employ models with up to three hidden layers.

From the plots for the three Probabilistic Modelss (PMs), we observe that models pre-trained using the proposed $\mathcal{GUIDE}$ training scheme generally have a better starting point for **ITSELF**, indicated by a lower loss, compared to all other models. Across a wide array of datasets, PGMs, and query percentages, the $\mathcal{GUIDE}$ method consistently converges to a lower or equivalent loss compared to other models. Remarkably, it sometimes achieves a loss value that is less than half of the nearest competing model. Furthermore, the losses for $\mathcal{GUIDE}$ are typically more stable than those of other initialization. In some scenarios, all models achieve a similar final loss, although models initialized with SSMP and those randomly initialized may experience oscillations in their loss values.

Models pre-trained using the traditional self-supervised loss (SSMP) typically have better or similar starting points than randomly initialized models. However, models pre-trained using the SSMP method might converge to a worse loss than their $\mathcal{GUIDE}$ pre-trained counterparts.

In most cases, convergence is rapid, even with a reduced learning rate of 10^{-4} compared to the experiments shown in the main paper. Most methods converge within 200 to 300 iterations, although some may still oscillate during the later iterations of **ITSELF**.

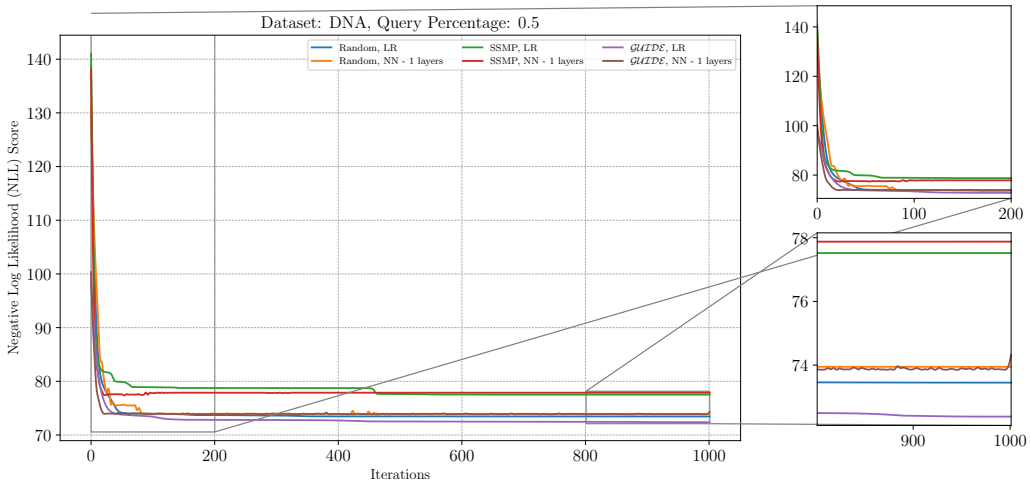


Figure 4: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the DNA Dataset at a Query Ratio of 0.5.

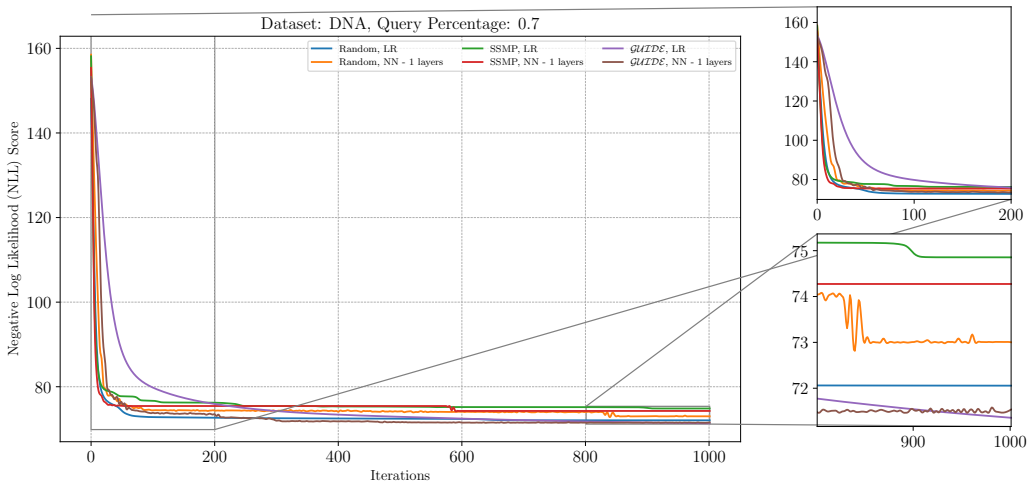


Figure 5: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the DNA Dataset at a Query Ratio of 0.7.

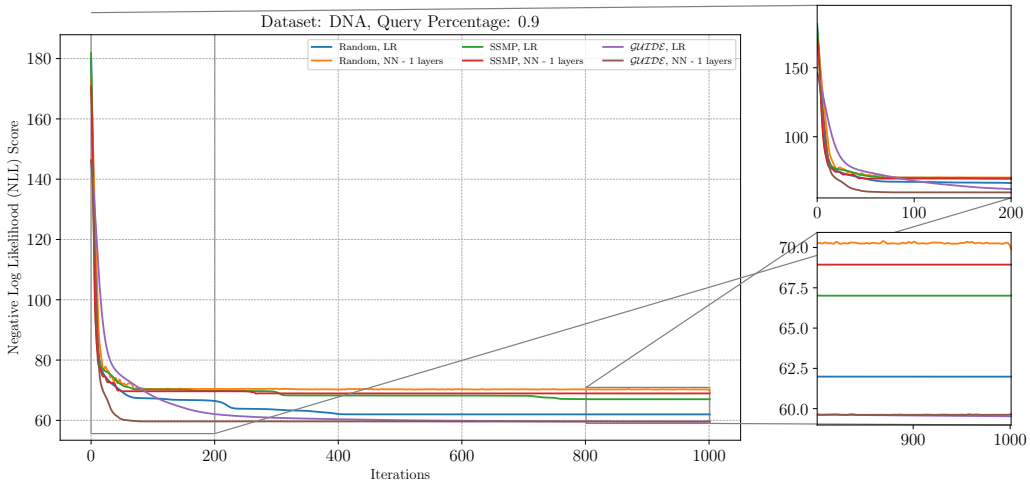


Figure 6: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the DNA Dataset at a Query Ratio of 0.9.

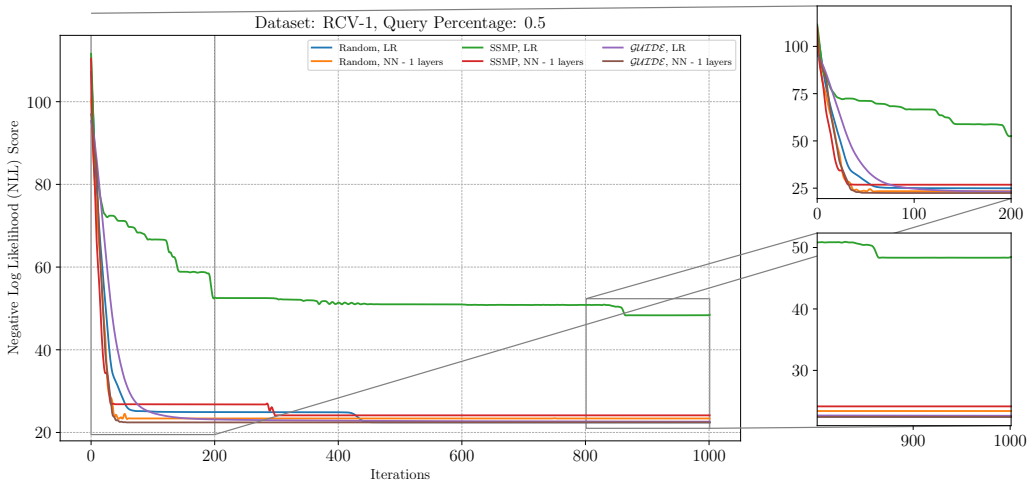


Figure 7: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the RCV-1 Dataset at a Query Ratio of 0.5.

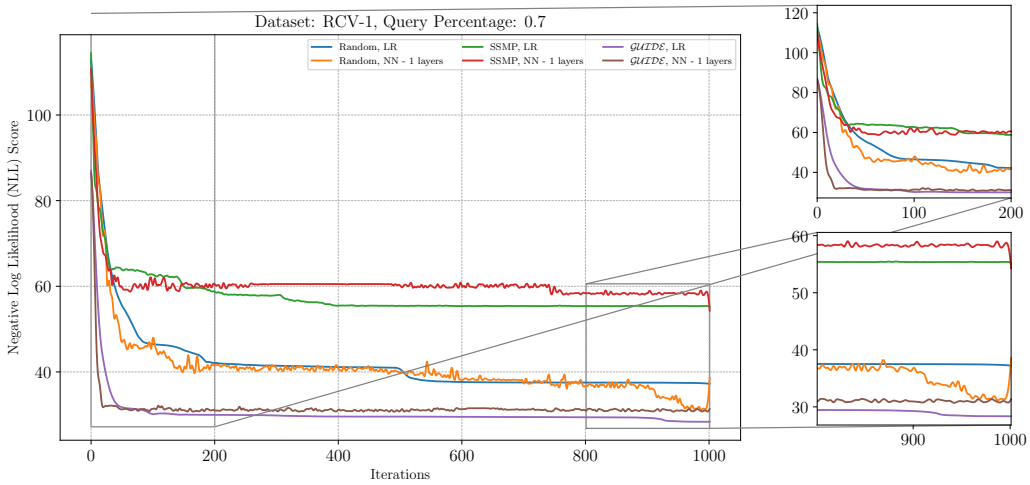


Figure 8: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the RCV-1 Dataset at a Query Ratio of 0.7.

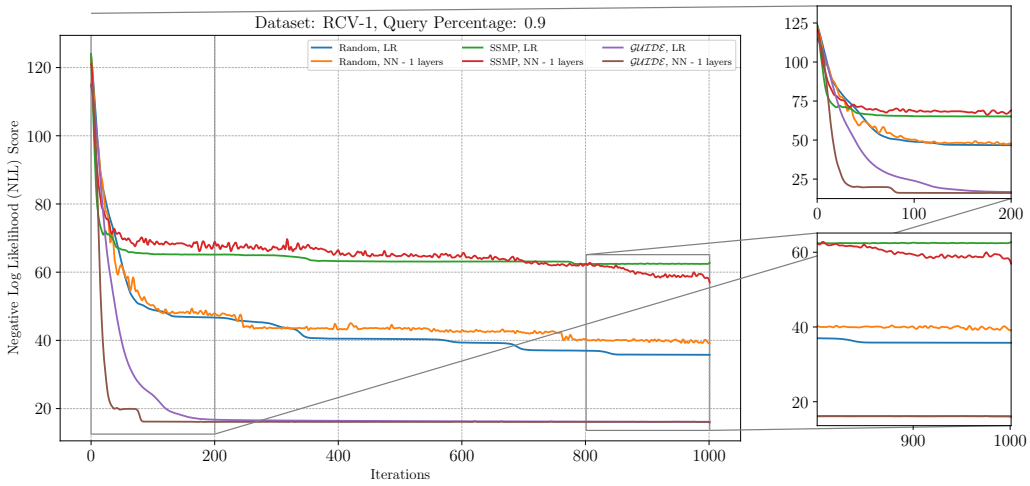


Figure 9: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the RCV-1 Dataset at a Query Ratio of 0.9.

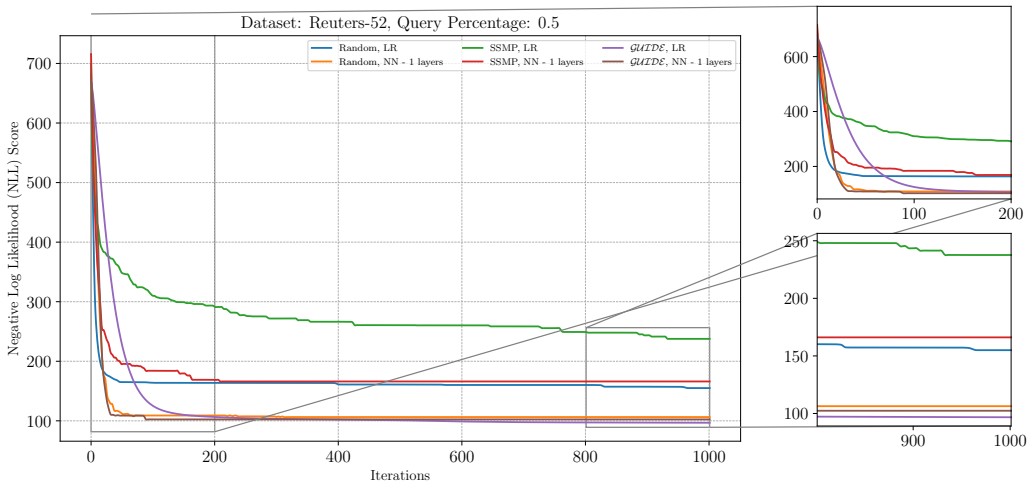


Figure 10: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the Reuters-52 Dataset at a Query Ratio of 0.5.

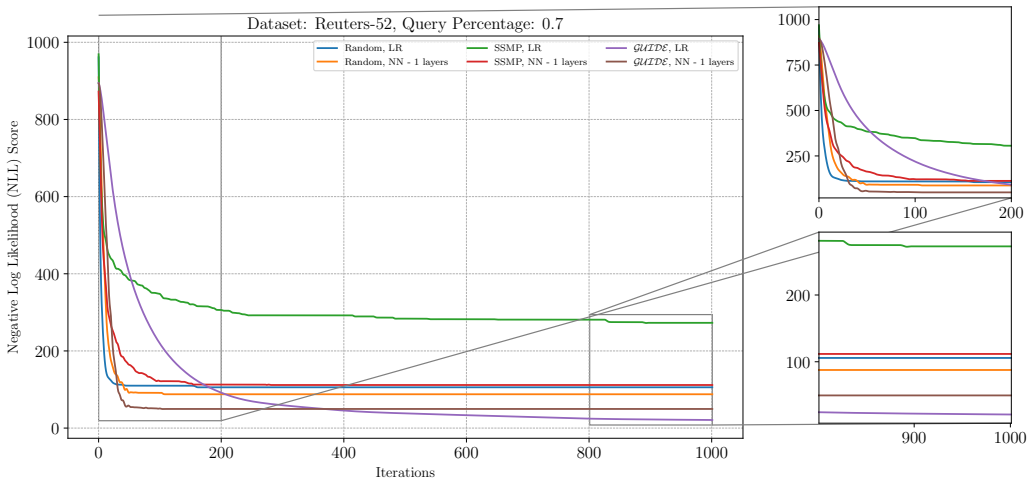


Figure 11: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the Reuters-52 Dataset at a Query Ratio of 0.7.

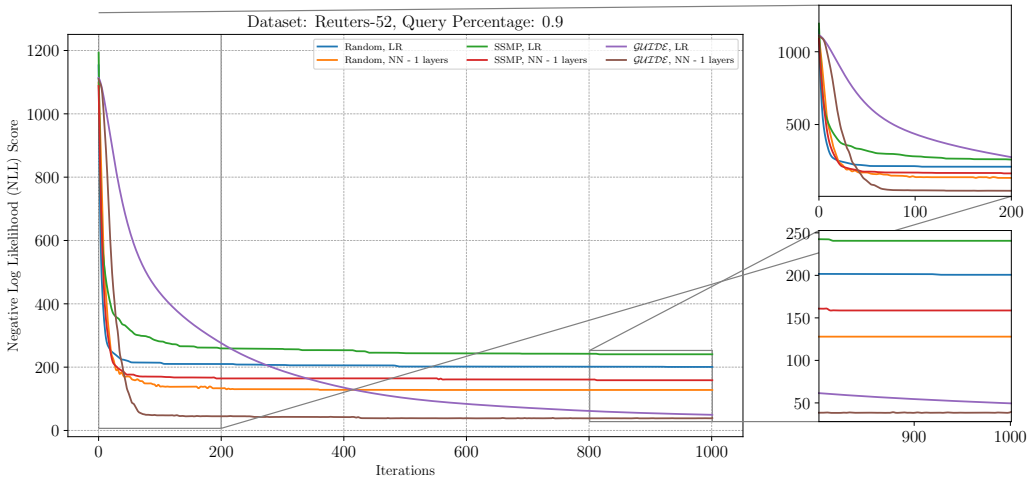


Figure 12: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the Reuters-52 Dataset at a Query Ratio of 0.9.

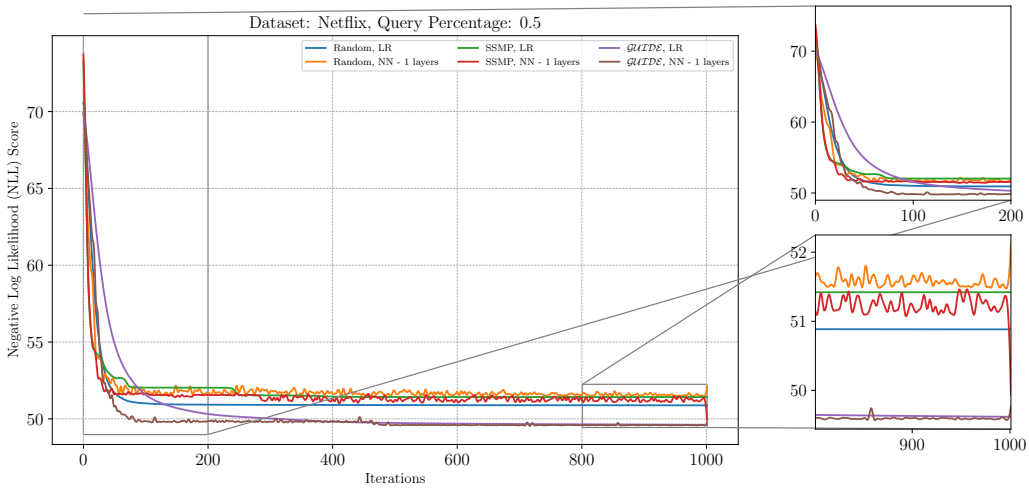


Figure 13: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the Netflix Dataset at a Query Ratio of 0.5.

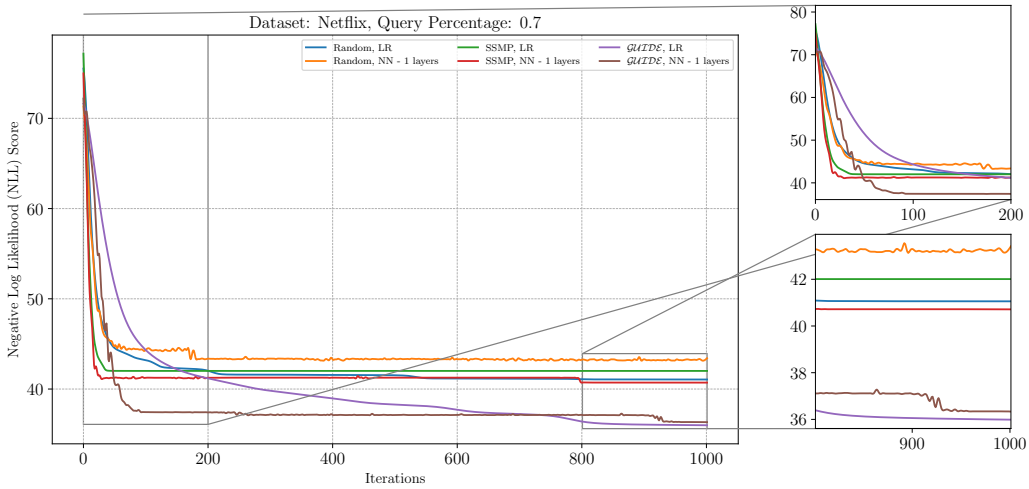


Figure 14: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the Netflix Dataset at a Query Ratio of 0.7.

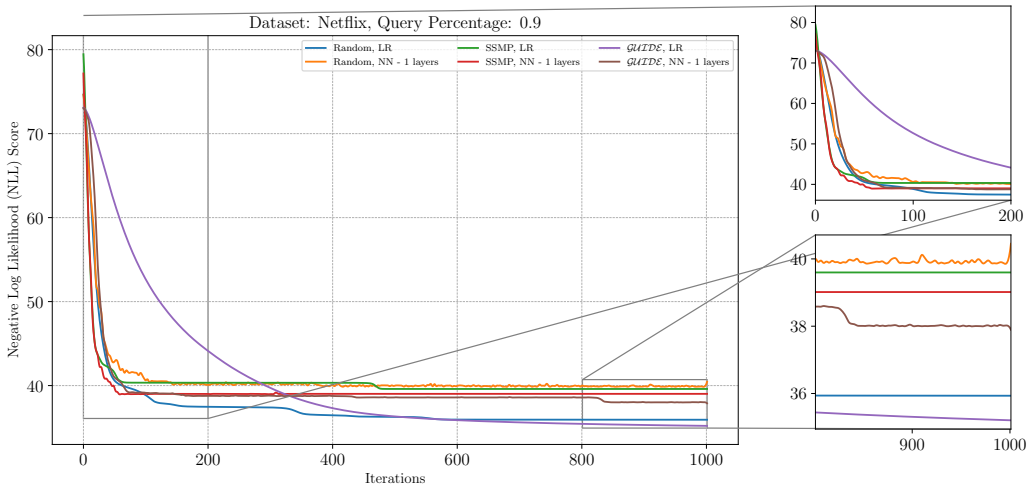


Figure 15: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the Netflix Dataset at a Query Ratio of 0.9.

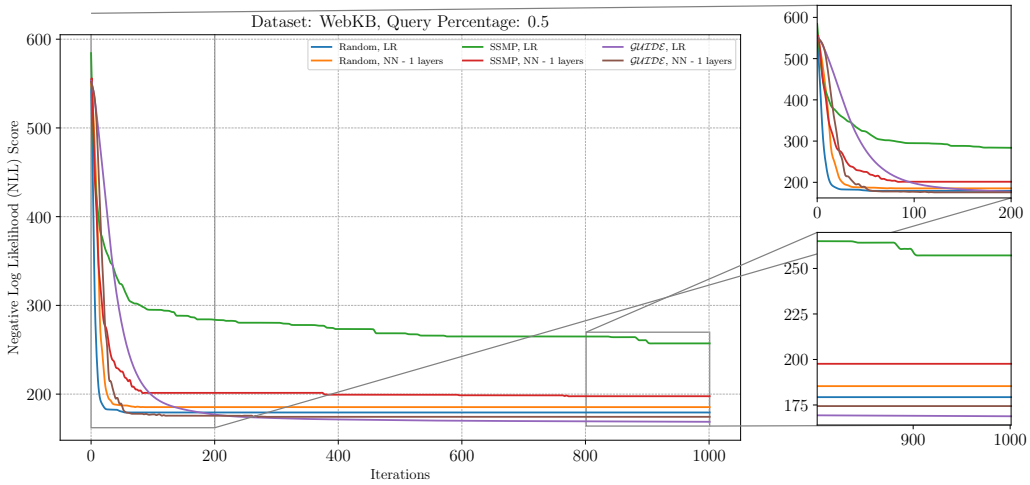


Figure 16: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the WebKB Dataset at a Query Ratio of 0.5.

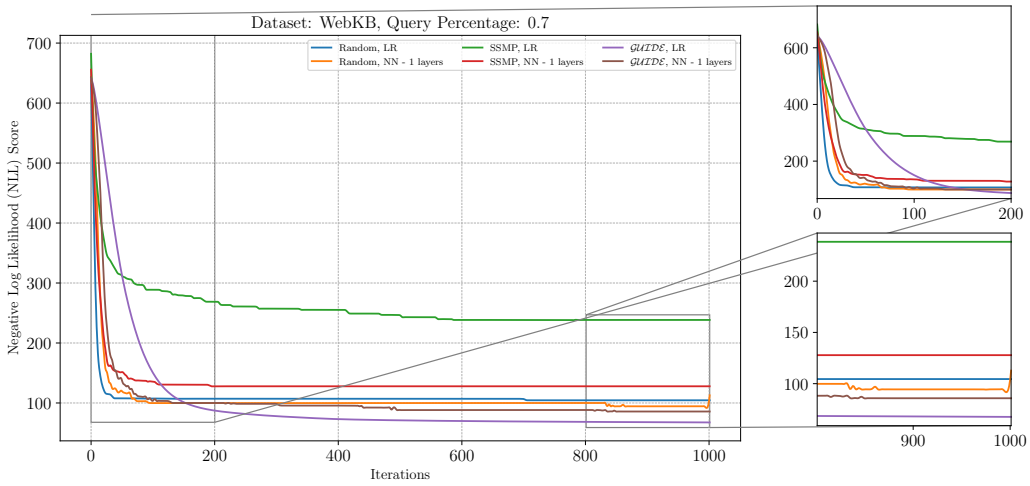


Figure 17: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the WebKB Dataset at a Query Ratio of 0.7.

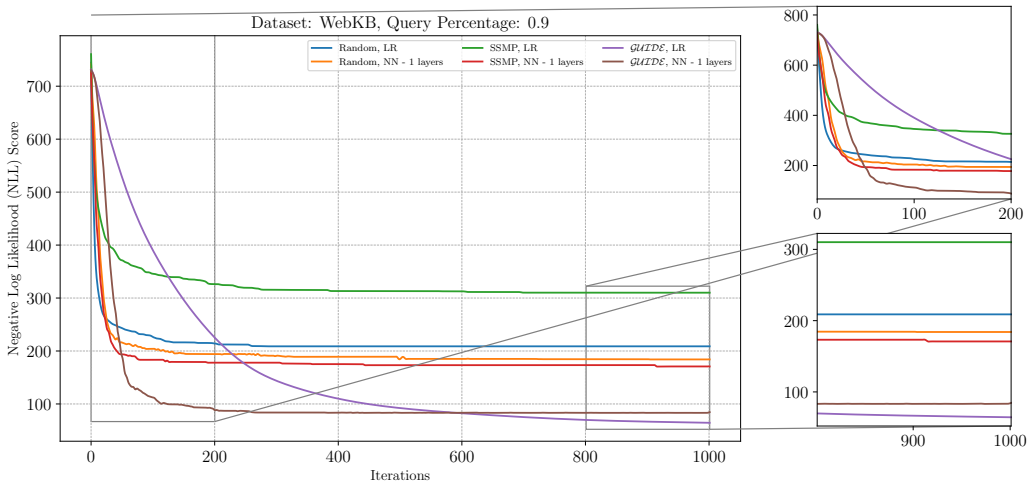


Figure 18: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the WebKB Dataset at a Query Ratio of 0.9.

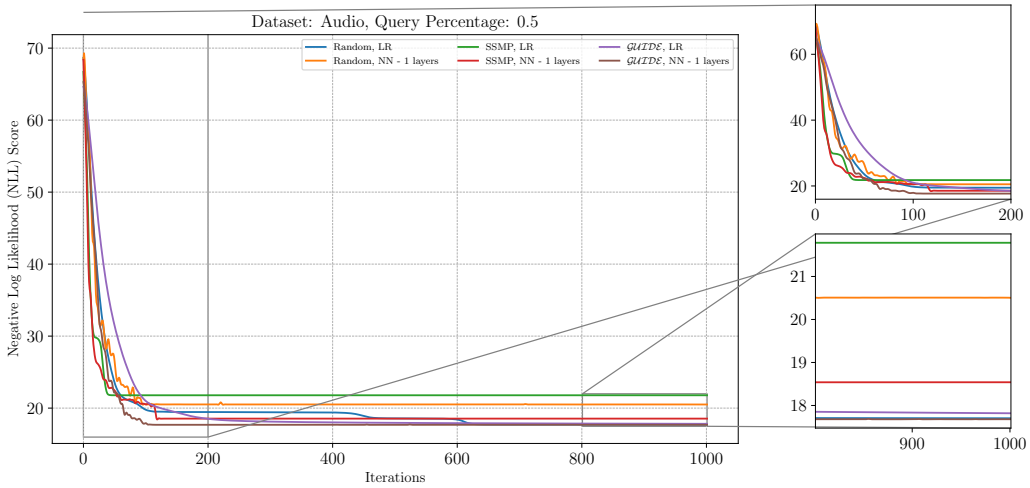


Figure 19: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the Audio Dataset at a Query Ratio of 0.5.

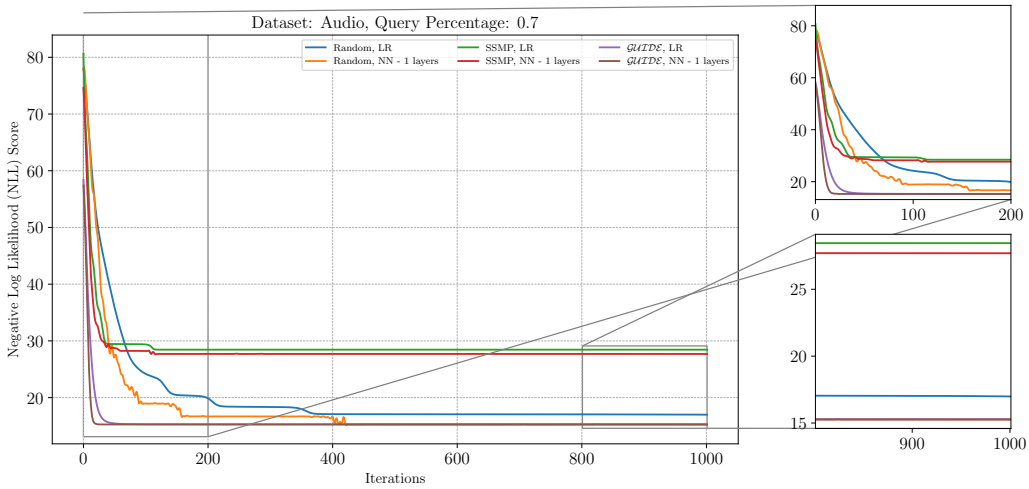


Figure 20: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the Audio Dataset at a Query Ratio of 0.7.

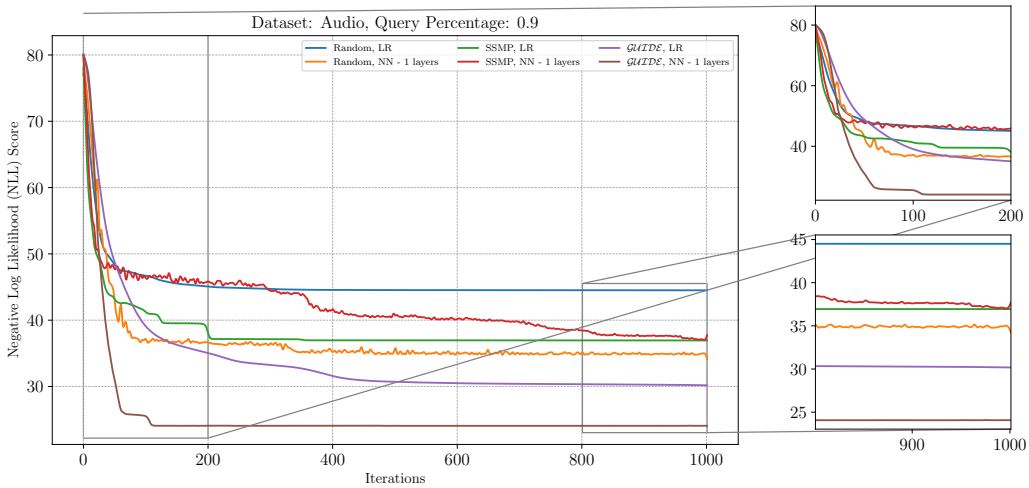


Figure 21: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the Audio Dataset at a Query Ratio of 0.9.

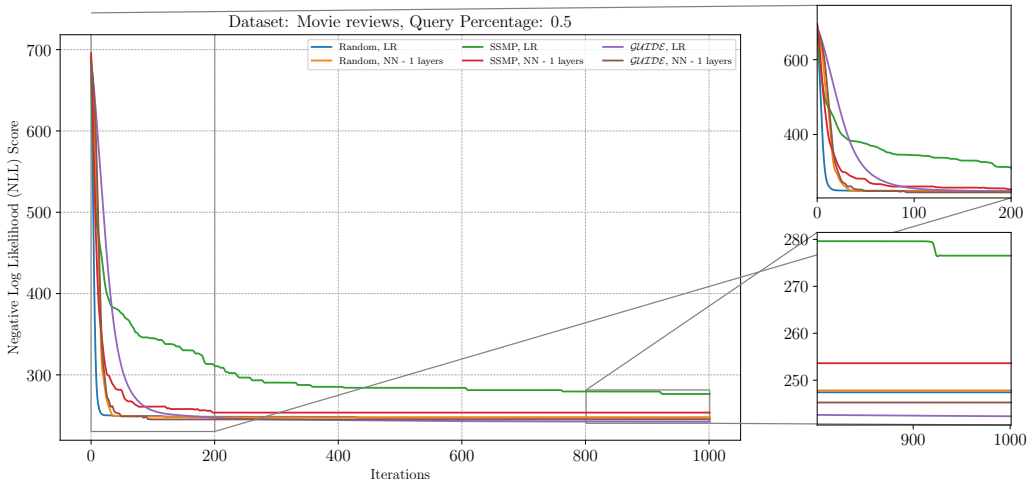


Figure 22: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the Movie reviews Dataset at a Query Ratio of 0.5.

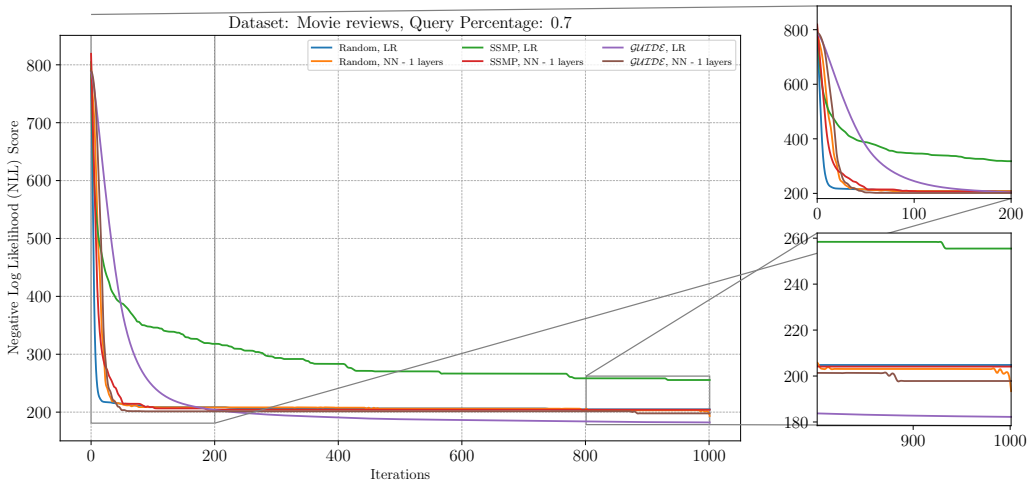


Figure 23: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the Movie reviews Dataset at a Query Ratio of 0.7.

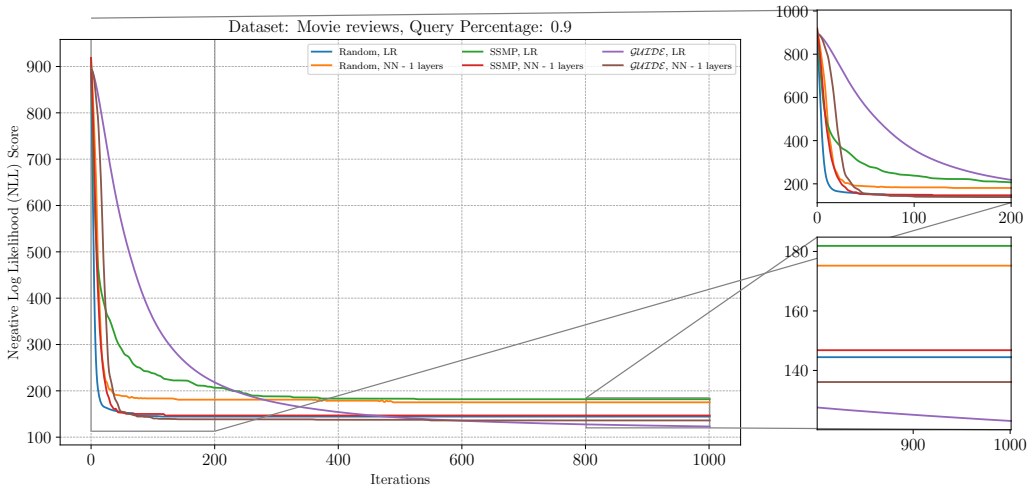


Figure 24: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the Movie reviews Dataset at a Query Ratio of 0.9.

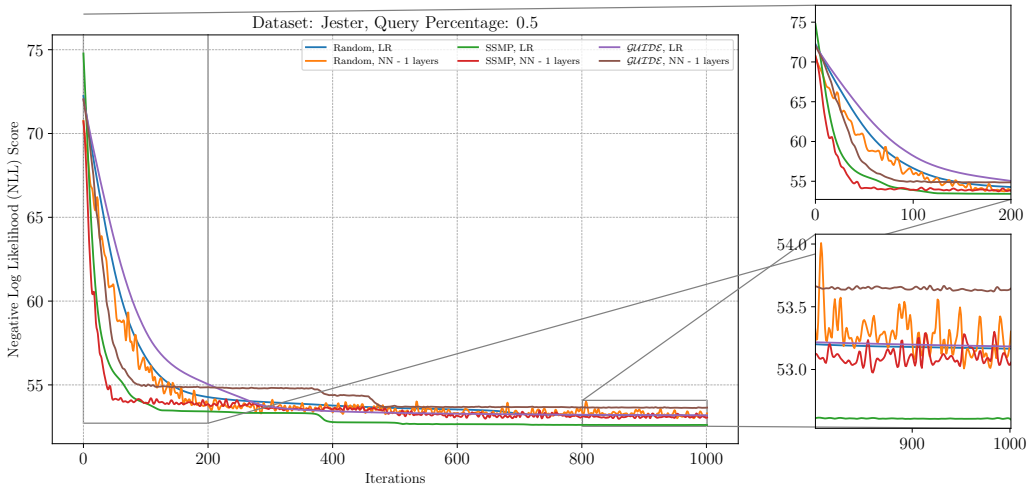


Figure 25: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the Jester Dataset at a Query Ratio of 0.5.

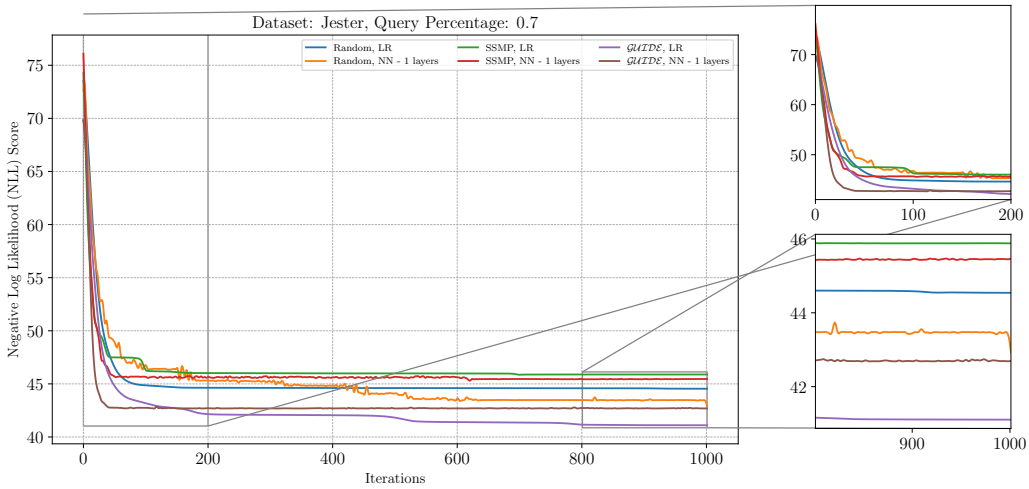


Figure 26: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the Jester Dataset at a Query Ratio of 0.7.

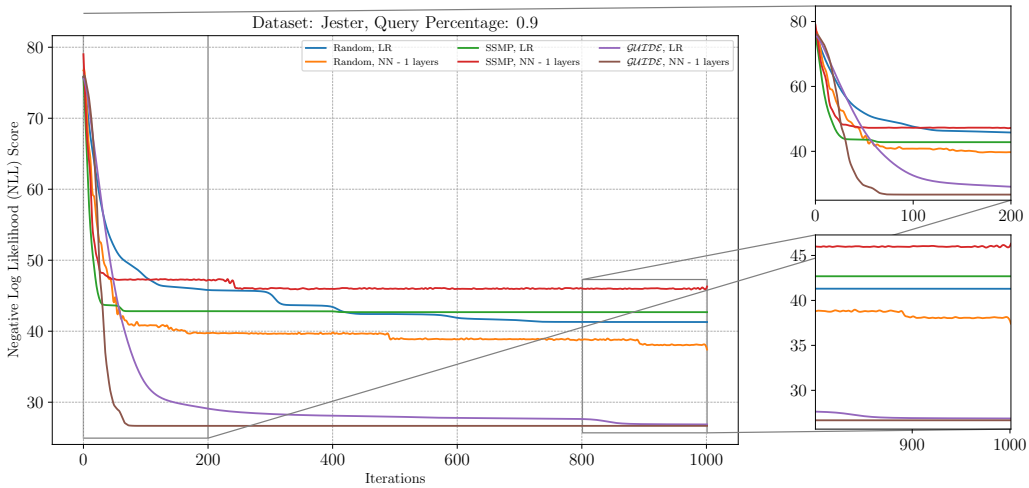


Figure 27: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for NAMs on the Jester Dataset at a Query Ratio of 0.9.

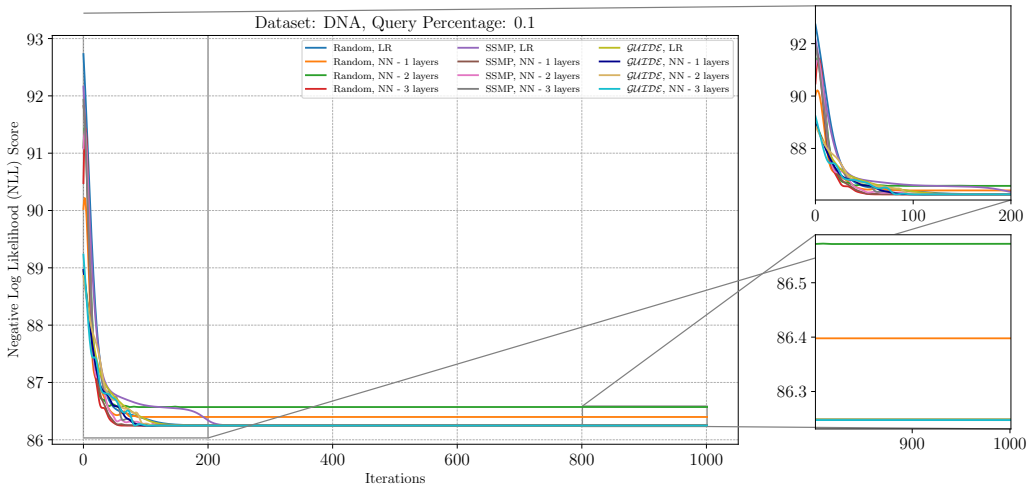


Figure 28: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the DNA Dataset at a Query Ratio of 0.1.

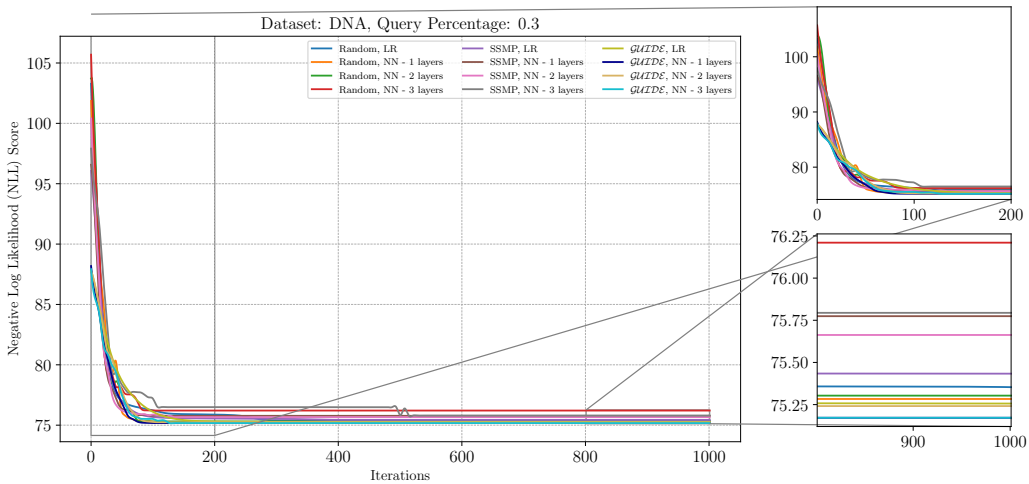


Figure 29: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the DNA Dataset at a Query Ratio of 0.3.

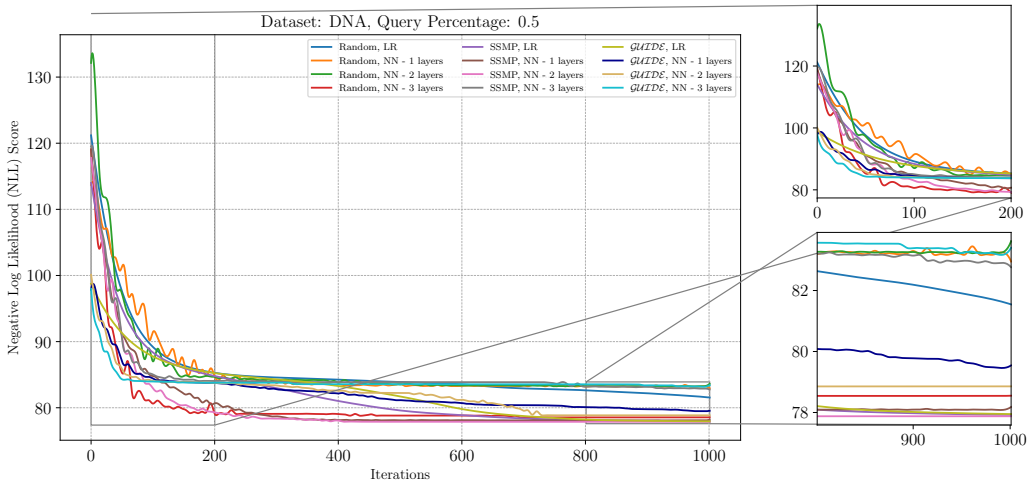


Figure 30: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the DNA Dataset at a Query Ratio of 0.5.

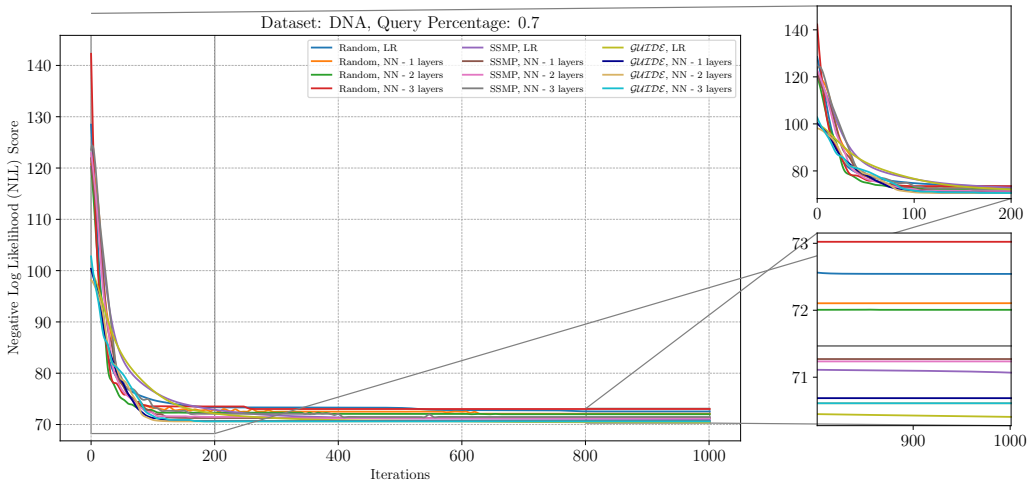


Figure 31: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the DNA Dataset at a Query Ratio of 0.7.

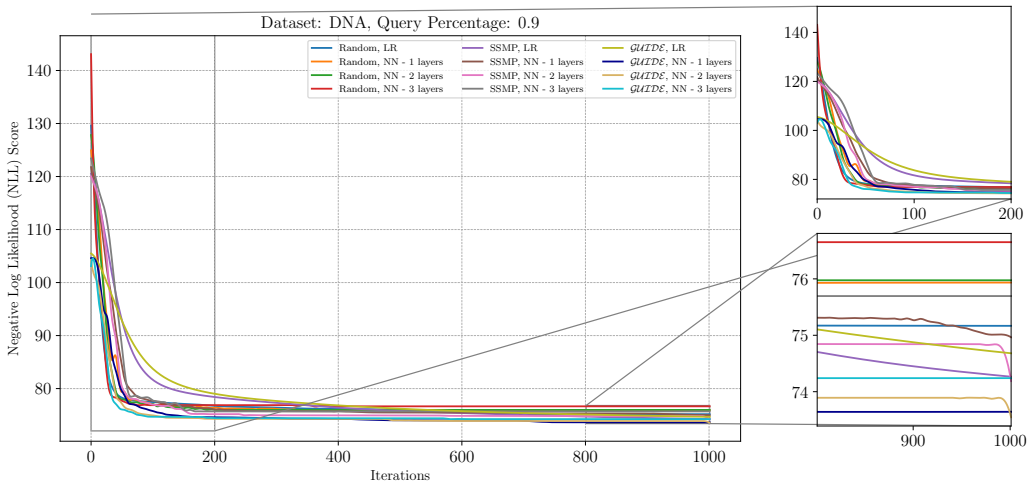


Figure 32: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the DNA Dataset at a Query Ratio of 0.9.

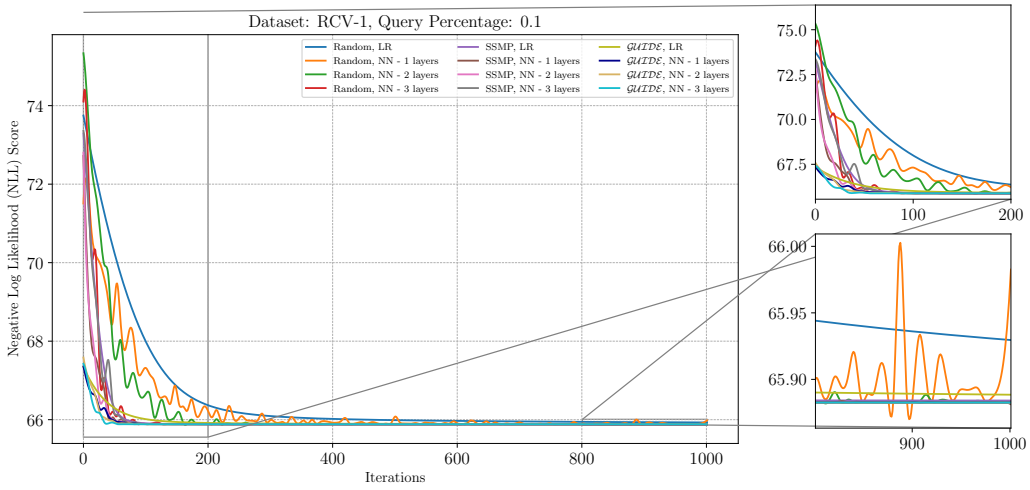


Figure 33: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the RCV-1 Dataset at a Query Ratio of 0.1.

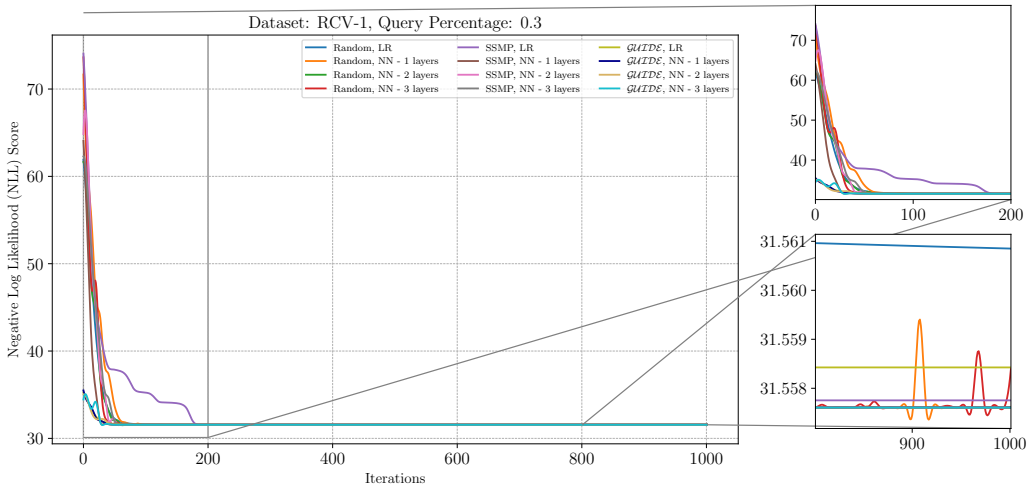


Figure 34: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the RCV-1 Dataset at a Query Ratio of 0.3.

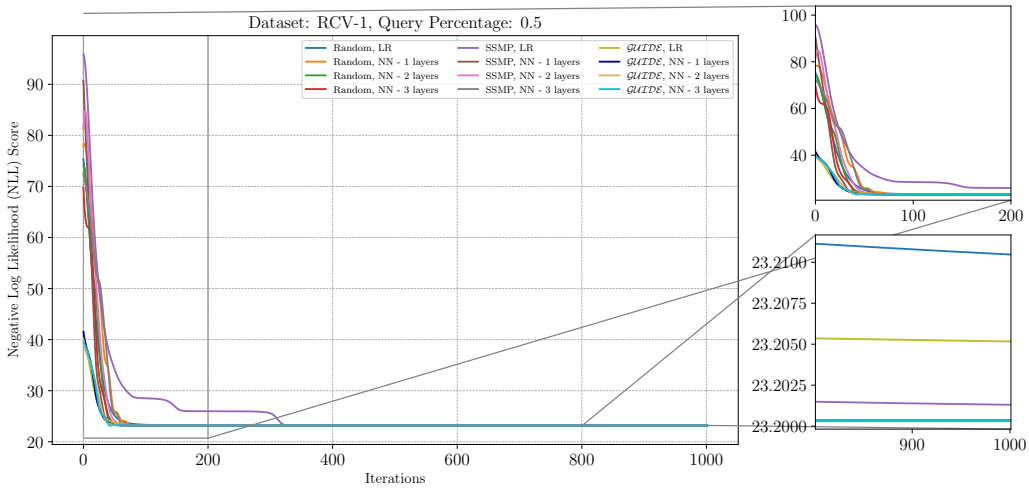


Figure 35: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the RCV-1 Dataset at a Query Ratio of 0.5.

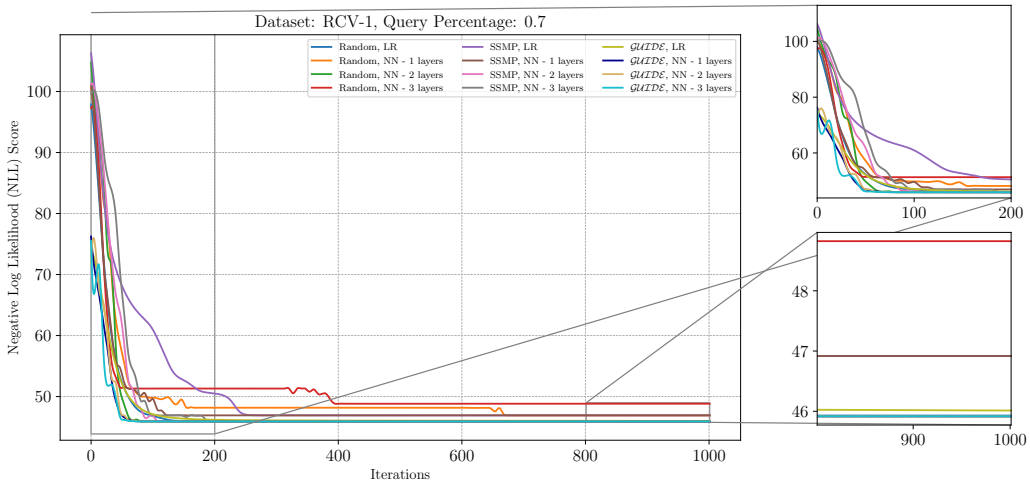


Figure 36: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the RCV-1 Dataset at a Query Ratio of 0.7.

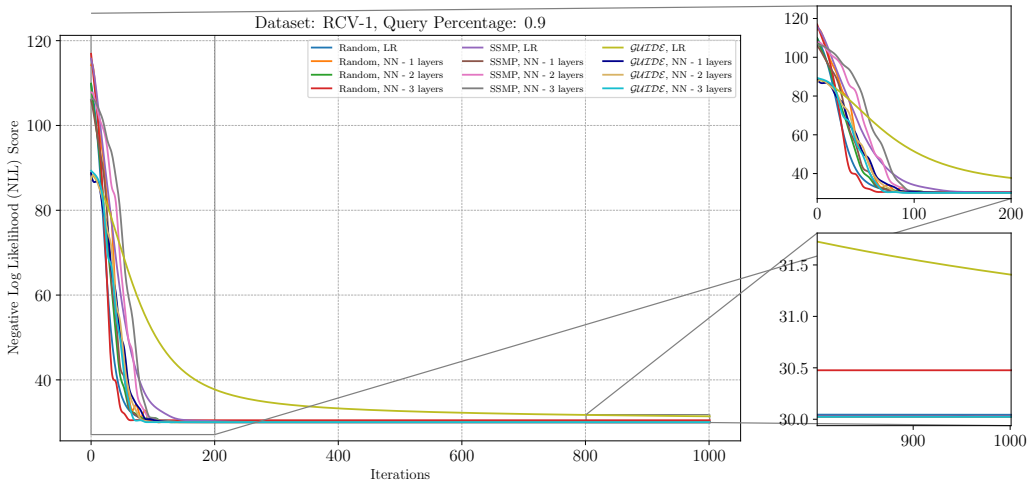


Figure 37: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the RCV-1 Dataset at a Query Ratio of 0.9.

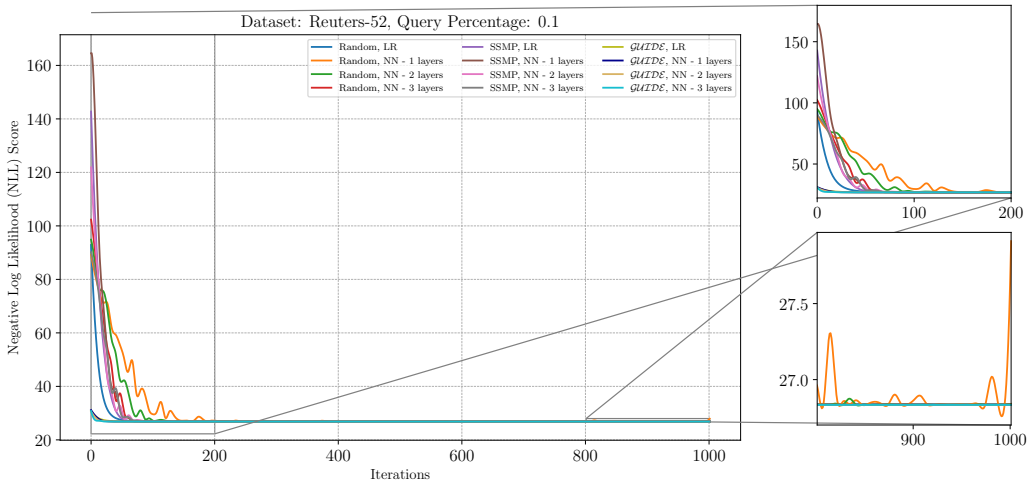


Figure 38: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Reuters-52 Dataset at a Query Ratio of 0.1.

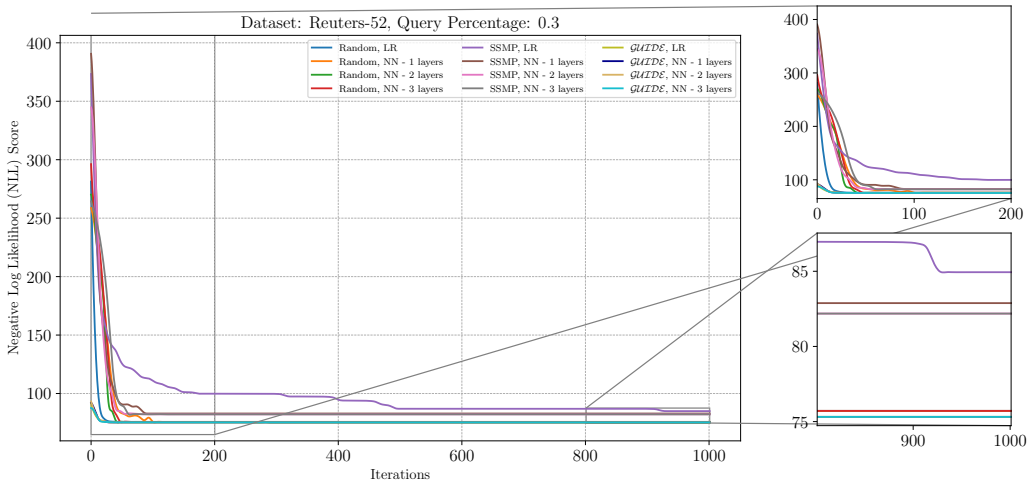


Figure 39: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Reuters-52 Dataset at a Query Ratio of 0.3.

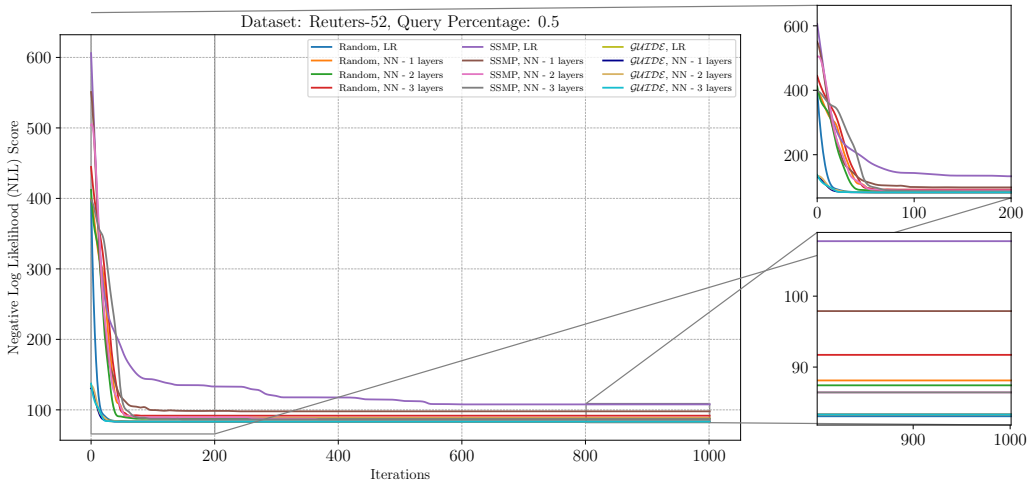


Figure 40: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Reuters-52 Dataset at a Query Ratio of 0.5.

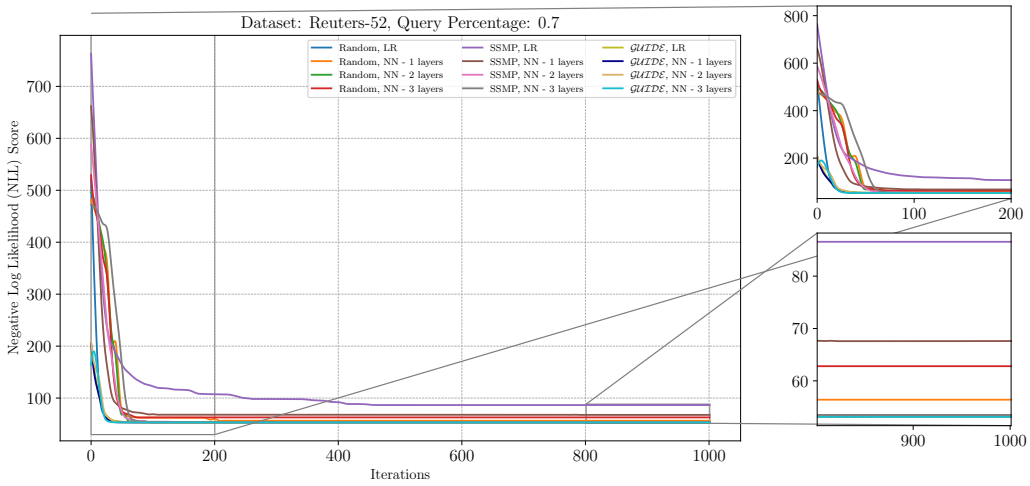


Figure 41: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Reuters-52 Dataset at a Query Ratio of 0.7.

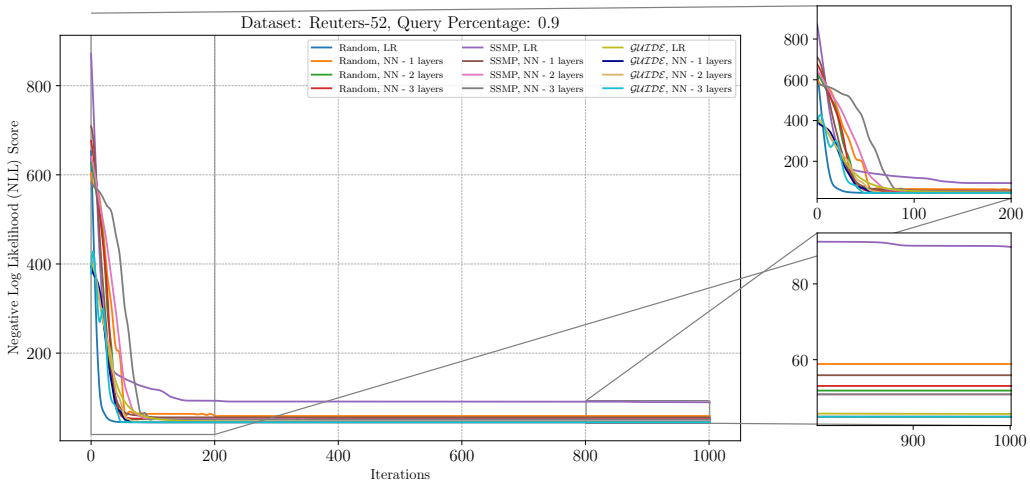


Figure 42: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Reuters-52 Dataset at a Query Ratio of 0.9.

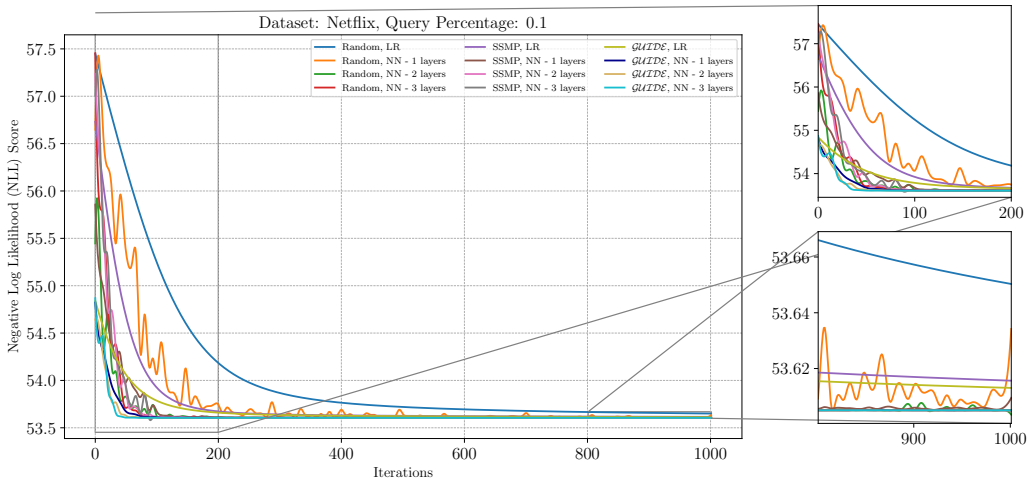


Figure 43: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Netflix Dataset at a Query Ratio of 0.1.

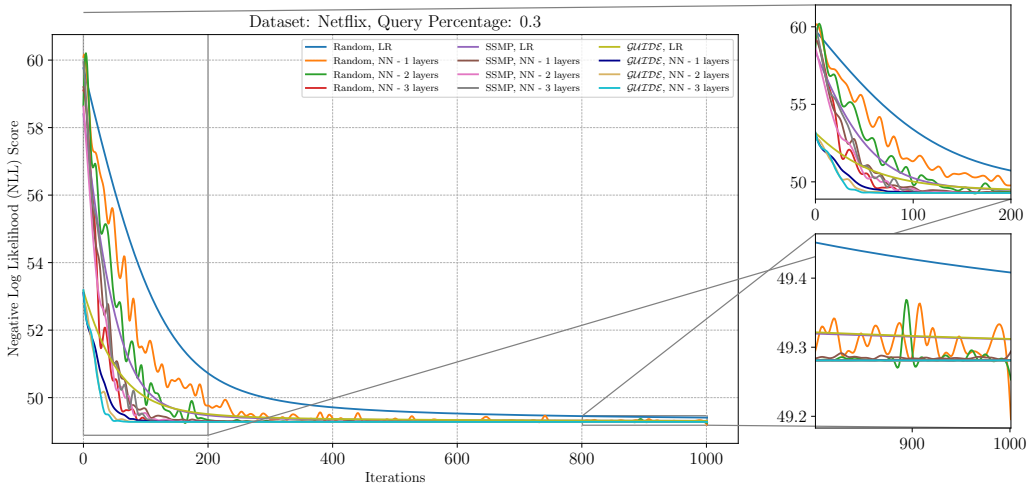


Figure 44: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Netflix Dataset at a Query Ratio of 0.3.

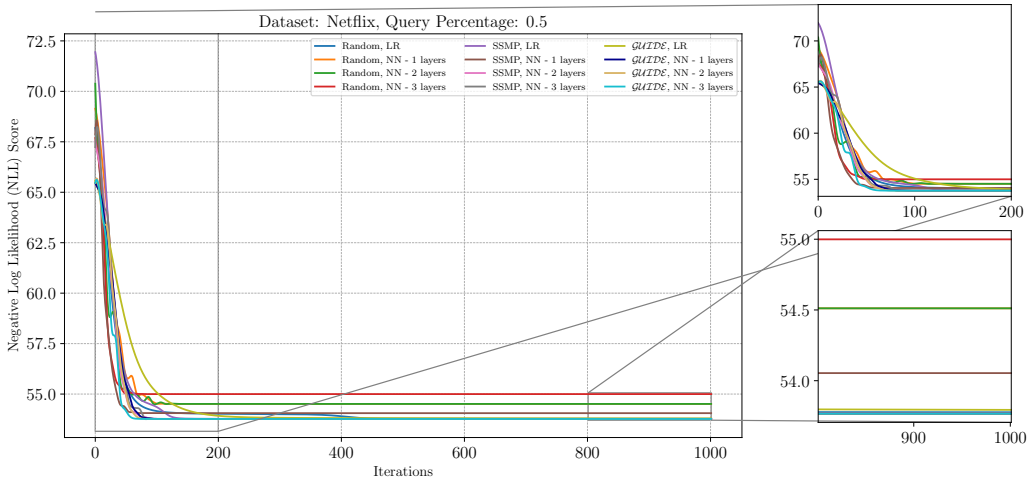


Figure 45: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Netflix Dataset at a Query Ratio of 0.5.

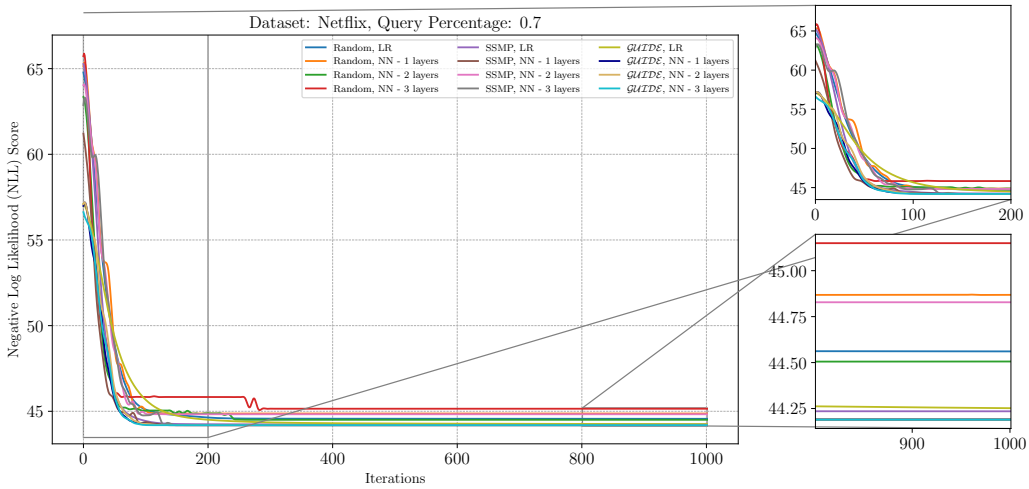


Figure 46: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Netflix Dataset at a Query Ratio of 0.7.

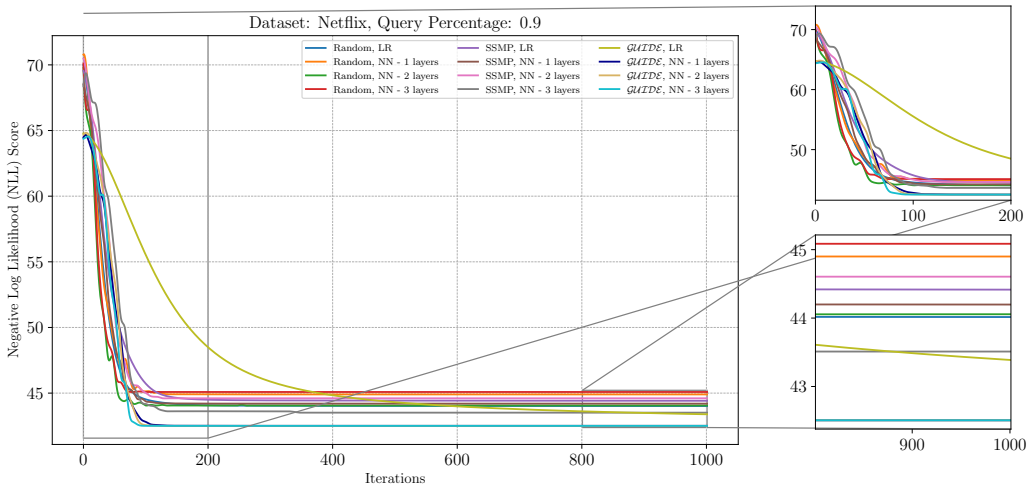


Figure 47: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Netflix Dataset at a Query Ratio of 0.9.

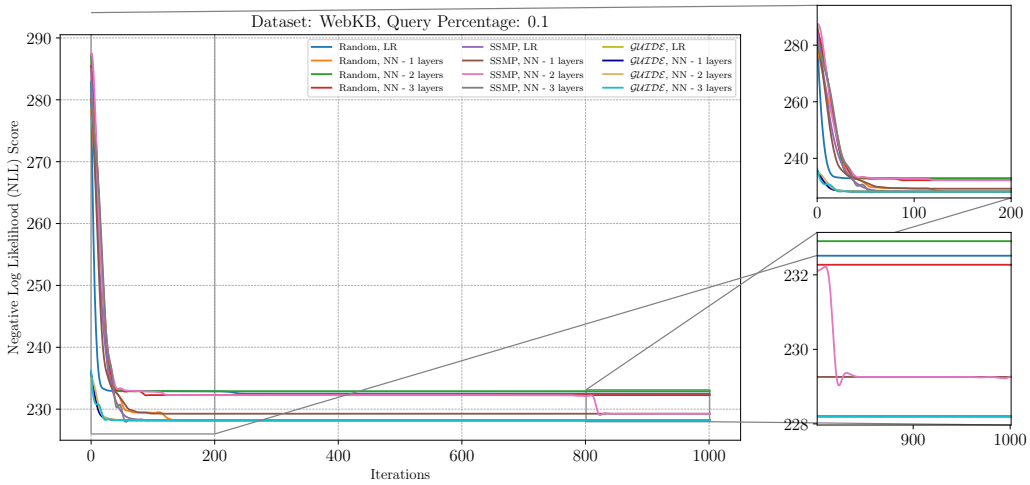


Figure 48: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the WebKB Dataset at a Query Ratio of 0.1.

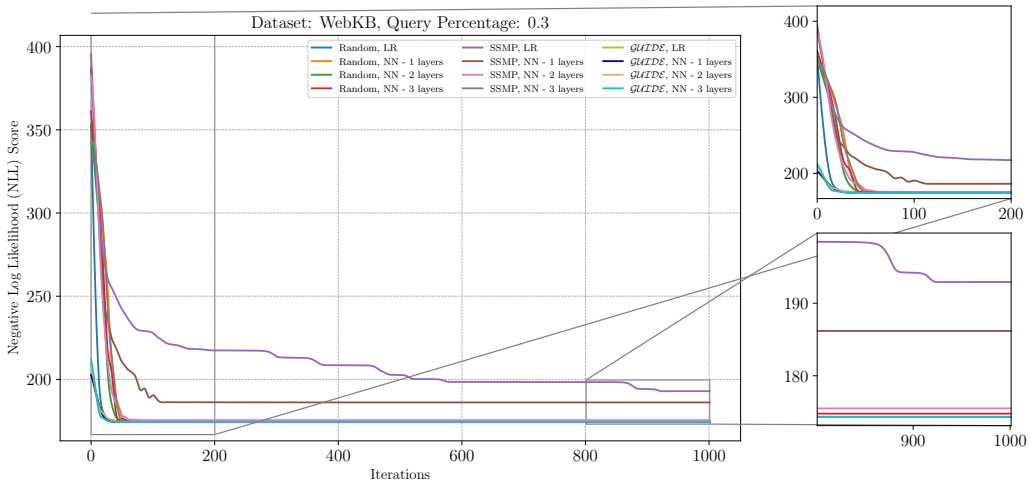


Figure 49: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the WebKB Dataset at a Query Ratio of 0.3.

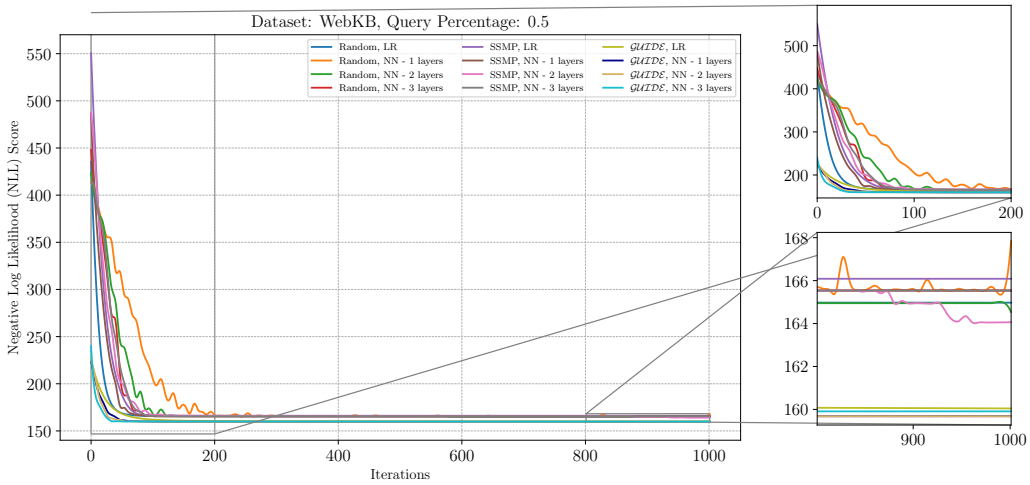


Figure 50: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the WebKB Dataset at a Query Ratio of 0.5.

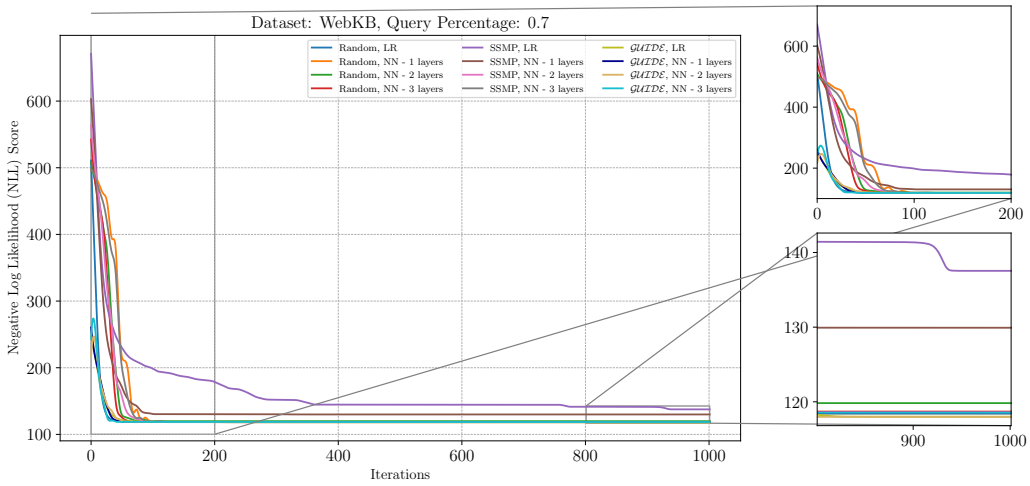


Figure 51: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the WebKB Dataset at a Query Ratio of 0.7.

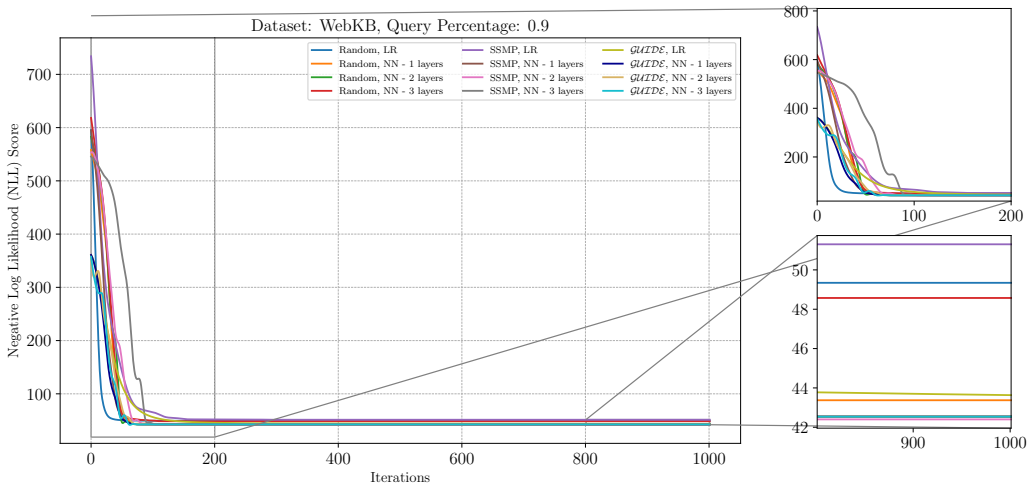


Figure 52: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the WebKB Dataset at a Query Ratio of 0.9.

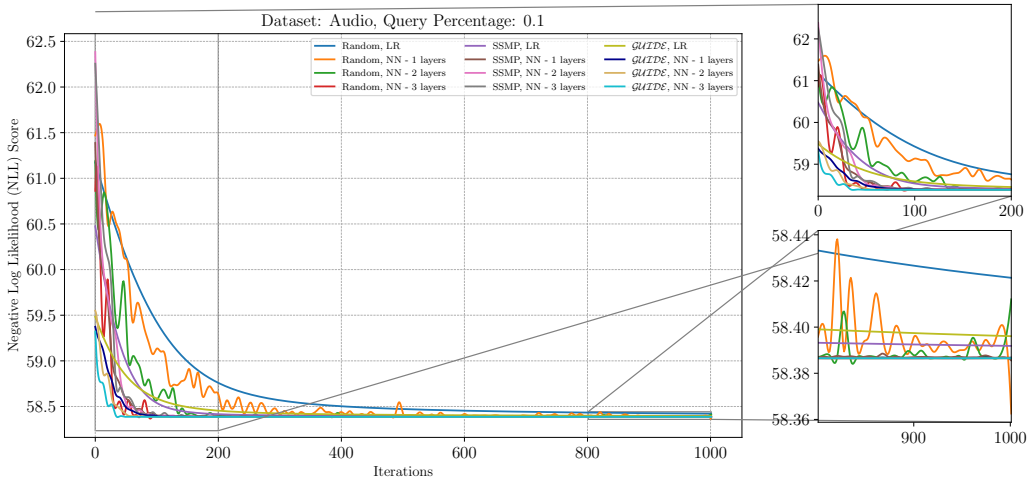


Figure 53: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Audio Dataset at a Query Ratio of 0.1.

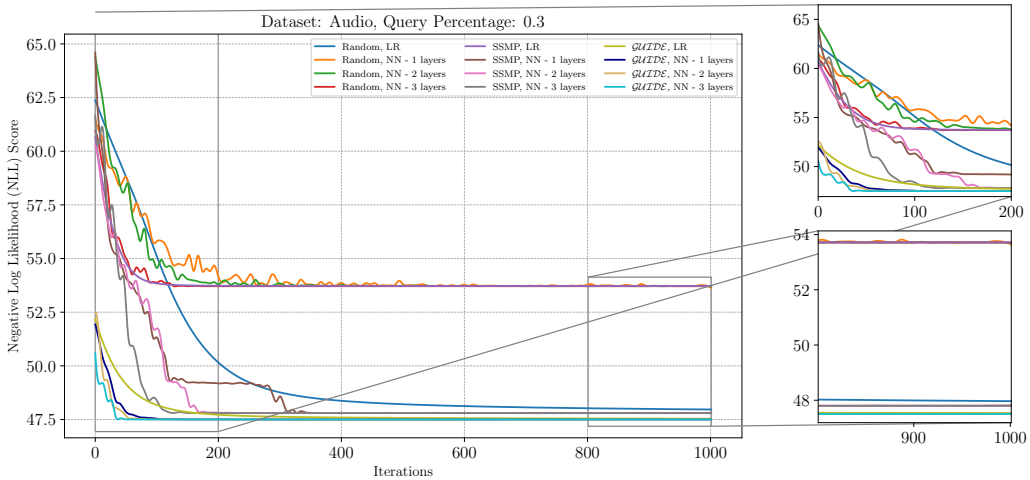


Figure 54: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Audio Dataset at a Query Ratio of 0.3.

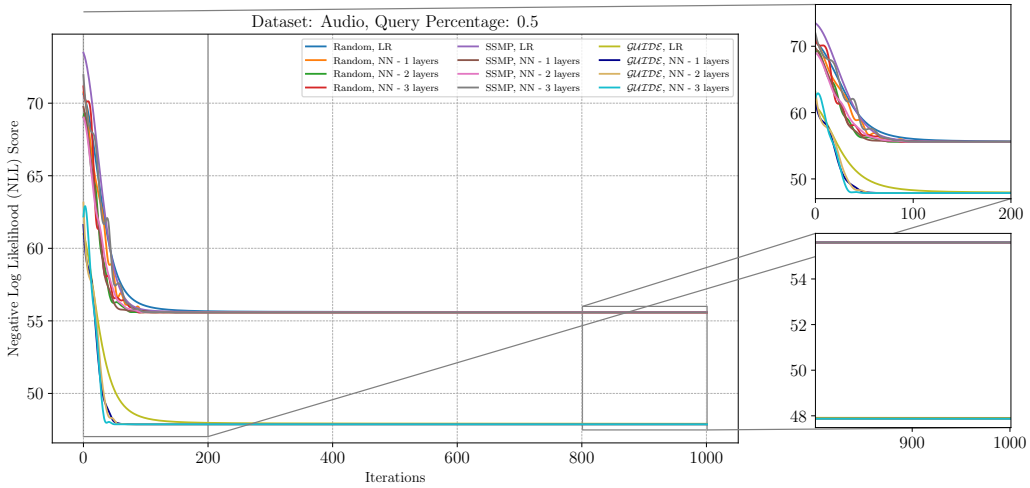


Figure 55: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Audio Dataset at a Query Ratio of 0.5.

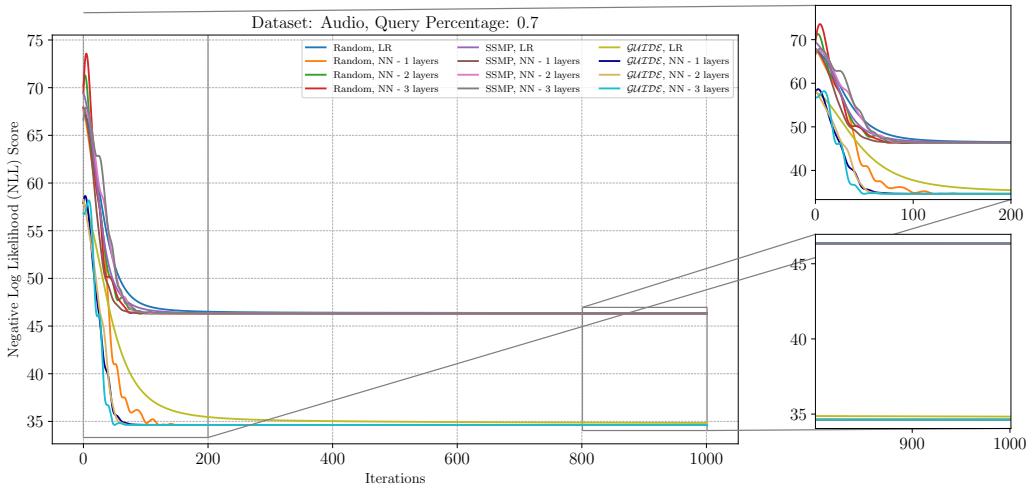


Figure 56: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Audio Dataset at a Query Ratio of 0.7.

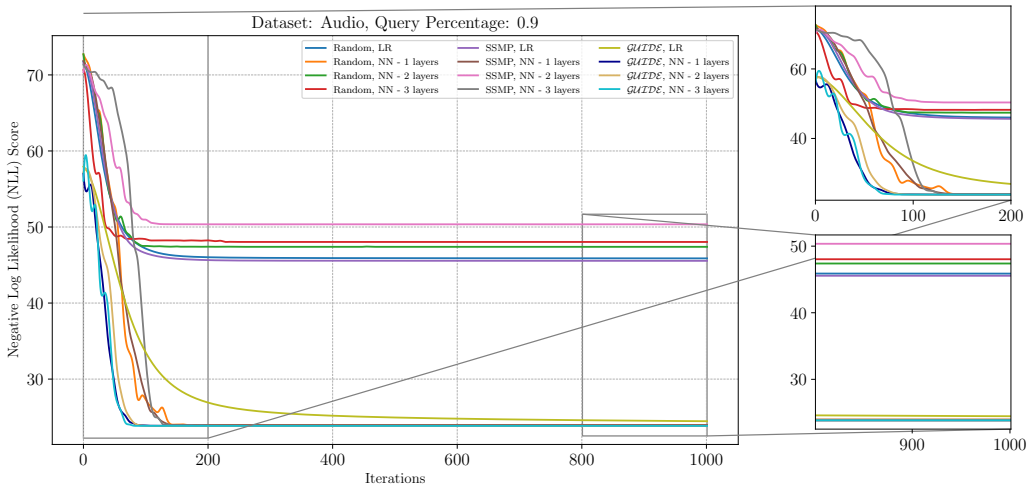


Figure 57: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Audio Dataset at a Query Ratio of 0.9.

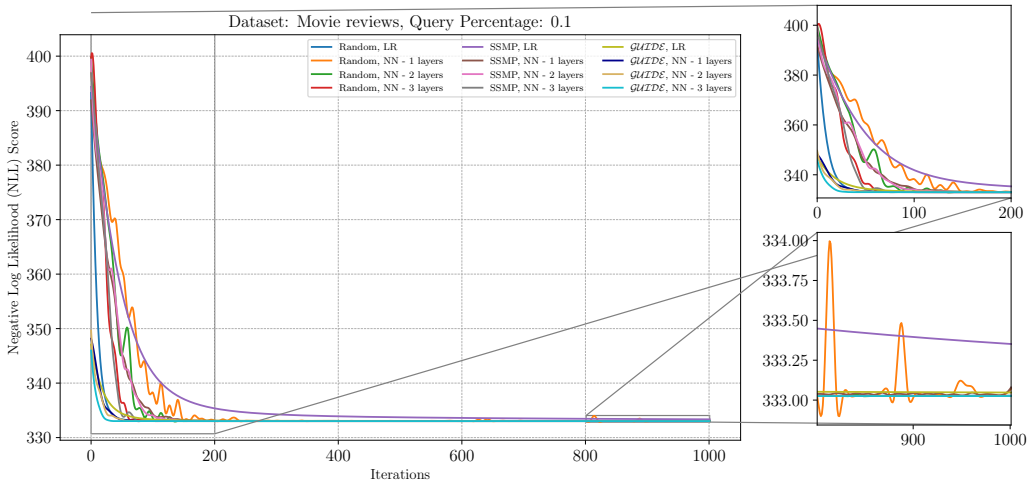


Figure 58: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Movie reviews Dataset at a Query Ratio of 0.1.

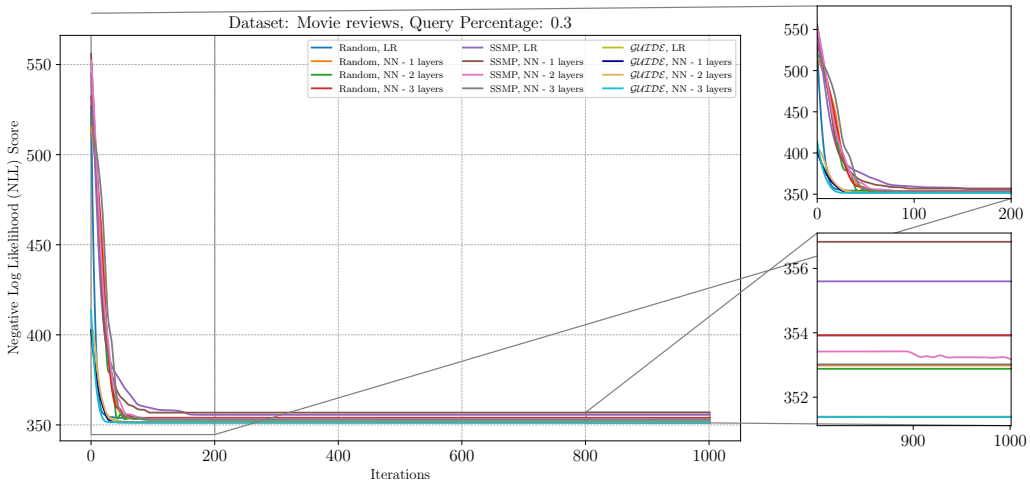


Figure 59: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Movie reviews Dataset at a Query Ratio of 0.3.

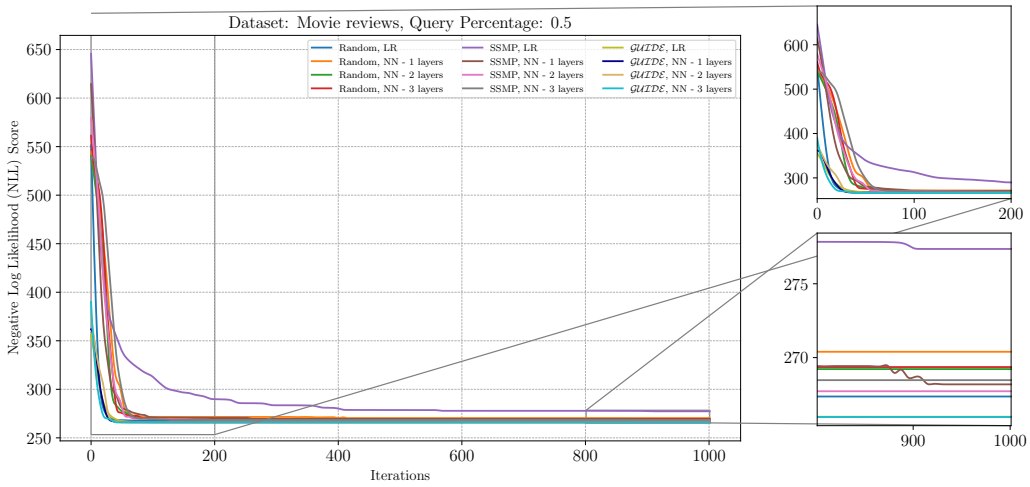


Figure 60: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Movie reviews Dataset at a Query Ratio of 0.5.

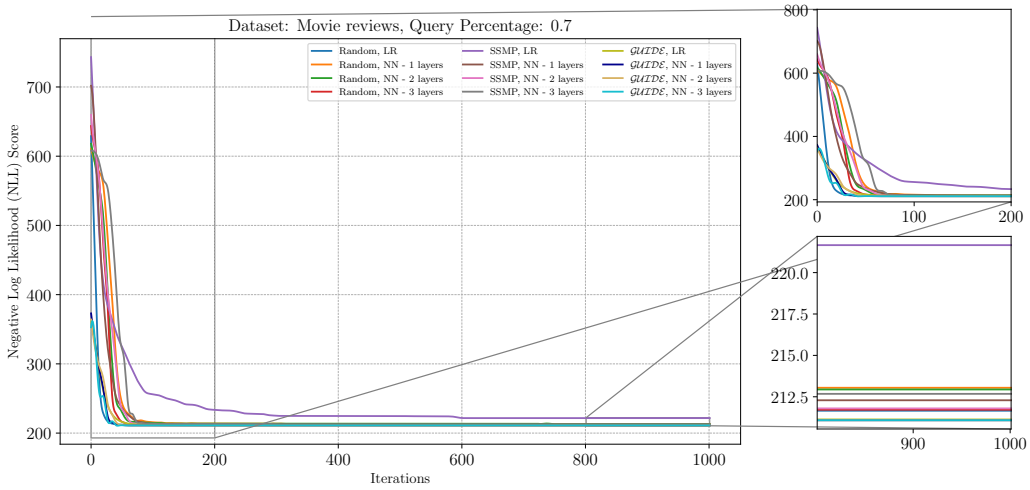


Figure 61: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Movie reviews Dataset at a Query Ratio of 0.7.

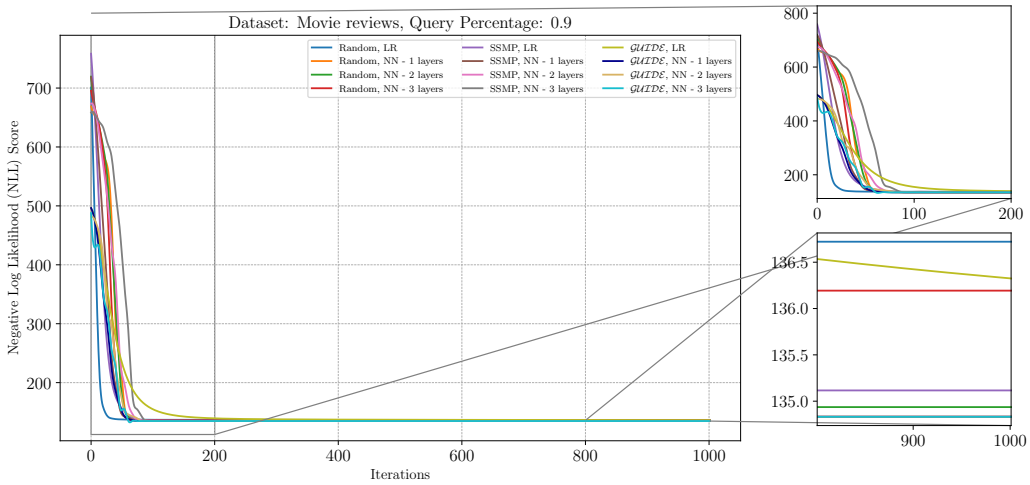


Figure 62: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Movie reviews Dataset at a Query Ratio of 0.9.

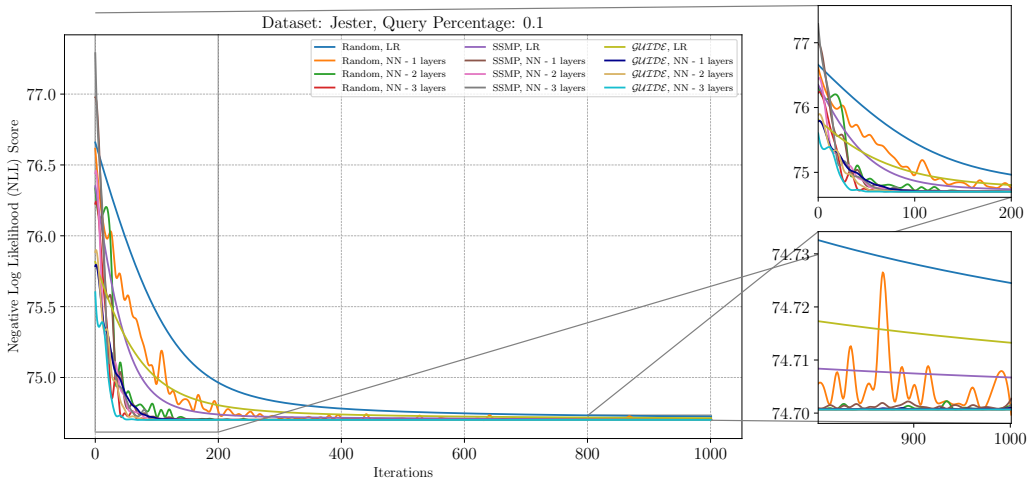


Figure 63: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Jester Dataset at a Query Ratio of 0.1.

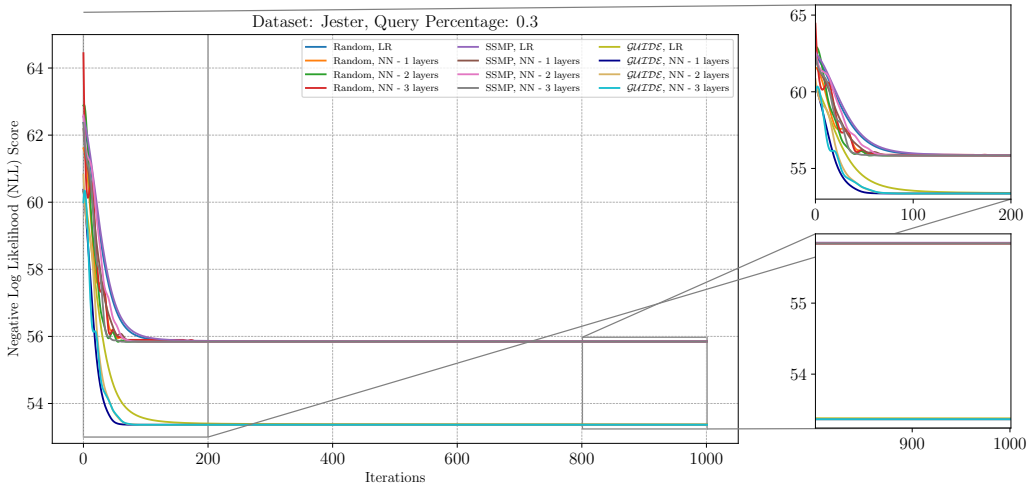


Figure 64: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Jester Dataset at a Query Ratio of 0.3.

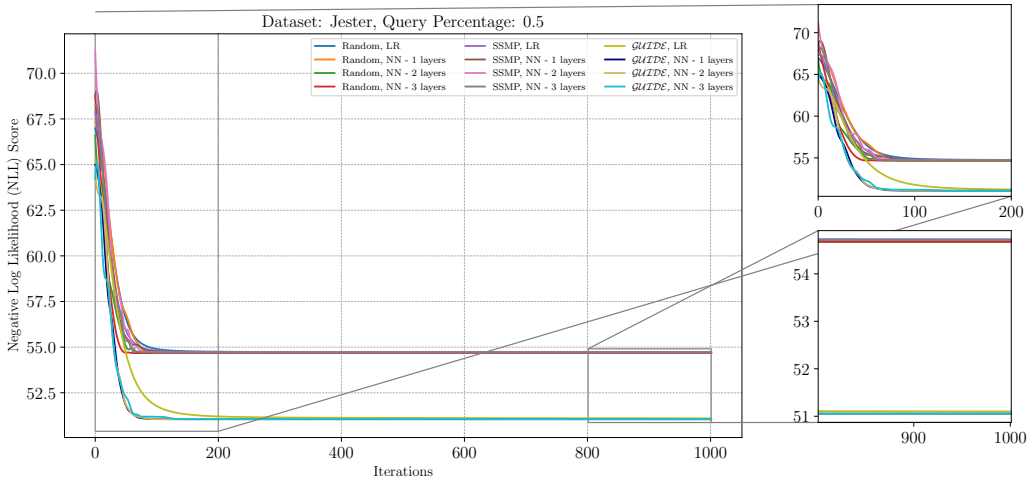


Figure 65: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Jester Dataset at a Query Ratio of 0.5.

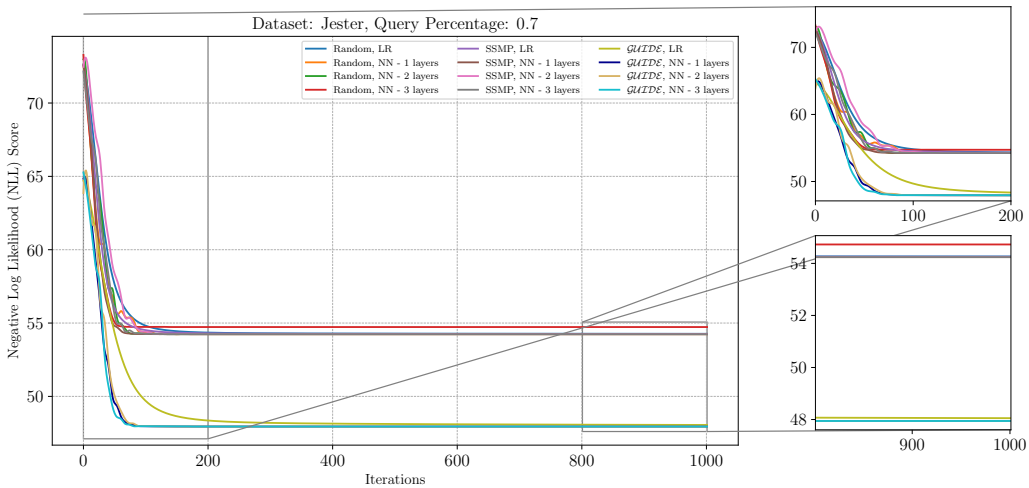


Figure 66: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PCs on the Jester Dataset at a Query Ratio of 0.7.

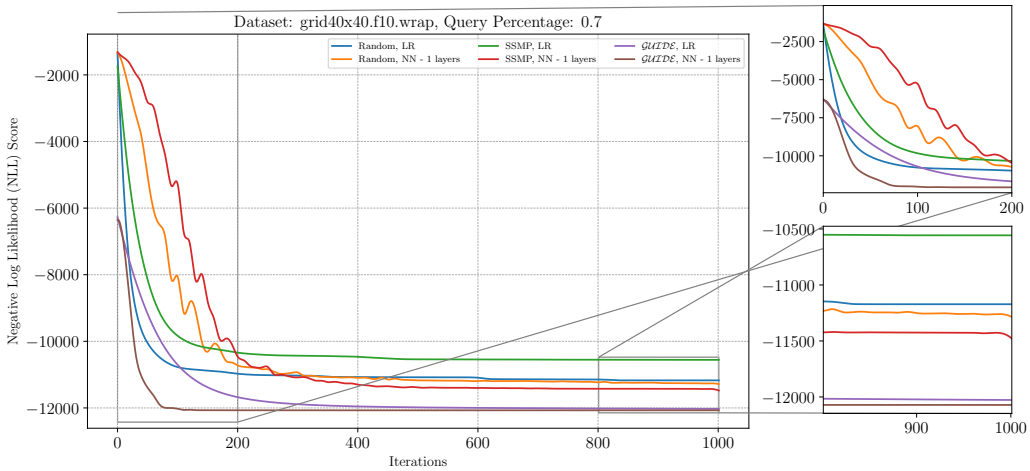


Figure 67: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PGMs on the grid40x40.f10.wrap Dataset at a Query Ratio of 0.7.

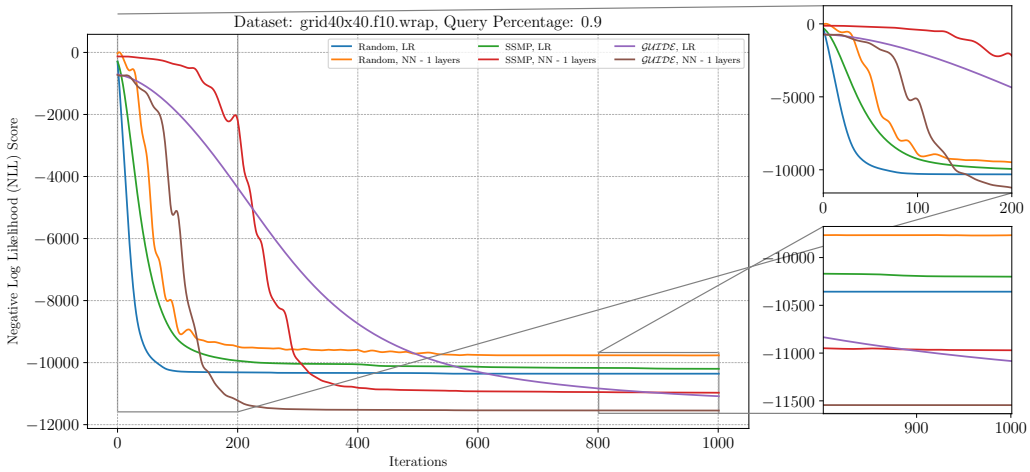


Figure 68: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PGMs on the grid40x40.f10.wrap Dataset at a Query Ratio of 0.9.

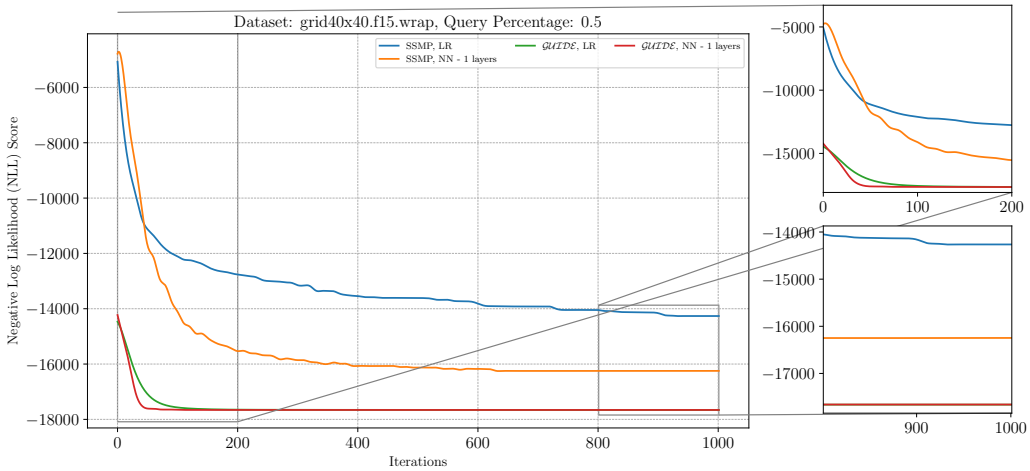


Figure 69: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PGMs on the grid40x40.f15.wrap Dataset at a Query Ratio of 0.5.

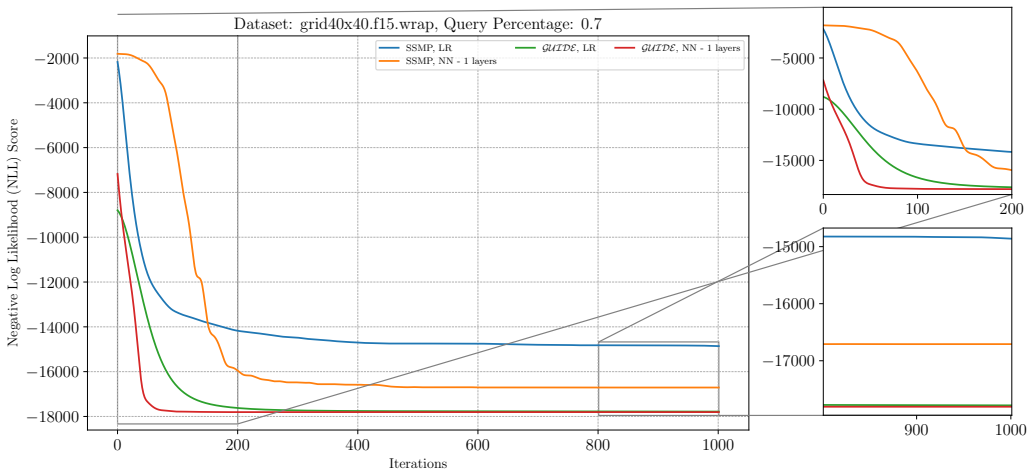


Figure 70: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PGMs on the grid40x40.f15.wrap Dataset at a Query Ratio of 0.7.

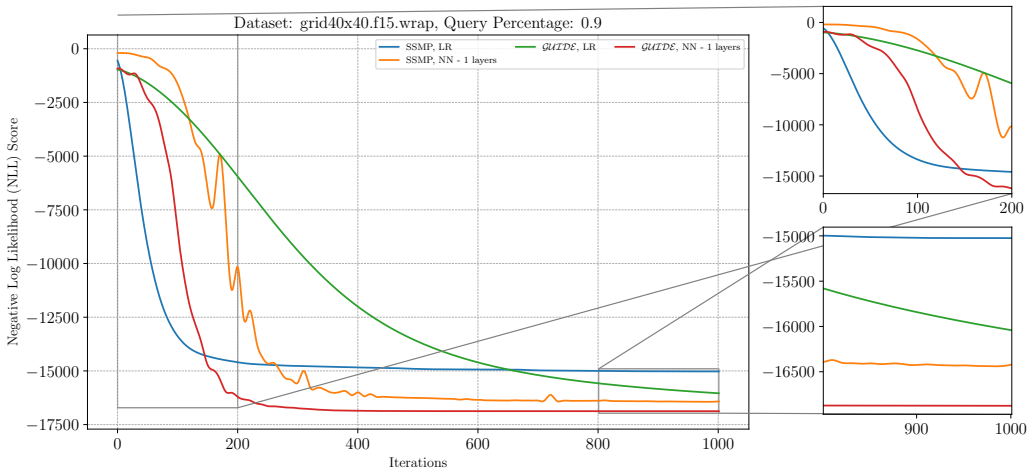


Figure 71: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PGMs on the grid40x40.f15.wrap Dataset at a Query Ratio of 0.9.

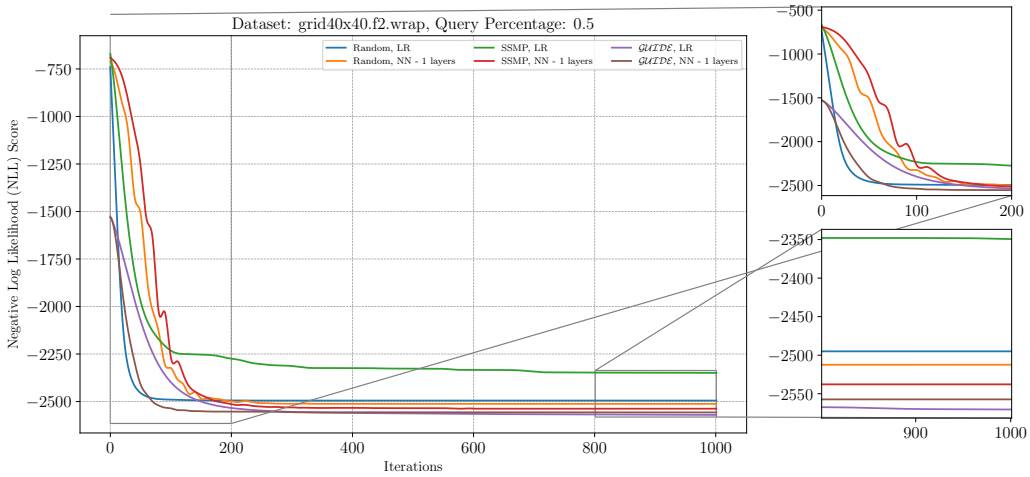


Figure 72: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PGMs on the grid40x40.f2.wrap Dataset at a Query Ratio of 0.5.

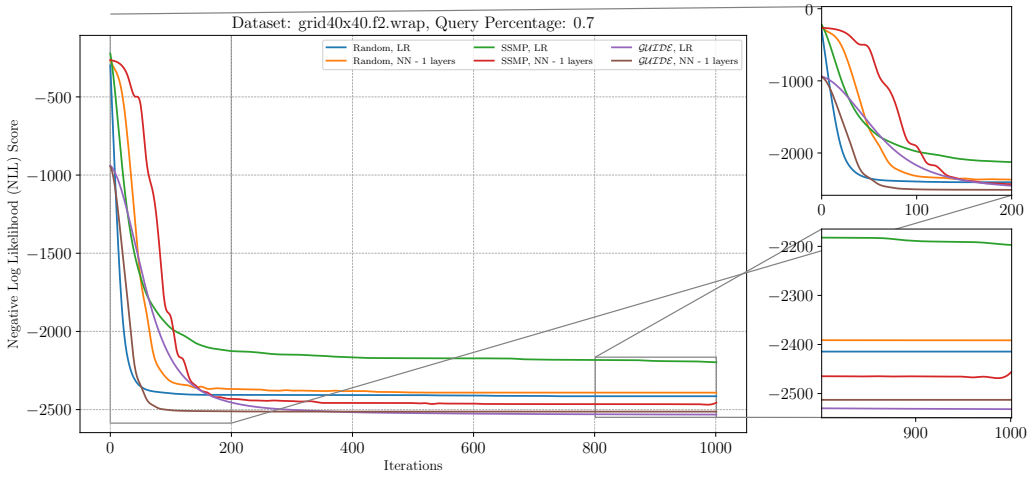


Figure 73: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PGMs on the grid40x40.f2.wrap Dataset at a Query Ratio of 0.7.

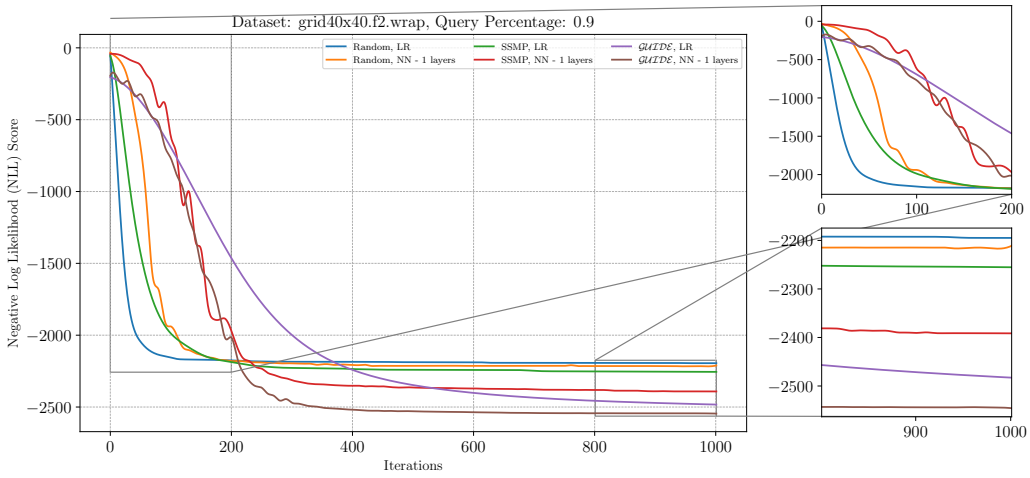


Figure 74: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PGMs on the grid40x40.f2.wrap Dataset at a Query Ratio of 0.9.

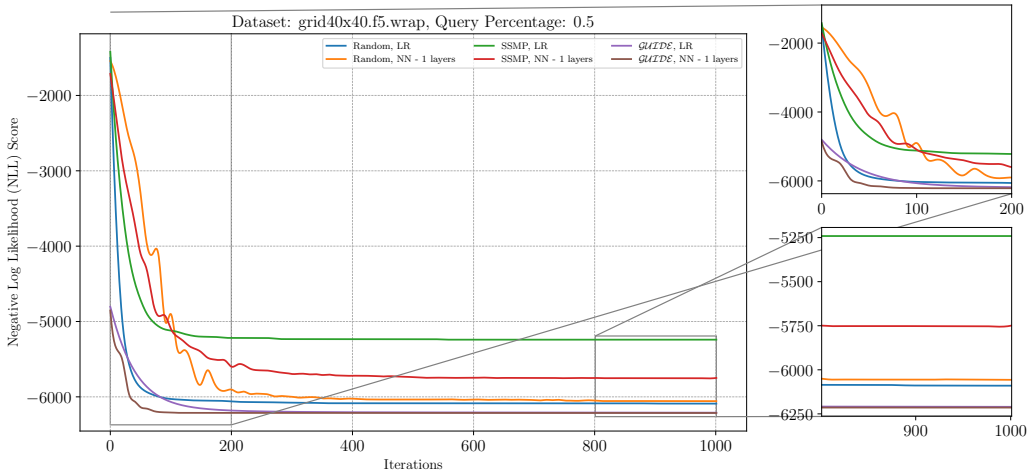


Figure 75: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PGMs on the grid40x40.f5.wrap Dataset at a Query Ratio of 0.5.

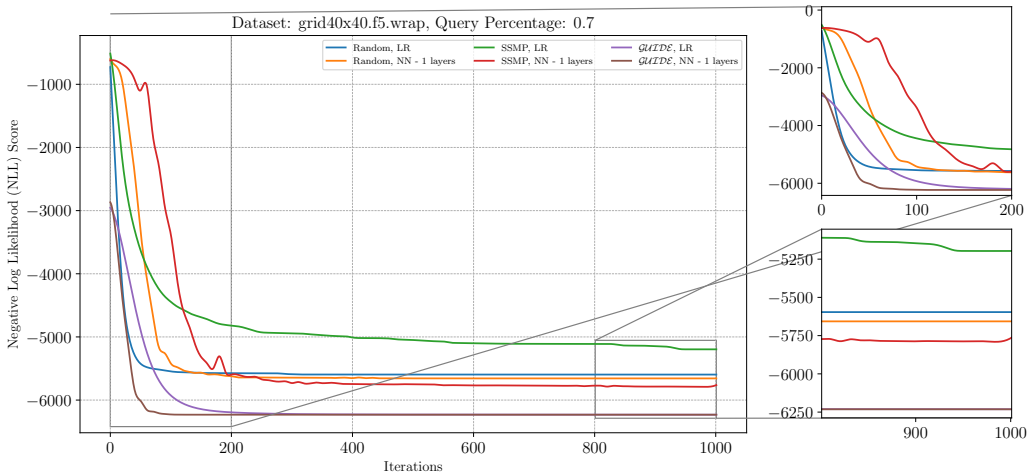


Figure 76: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PGMs on the grid40x40.f5.wrap Dataset at a Query Ratio of 0.7.

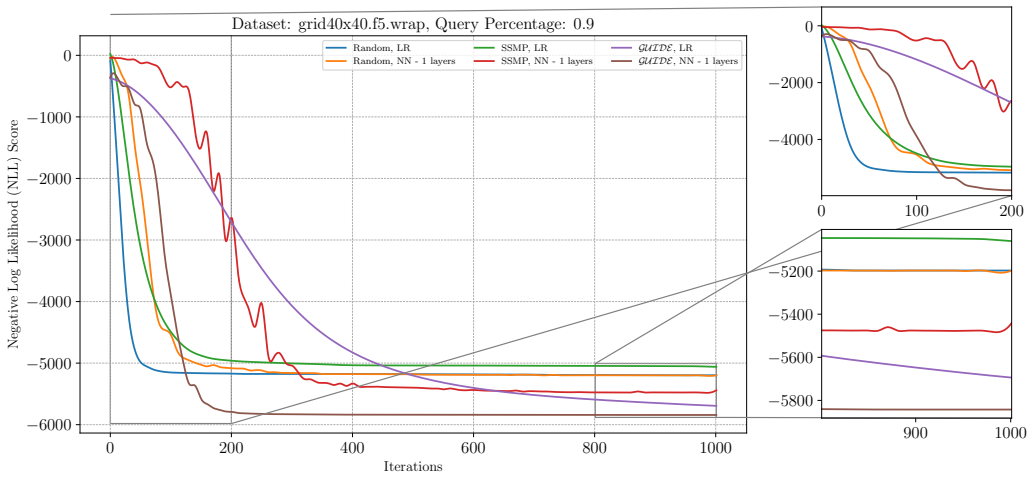


Figure 77: Analysis of **ITSELF** Loss Across Various Pre-Trained Models for PGMs on the grid40x40.f5.wrap Dataset at a Query Ratio of 0.9.

D INFERENCE TIME COMPARISON

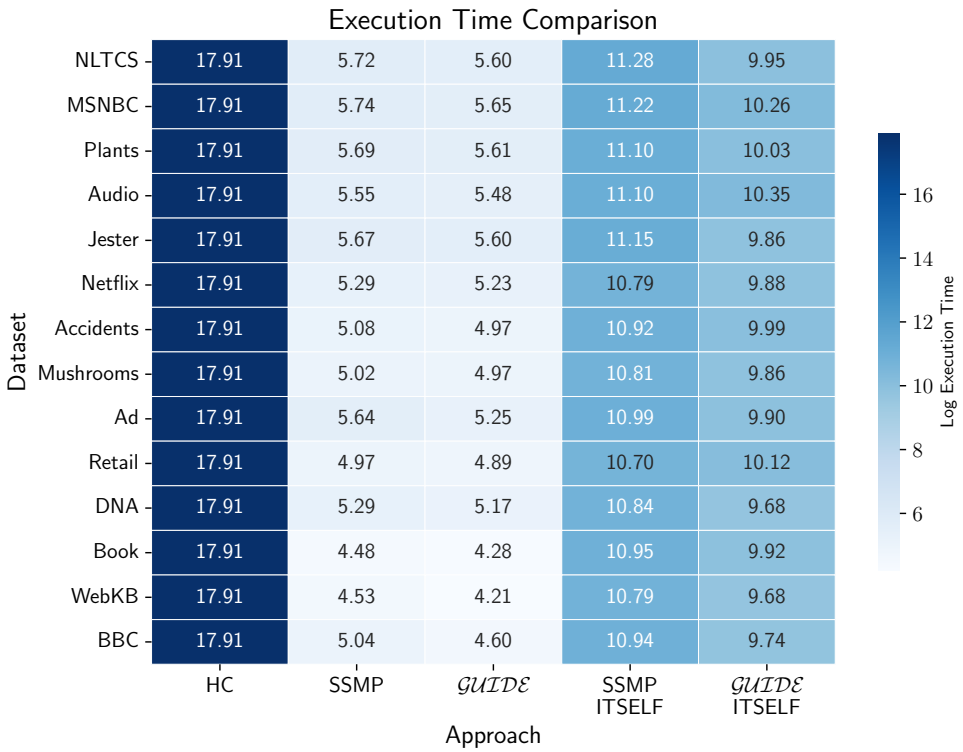


Figure 78: Heatmap depicting the inference time for MADE on a logarithmic microsecond scale, where a lighter color denotes shorter (more favorable) durations.

We present the inference times for all baselines and proposed methods in Figures 78 to 80. Figure 78 details the inference times for MADE, while Figures 79 and 80 respectively illustrate the times for PCs and PGMs. This comparison facilitates a direct evaluation of the computational efficiency across different models.

Each cell displays the natural logarithm of the time, measured in microseconds, for each method and dataset. Lighter colors indicate lower values. Notably, inferences using SSMP and $\mathcal{GUIDE}$ require the shortest time, as these methods necessitate only a single forward pass through the neural network to obtain the values for the query variables.

For MADE, the subsequent fastest method employs a model trained with $\mathcal{GUIDE}$ and conducts inference using **ITSELF**, outperforming the approach that uses SSMP for training. This advantage stems from the reduced number of **ITSELF** iterations required by $\mathcal{GUIDE}$, benefiting from a more effectively trained model. In PGMs, a similar pattern emerges with $\mathcal{GUIDE}$ + **ITSELF** as the next fastest method, followed by SSMP + **ITSELF**. For PCs, MAX ranks as the next fastest, closely followed by the $\mathcal{GUIDE}$ + **ITSELF** and SSMP + **ITSELF** methods. Finally, the ML and Seq methods display the highest inference times.

Thus, if you require a highly efficient method capable of performing inference in a fraction of a millisecond, $\mathcal{GUIDE}$ is the optimal choice. It outperforms the baseline for both MADE and PGMs. However, if higher log-likelihood scores are necessary, $\mathcal{GUIDE}$ + **ITSELF** would be suitable, as it generally surpasses the baselines in speed and performance across various scenarios.

E GAP ANALYSIS FOR PGM

Table 2 presents the gap in log-likelihood scores between the neural network techniques (SSMP, $\mathcal{GUIDE}$, SSMP + **ITSELF**, $\mathcal{GUIDE}$ + **ITSELF**) and the baseline model (AOBB). For each approach M, the gap is calculated as the relative difference between the score of the near-optimal solution (determined by AOBB) and the score achieved by M. The practicality of this evaluation stems from the utilization of small datasets, which enables the identification of exact solutions.

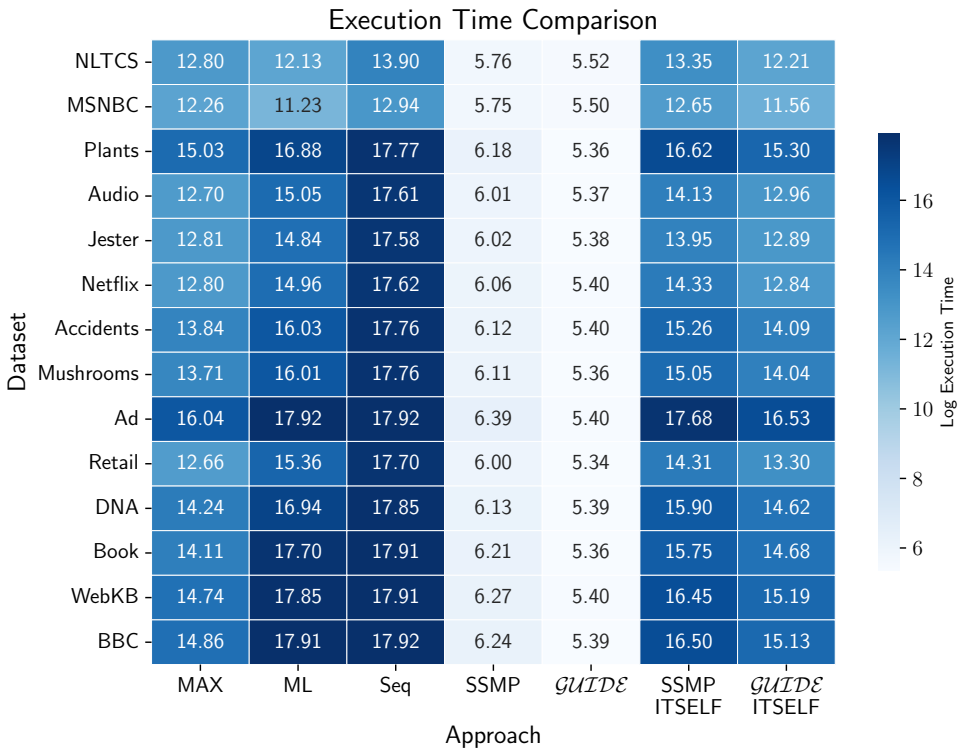


Figure 79: Heatmap depicting the inference time for PC on a logarithmic microsecond scale, where a lighter color denotes shorter (more favorable) durations.

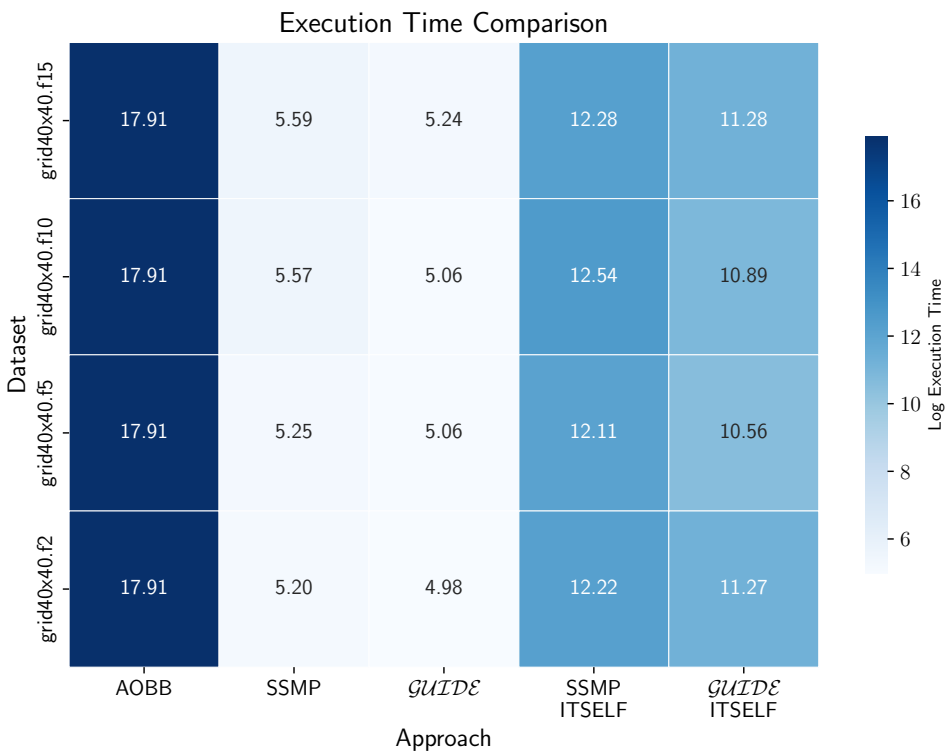


Figure 80: Heatmap depicting the inference time for PGM on a logarithmic microsecond scale, where a lighter color denotes shorter (more favorable) durations.

The final column emphasizes the neural-based approach that demonstrates superior performance for each dataset and query ratio combination. Notably, $\mathcal{GUIDE}$ and **ITSELF** consistently surpass other neural baselines across almost all dataset-query pairs. This examination offers a comprehensive assessment of the proposed methods on small datasets, facilitating a comparison of their effectiveness.

Table 2: Gap Between AOBB And Other Methods.

Method	Query Ratio	SSMP	$\mathcal{GUIDE}$	SSMP + ITSELF	$\mathcal{GUIDE}$ + ITSELF	Best Method
Grids-17	0.900	0.082	0.060	0.069	0.075	$\mathcal{GUIDE}$
Grids-17	0.800	0.051	0.038	0.040	0.035	$\mathcal{GUIDE}$ + **ITSELF**
Grids-17	0.700	0.042	0.030	0.034	0.016	$\mathcal{GUIDE}$ + **ITSELF**
Grids-17	0.500	0.026	0.024	0.024	0.007	$\mathcal{GUIDE}$ + **ITSELF**
Grids-18	0.900	0.081	0.062	0.071	0.102	$\mathcal{GUIDE}$
Grids-18	0.700	0.033	0.027	0.024	0.015	$\mathcal{GUIDE}$ + **ITSELF**
Grids-18	0.500	0.020	0.018	0.018	0.006	$\mathcal{GUIDE}$ + **ITSELF**
Grids-18	0.800	0.054	0.035	0.045	0.037	$\mathcal{GUIDE}$
Segmentation-14	0.500	0.032	0.032	0.032	0.004	$\mathcal{GUIDE}$ + **ITSELF**
Segmentation-14	0.900	0.045	0.014	0.014	0.005	$\mathcal{GUIDE}$ + **ITSELF**
Segmentation-14	0.800	0.051	0.024	0.024	0.006	$\mathcal{GUIDE}$ + **ITSELF**
Segmentation-14	0.700	0.029	0.029	0.029	0.005	$\mathcal{GUIDE}$ + **ITSELF**
Segmentation-15	0.800	0.046	0.002	0.002	0.002	$\mathcal{GUIDE}$ + **ITSELF**
Segmentation-15	0.500	0.003	0.003	0.003	0.000	$\mathcal{GUIDE}$ + **ITSELF**
Segmentation-15	0.900	0.675	0.255	0.433	0.305	$\mathcal{GUIDE}$
Segmentation-15	0.700	0.003	0.003	0.003	0.002	$\mathcal{GUIDE}$ + **ITSELF**

F LOG LIKELIHOOD SCORES COMPARISON

This section compares log-likelihood scores across baselines, SSMP, SSMP + **ITSELF**, $\mathcal{GUIDE}$ and $\mathcal{GUIDE}$ + **ITSELF** for all datasets and PMs. The log likelihood plots for NAMs are depicted in Figures 81 to 100, while those for PCs are illustrated in Figures 101 to 120. Each bar represents the mean log likelihood score of the corresponding method, with tick marks indicating the mean ± standard deviation. Higher values in these scores signify better performance by the method, considering they represent log likelihood scores.

F.1 SCORES FOR NAM

Figures 81 to 100 present the log likelihood scores for NAMs, illustrating the performance of **ITSELF** inference and $\mathcal{GUIDE}$ training relative to other baselines. The heatmaps and contingency tables discussed in the main paper corroborate the superior performance of $\mathcal{GUIDE}$ + **ITSELF**. These visual representations allow for a comprehensive understanding of the performance of our methods and baseline approaches, including HC and SSMP, across various datasets and query ratios.

F.2 SCORES FOR PCS

Analyzing Figures 101 to 120, which focuses on PCs, reveals similar patterns. The neural-based methods significantly outperform the MAX baseline. Among these, $\mathcal{GUIDE}$ + **ITSELF** surpasses all other polynomial-time baselines and neural methods in over 80 percent of the experiments. This demonstrates that **ITSELF** substantially enhances the chances of approaching optimal solutions by performing test-time optimization. When comparing traditional inference with **ITSELF**, **ITSELF** consistently proves superior. Moreover, $\mathcal{GUIDE}$ outperforms the other neural-based training methods (SSMP).

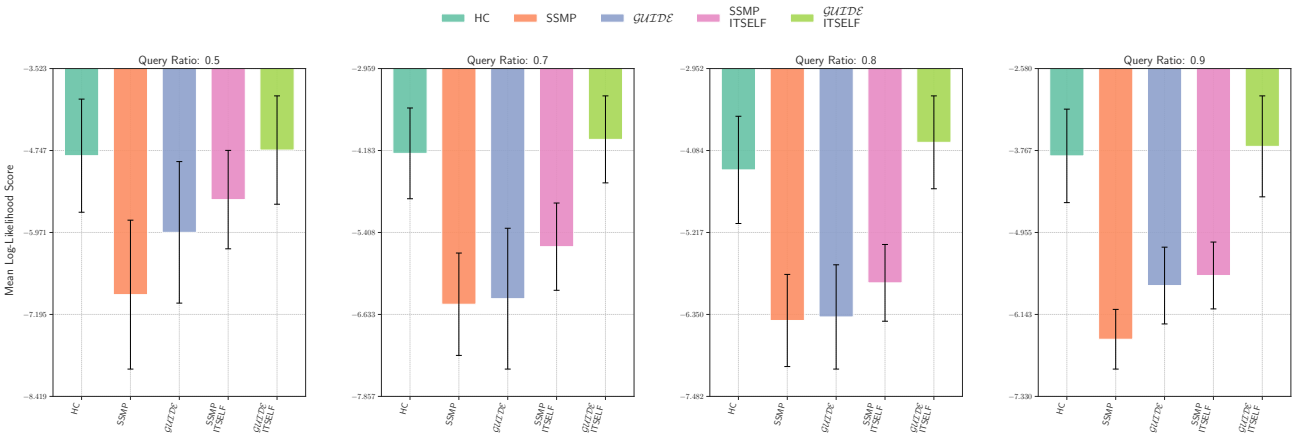


Figure 81: Log-Likelihood Scores on NLTCS for NAM. Higher Scores Indicate Better Performance.

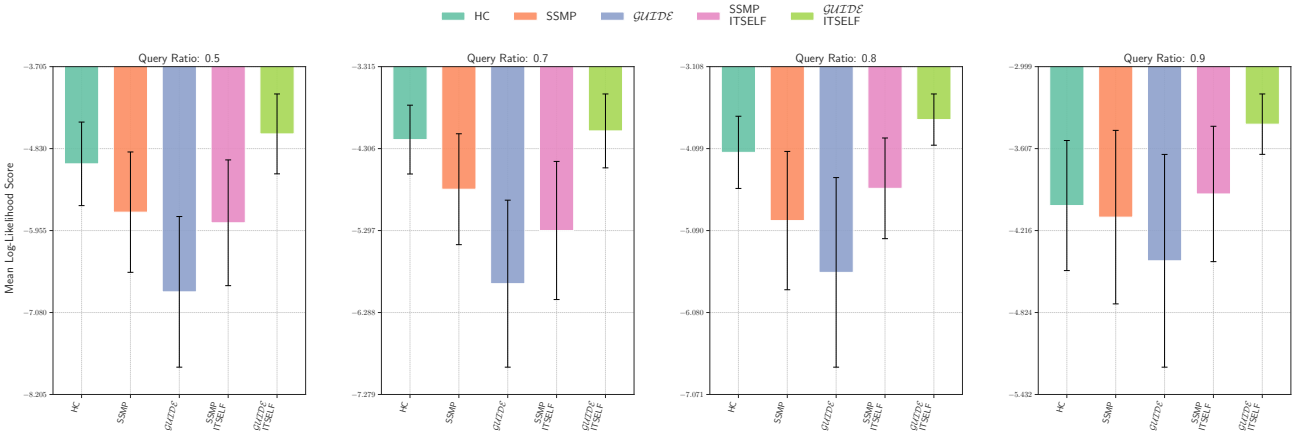


Figure 82: Log-Likelihood Scores on for NAM. Higher Scores Indicate Better Performance.

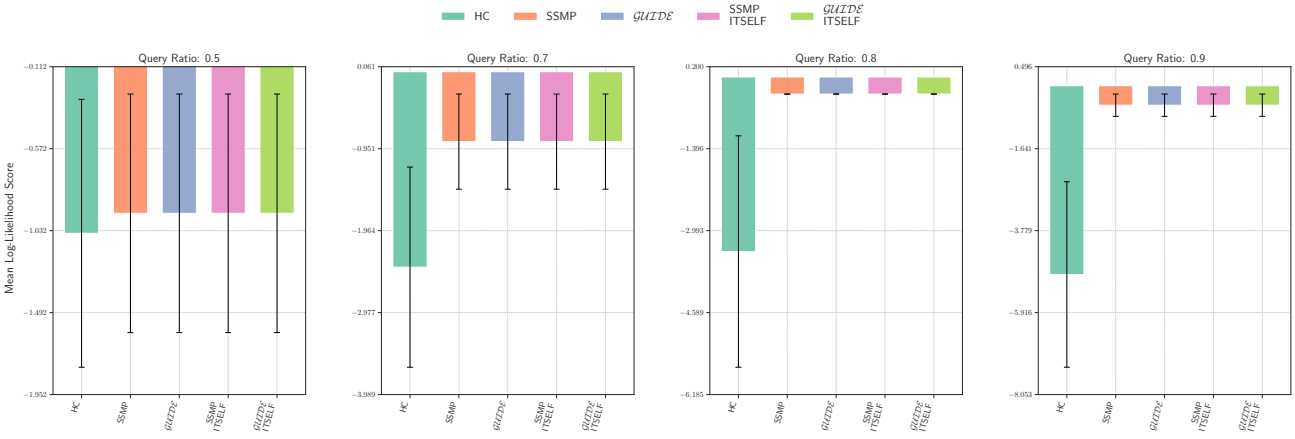


Figure 83: Log-Likelihood Scores on KDDCup2k for NAM. Higher Scores Indicate Better Performance.

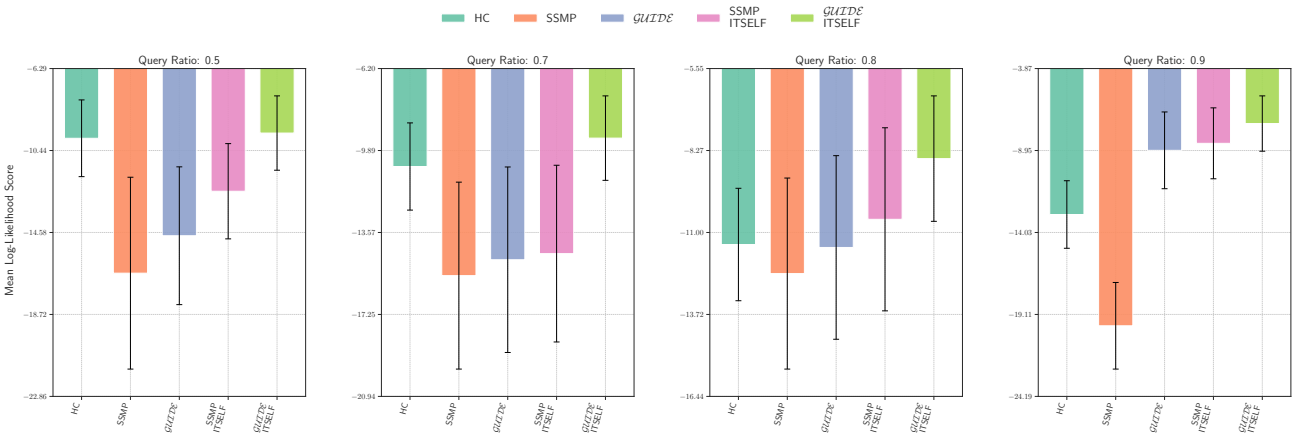


Figure 84: Log-Likelihood Scores on Plants for NAM. Higher Scores Indicate Better Performance.

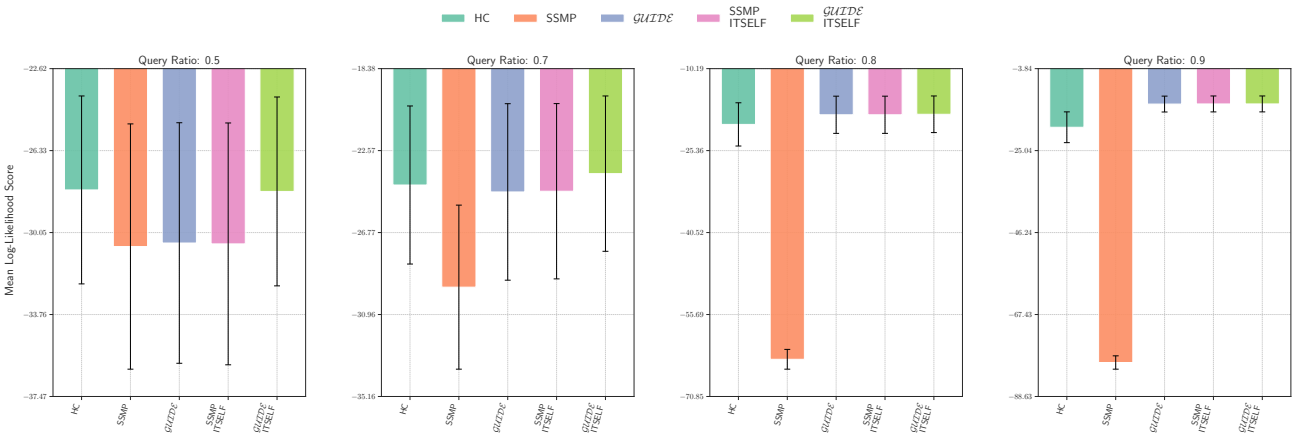


Figure 85: Log-Likelihood Scores on Audio for NAM. Higher Scores Indicate Better Performance.

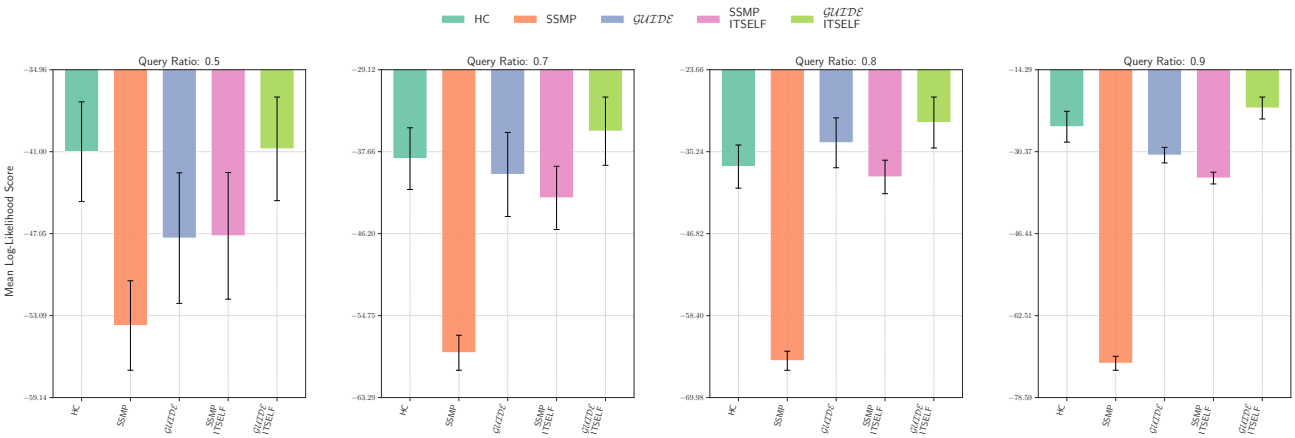


Figure 86: Log-Likelihood Scores on Jester for NAM. Higher Scores Indicate Better Performance.

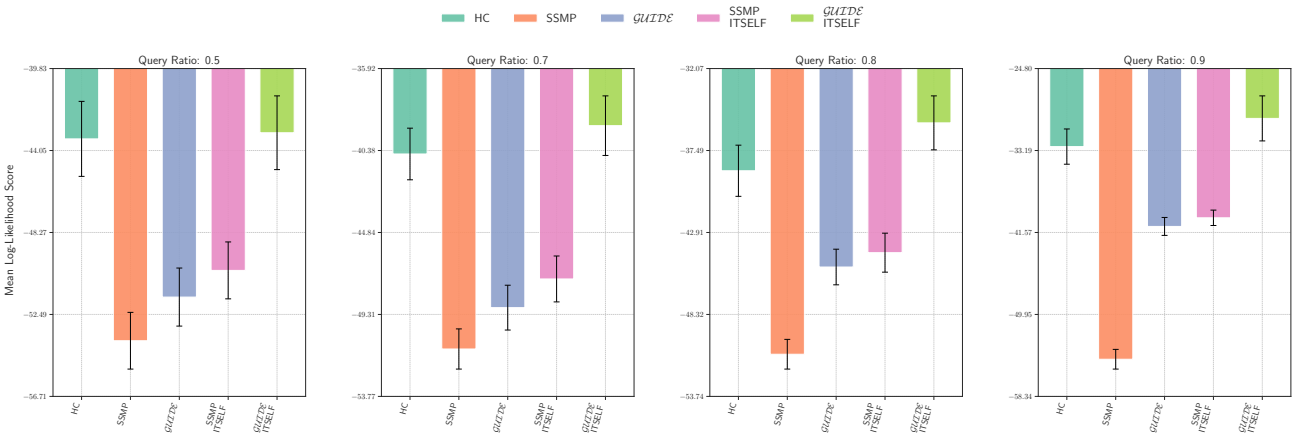


Figure 87: Log-Likelihood Scores on Netflix for NAM. Higher Scores Indicate Better Performance.

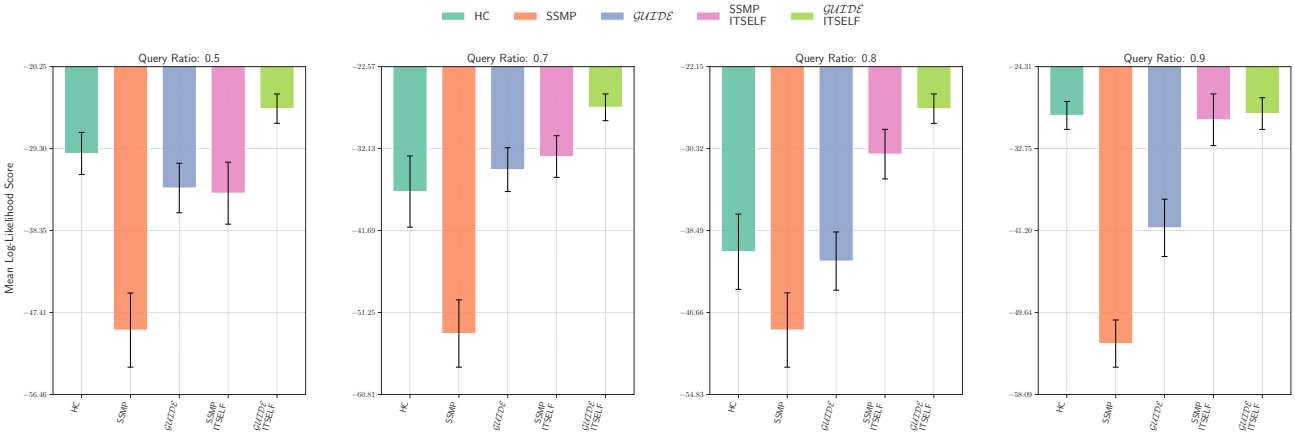


Figure 88: Log-Likelihood Scores on Accidents for NAM. Higher Scores Indicate Better Performance.

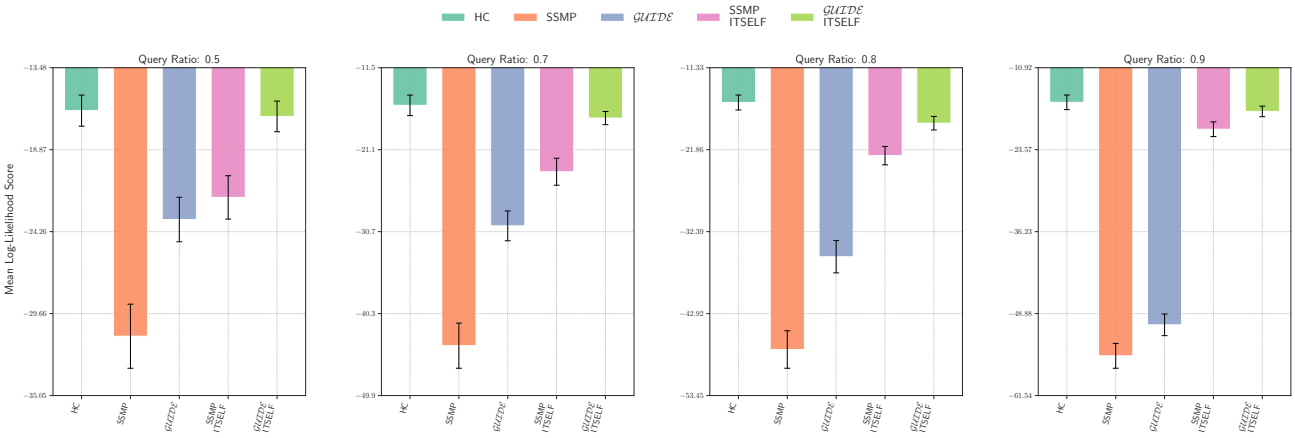


Figure 89: Log-Likelihood Scores on Mushrooms for NAM. Higher Scores Indicate Better Performance.

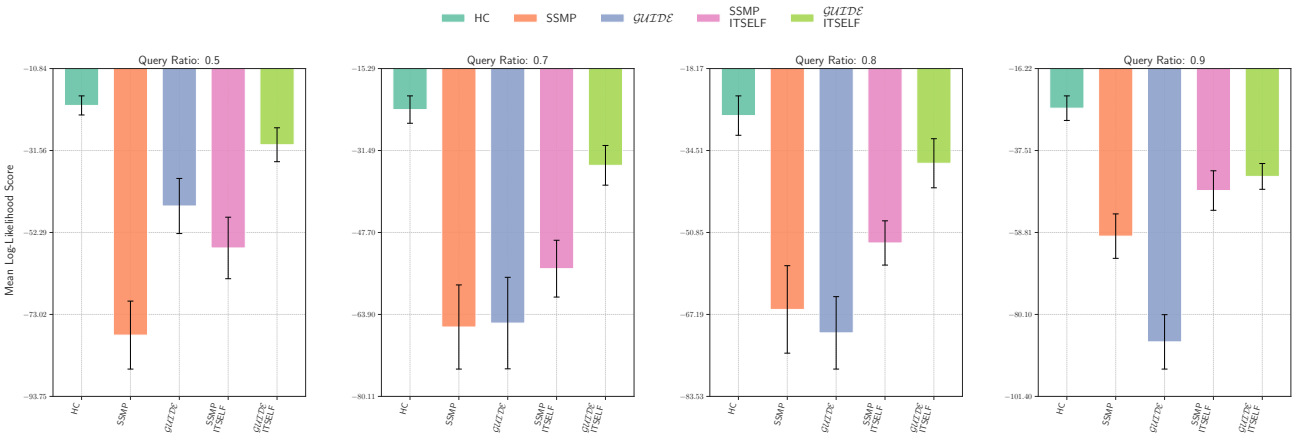


Figure 90: Log-Likelihood Scores on Connect 4 for NAM. Higher Scores Indicate Better Performance.

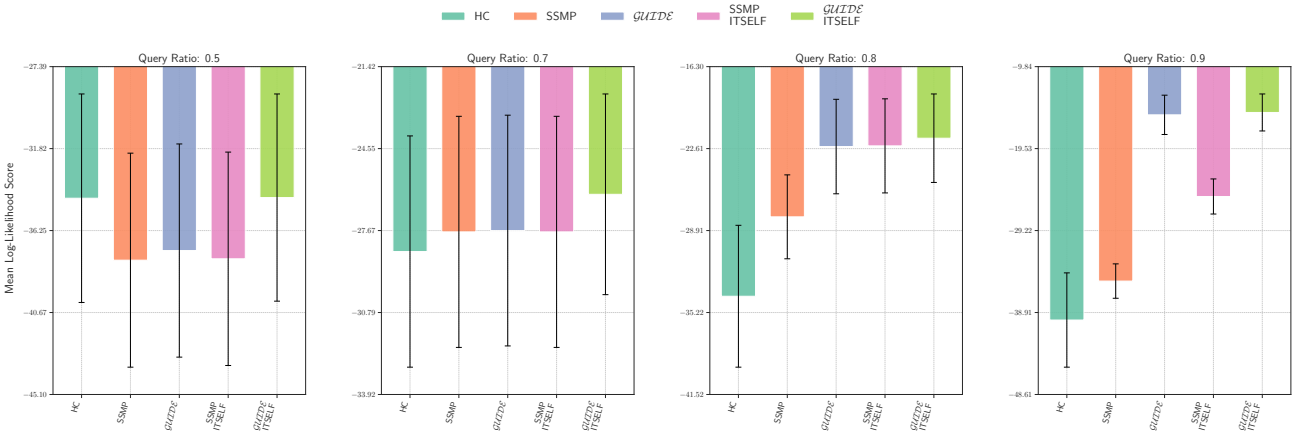


Figure 91: Log-Likelihood Scores on RCV-1 for NAM. Higher Scores Indicate Better Performance.

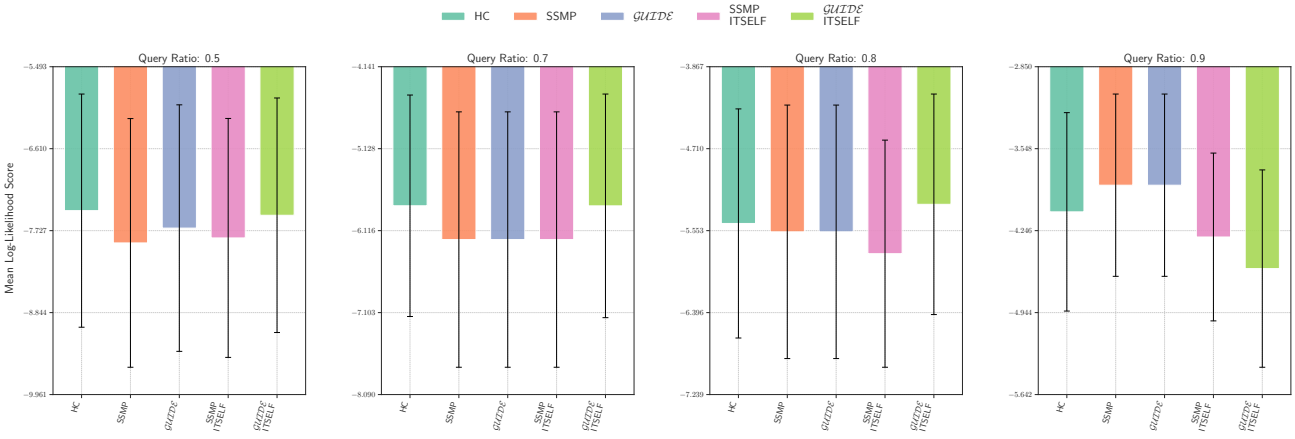


Figure 92: Log-Likelihood Scores on Retail for NAM. Higher Scores Indicate Better Performance.

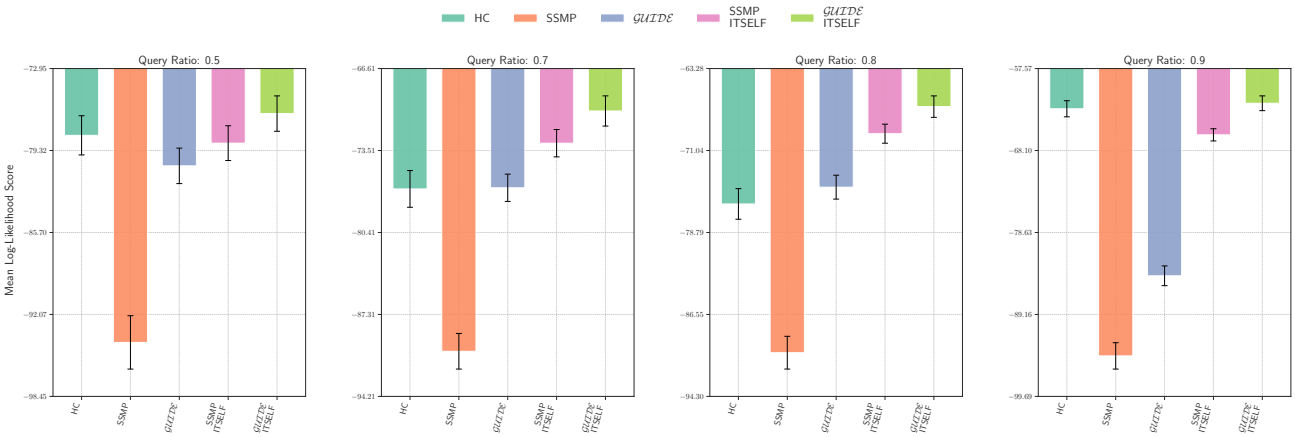


Figure 93: Log-Likelihood Scores on DNA for NAM. Higher Scores Indicate Better Performance.

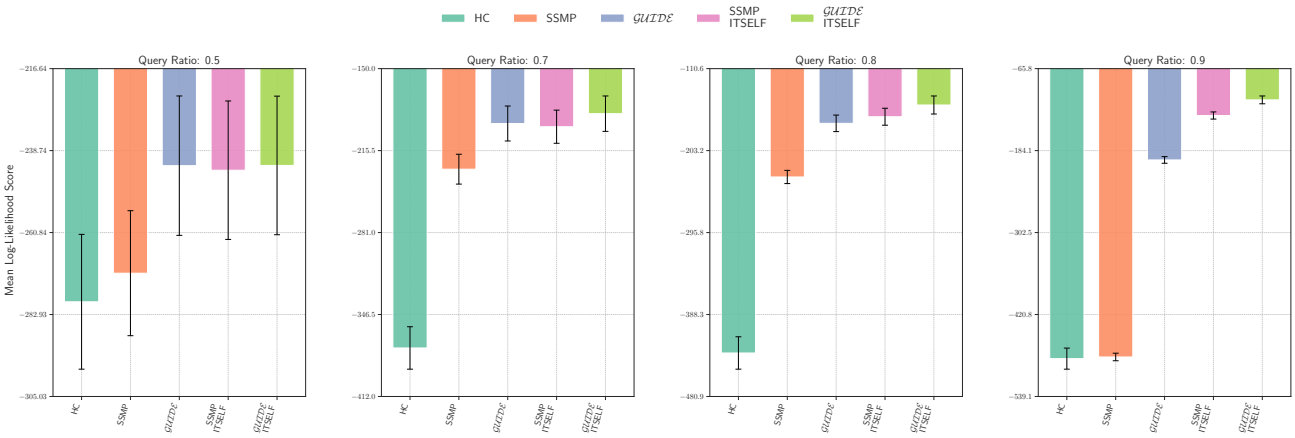


Figure 94: Log-Likelihood Scores on Movie reviews for NAM. Higher Scores Indicate Better Performance.

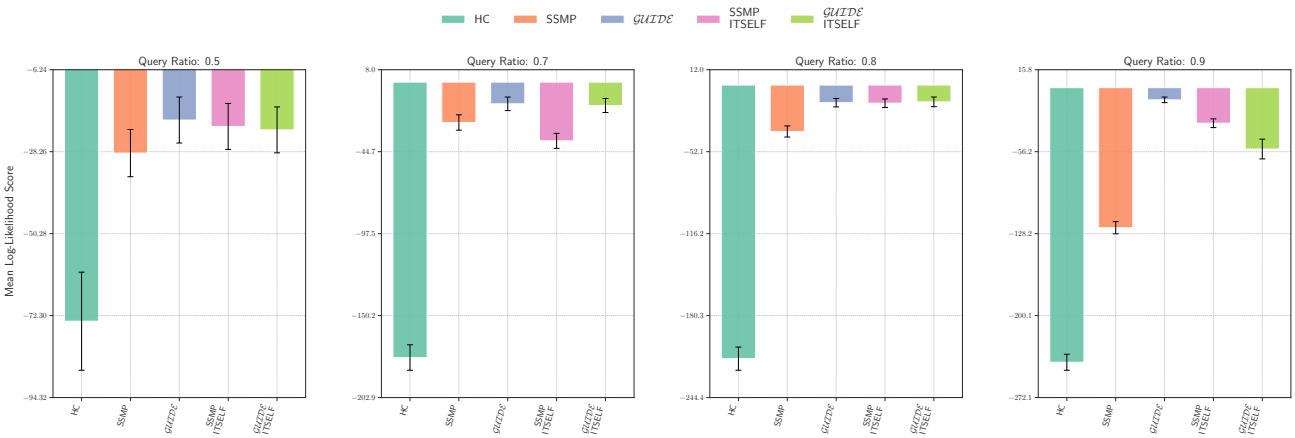


Figure 95: Log-Likelihood Scores on Book for NAM. Higher Scores Indicate Better Performance.

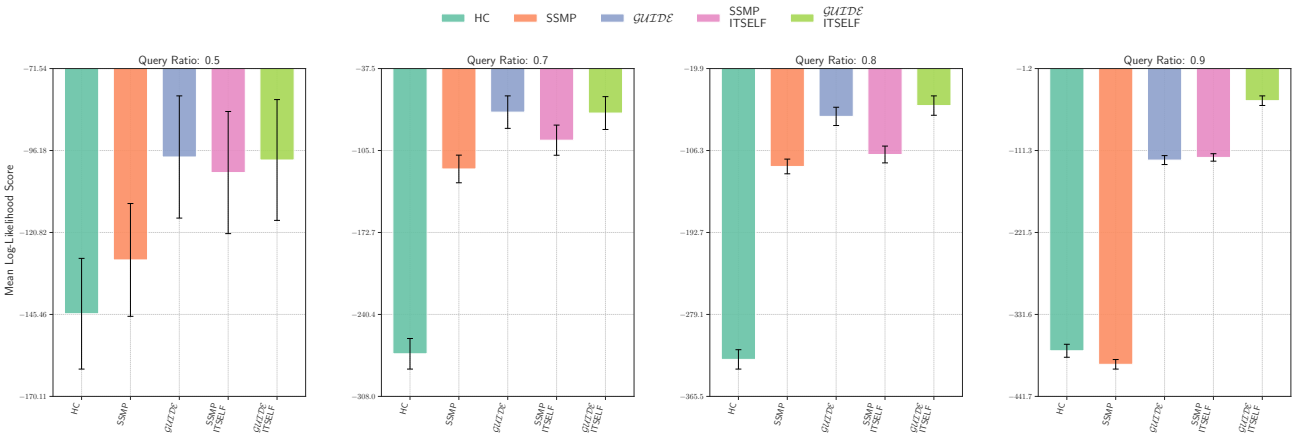


Figure 96: Log-Likelihood Scores on WebKB for NAM. Higher Scores Indicate Better Performance.

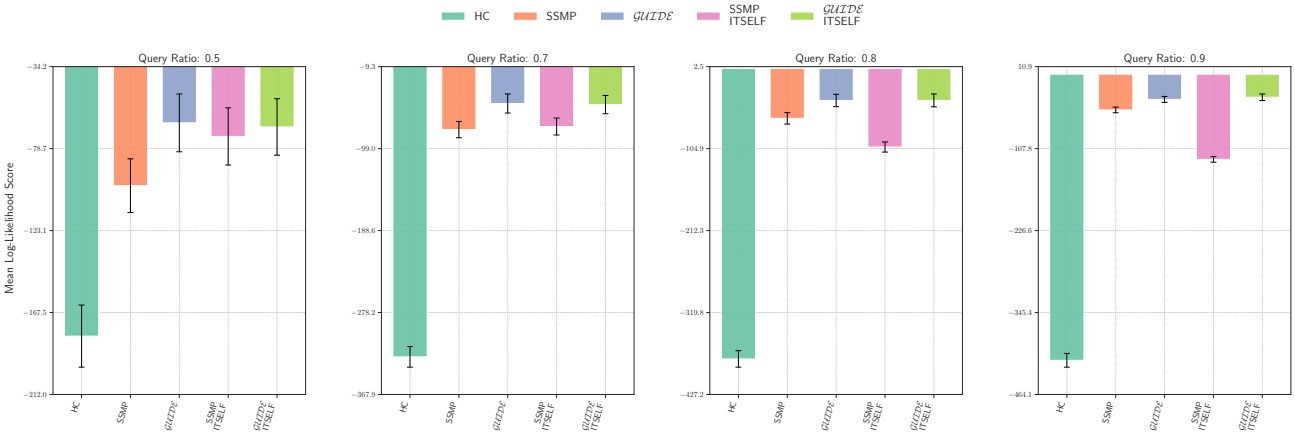


Figure 97: Log-Likelihood Scores on Reuters-52 for NAM. Higher Scores Indicate Better Performance.

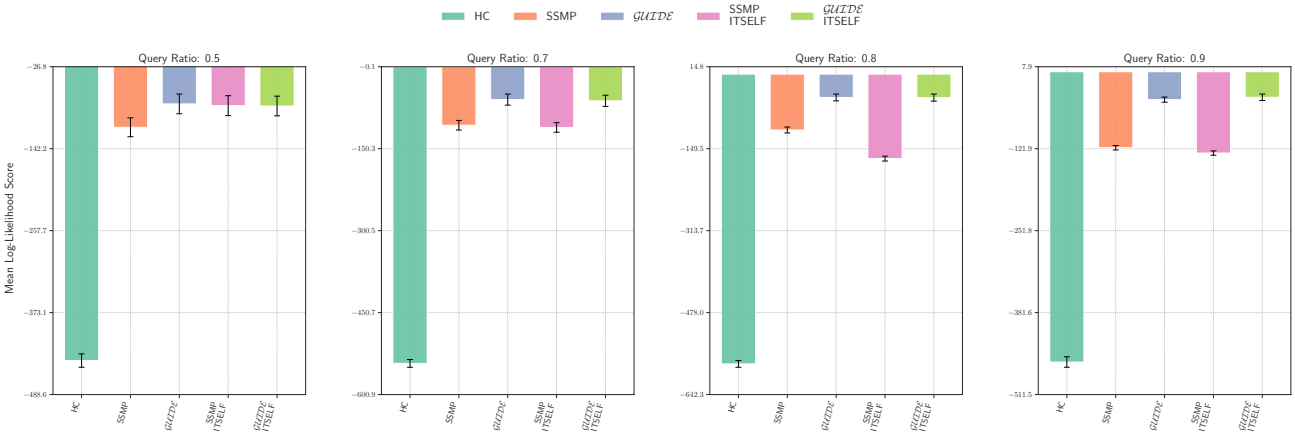


Figure 98: Log-Likelihood Scores on 20 NewsGroup for NAM. Higher Scores Indicate Better Performance.

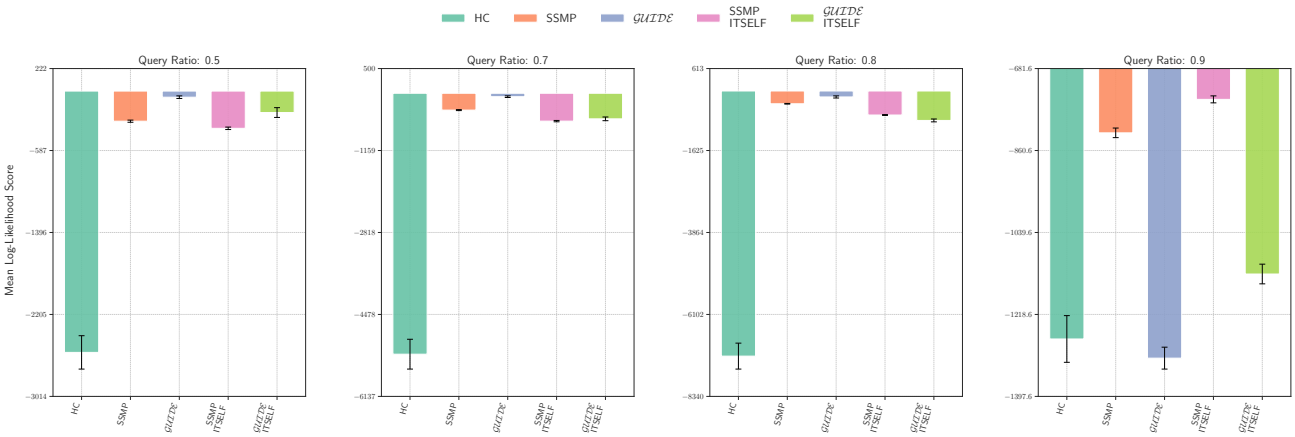


Figure 99: Log-Likelihood Scores on Ad for NAM. Higher Scores Indicate Better Performance.

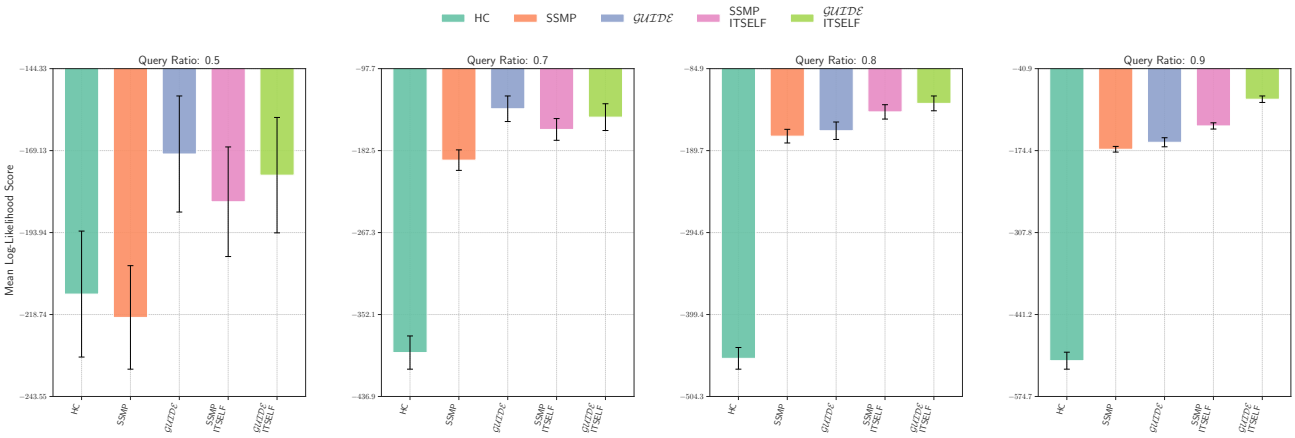


Figure 100: Log-Likelihood Scores on BBC for NAM. Higher Scores Indicate Better Performance.

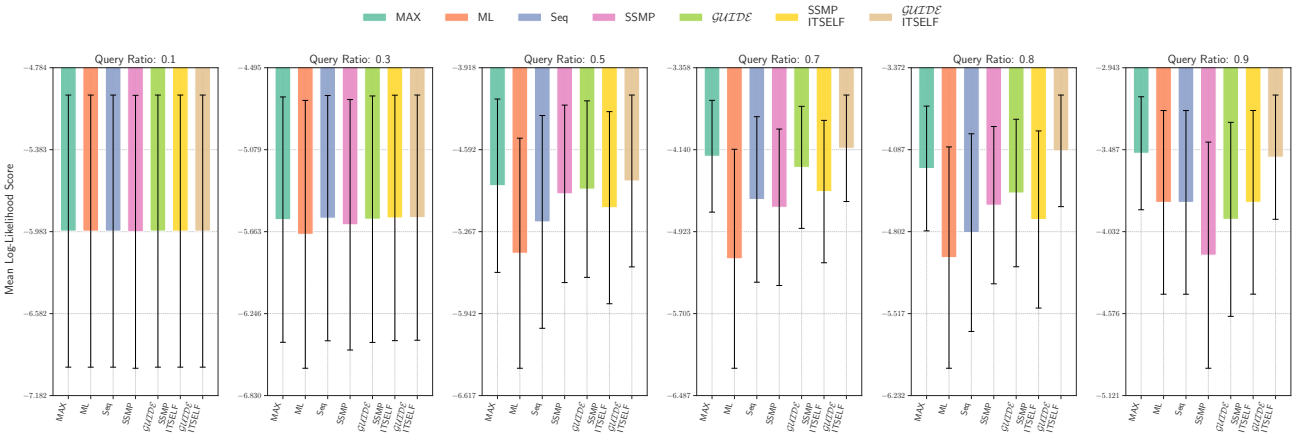


Figure 101: Log-Likelihood Scores on NLTCS for PCs. Higher Scores Indicate Better Performance.

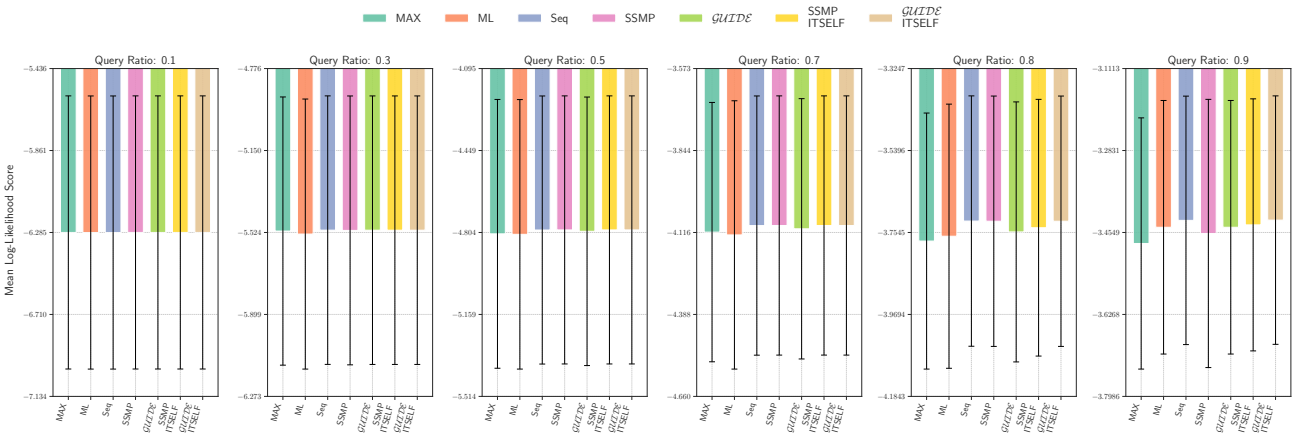


Figure 102: Log-Likelihood Scores on for PCs. Higher Scores Indicate Better Performance.

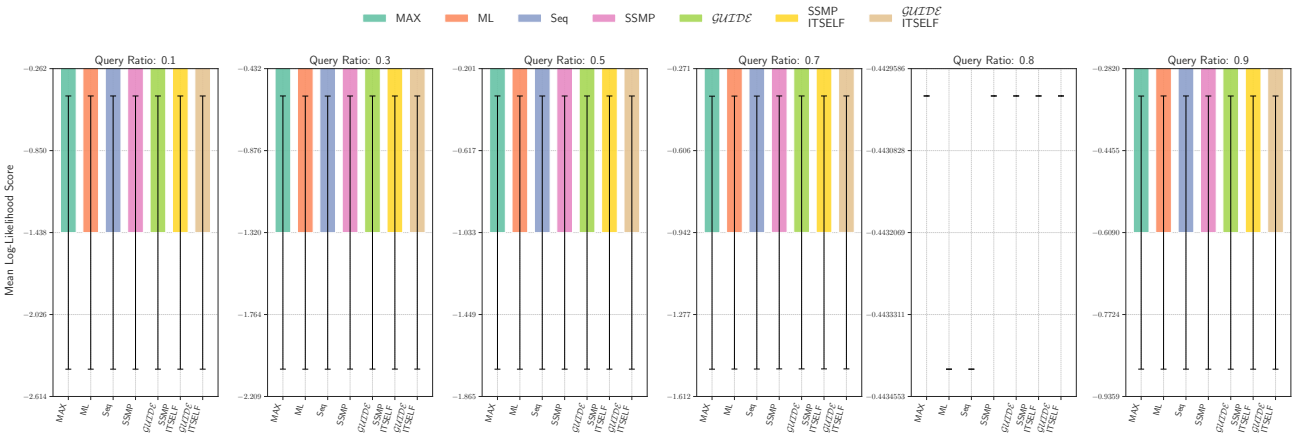


Figure 103: Log-Likelihood Scores on KDDCup2k for PCs. Higher Scores Indicate Better Performance.

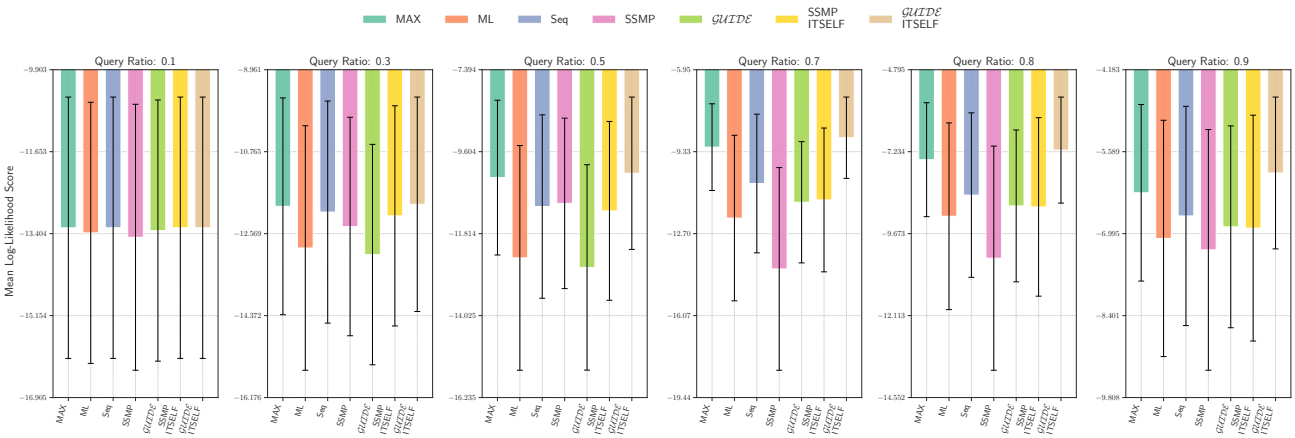


Figure 104: Log-Likelihood Scores on Plants for PCs. Higher Scores Indicate Better Performance.

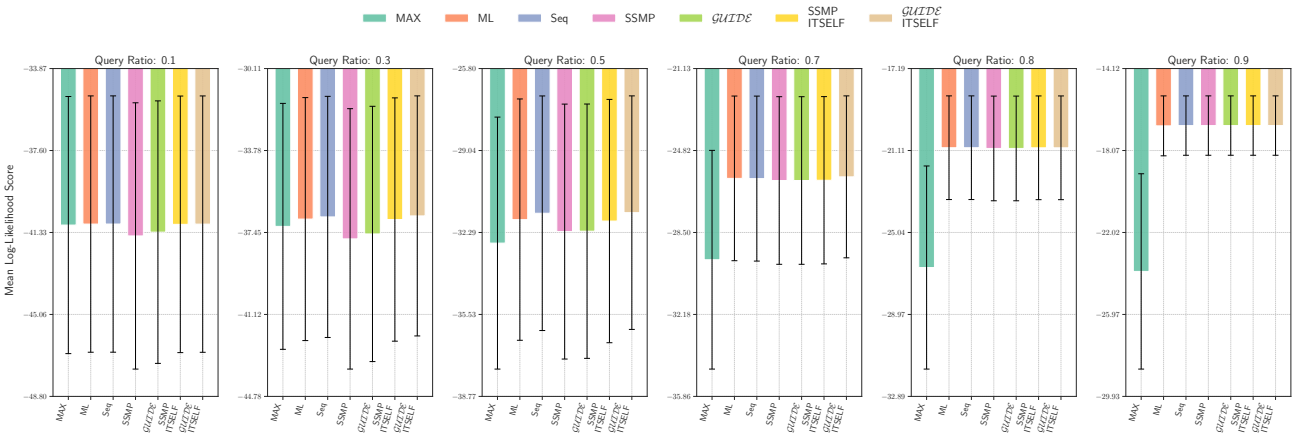


Figure 105: Log-Likelihood Scores on Audio for PCs. Higher Scores Indicate Better Performance.

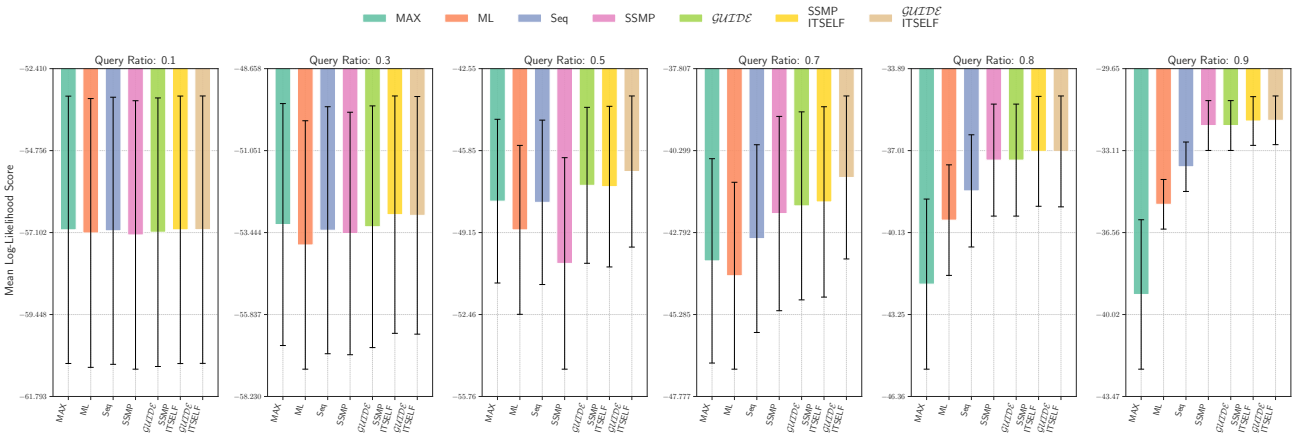


Figure 106: Log-Likelihood Scores on Jester for PCs. Higher Scores Indicate Better Performance.

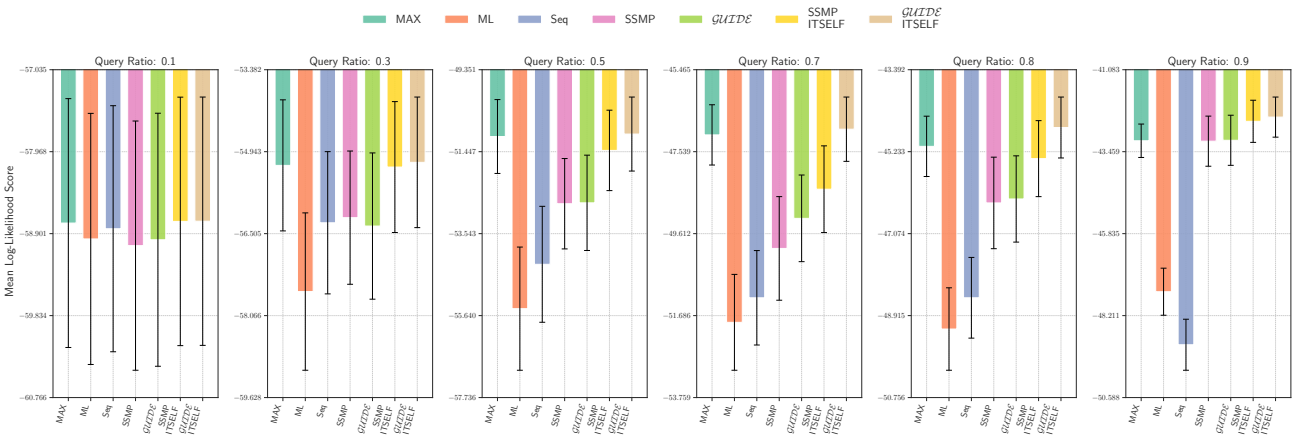


Figure 107: Log-Likelihood Scores on Netflix for PCs. Higher Scores Indicate Better Performance.

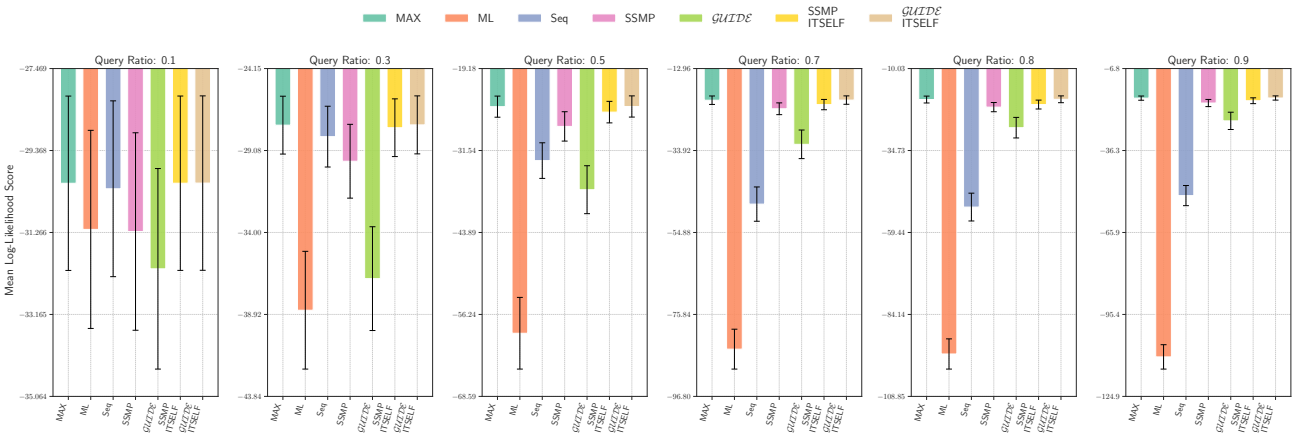


Figure 108: Log-Likelihood Scores on Accidents for PCs. Higher Scores Indicate Better Performance.

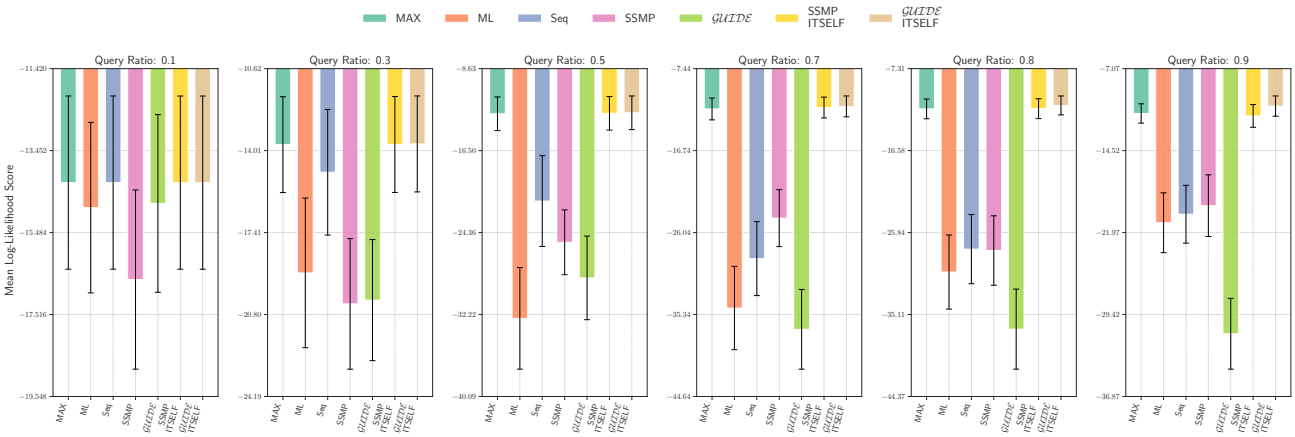


Figure 109: Log-Likelihood Scores on Mushrooms for PCs. Higher Scores Indicate Better Performance.

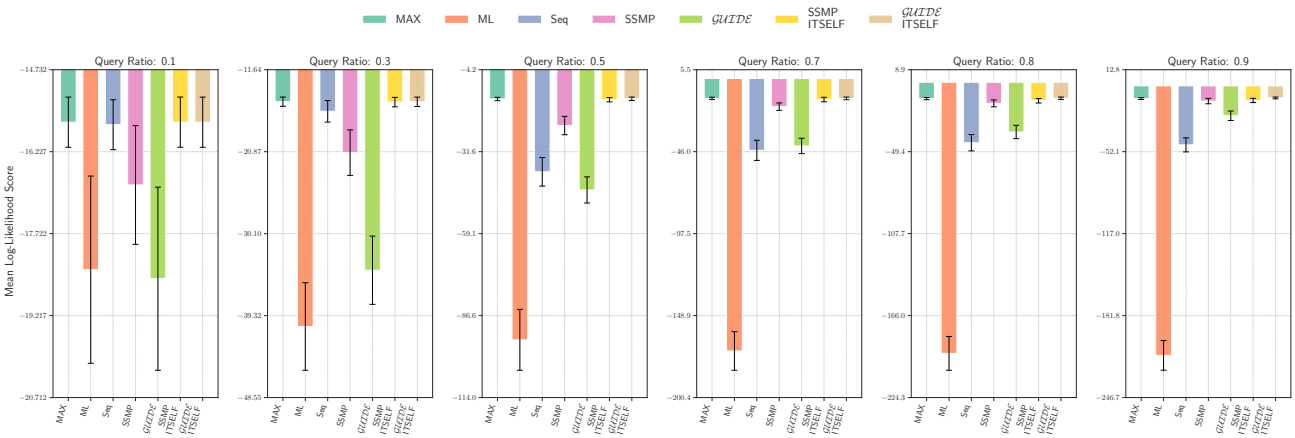


Figure 110: Log-Likelihood Scores on Connect 4 for PCs. Higher Scores Indicate Better Performance.

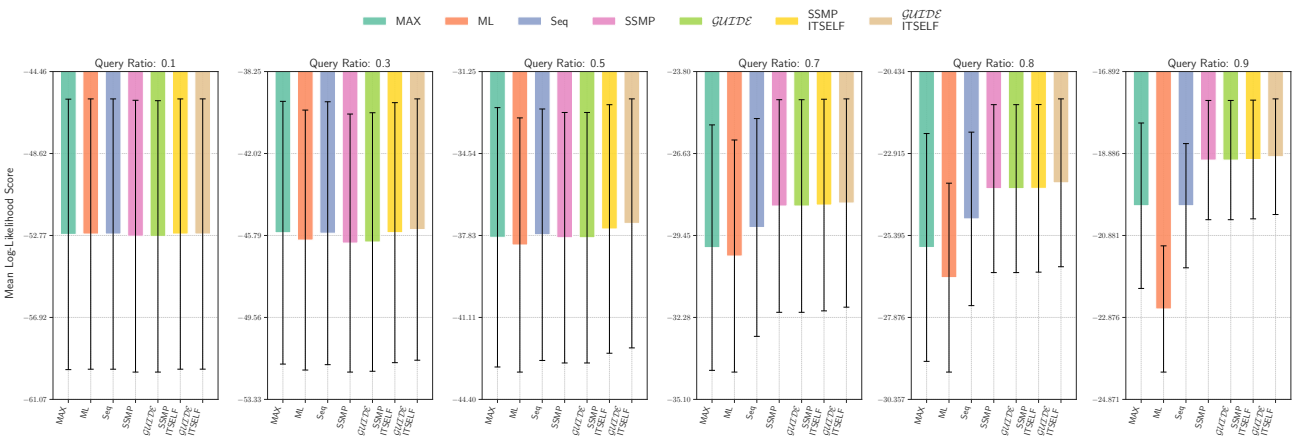


Figure 111: Log-Likelihood Scores on RCV-1 for PCs. Higher Scores Indicate Better Performance.

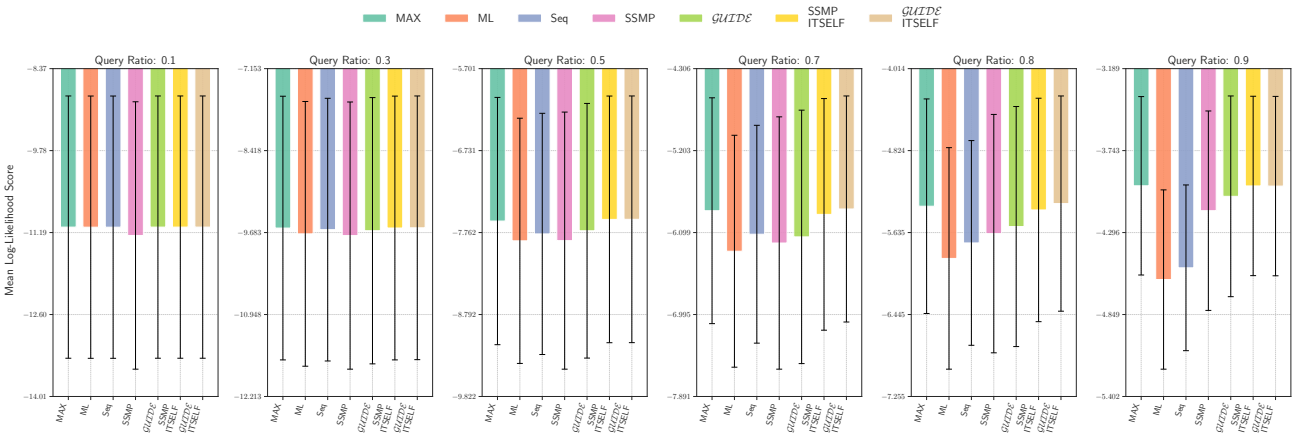


Figure 112: Log-Likelihood Scores on Retail for PCs. Higher Scores Indicate Better Performance.

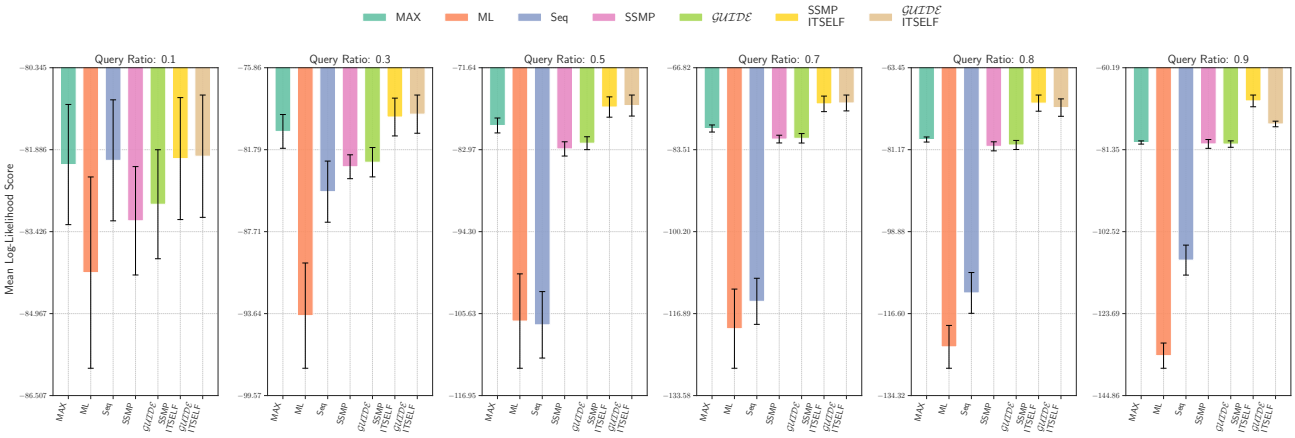


Figure 113: Log-Likelihood Scores on DNA for PCs. Higher Scores Indicate Better Performance.

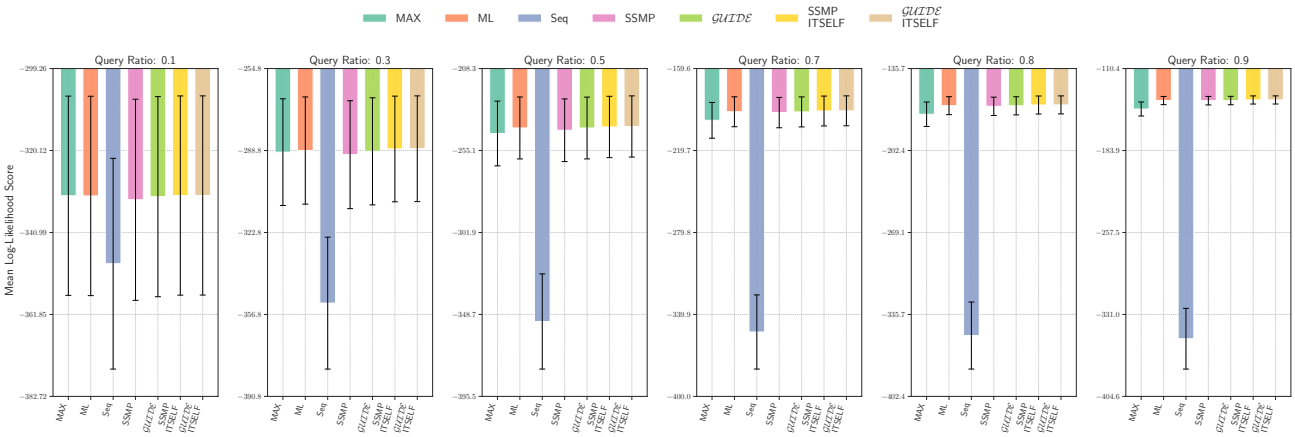


Figure 114: Log-Likelihood Scores on Movie reviews for PCs. Higher Scores Indicate Better Performance.

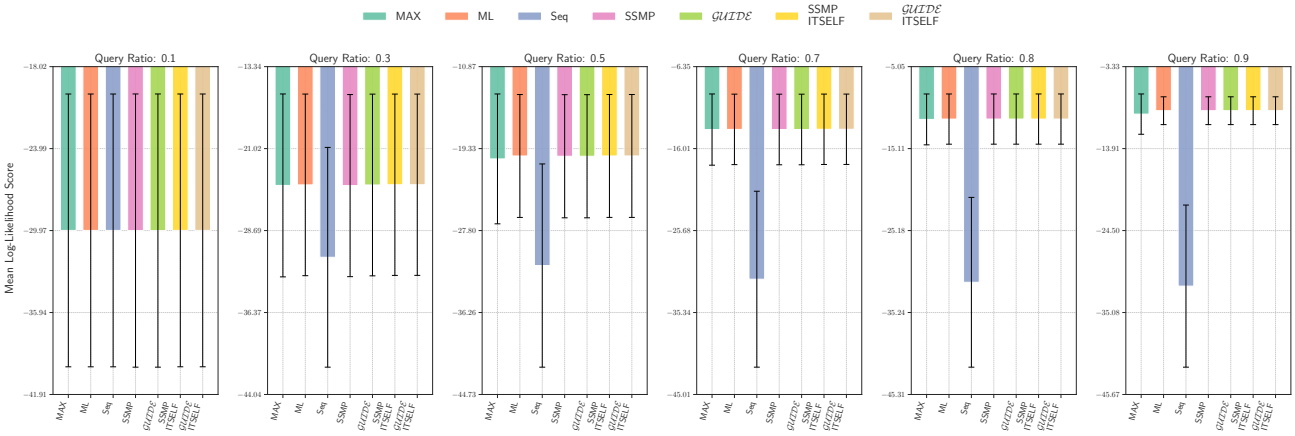


Figure 115: Log-Likelihood Scores on Book for PCs. Higher Scores Indicate Better Performance.

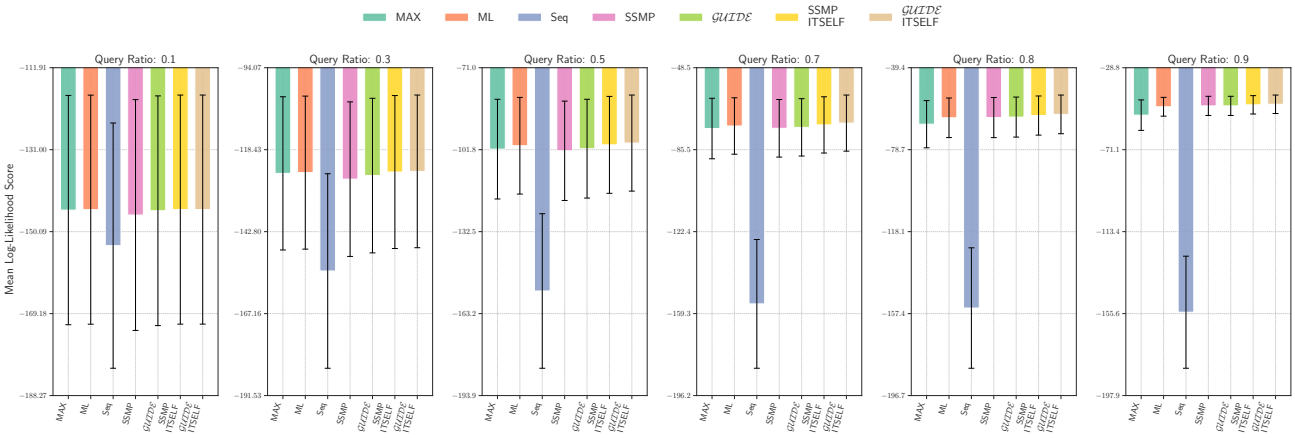


Figure 116: Log-Likelihood Scores on WebKB for PCs. Higher Scores Indicate Better Performance.

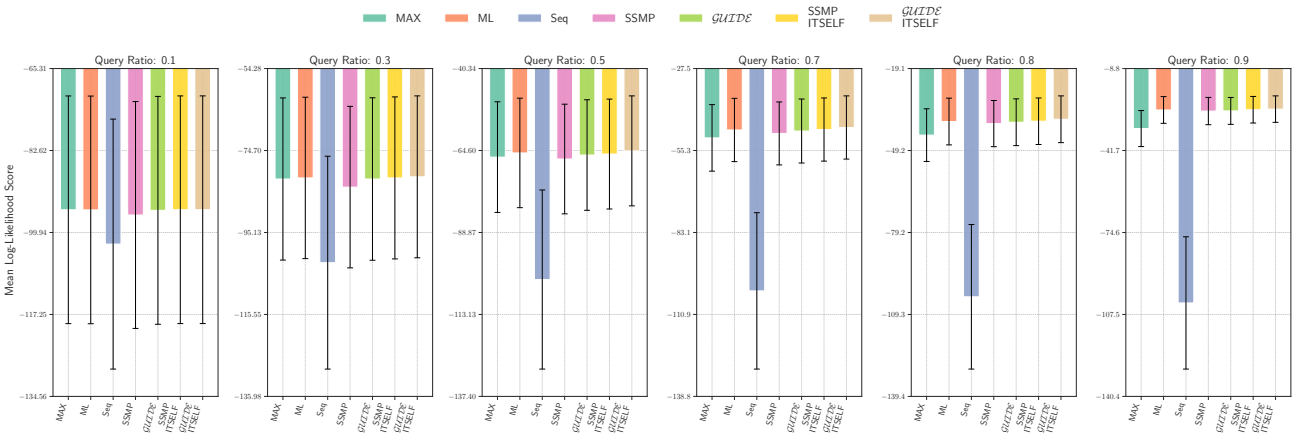


Figure 117: Log-Likelihood Scores on Reuters-52 for PCs. Higher Scores Indicate Better Performance.

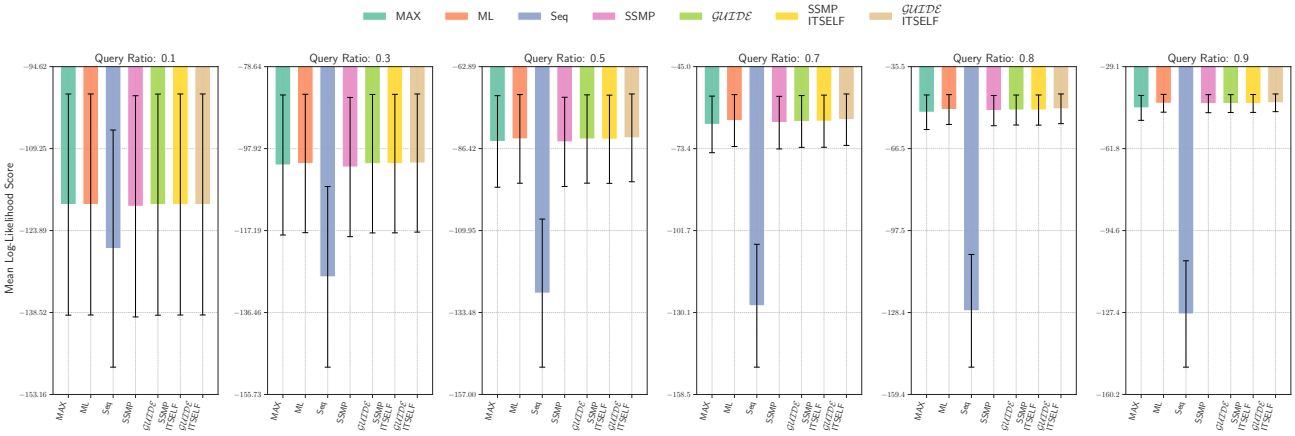


Figure 118: Log-Likelihood Scores on 20 NewsGroup for PCs. Higher Scores Indicate Better Performance.

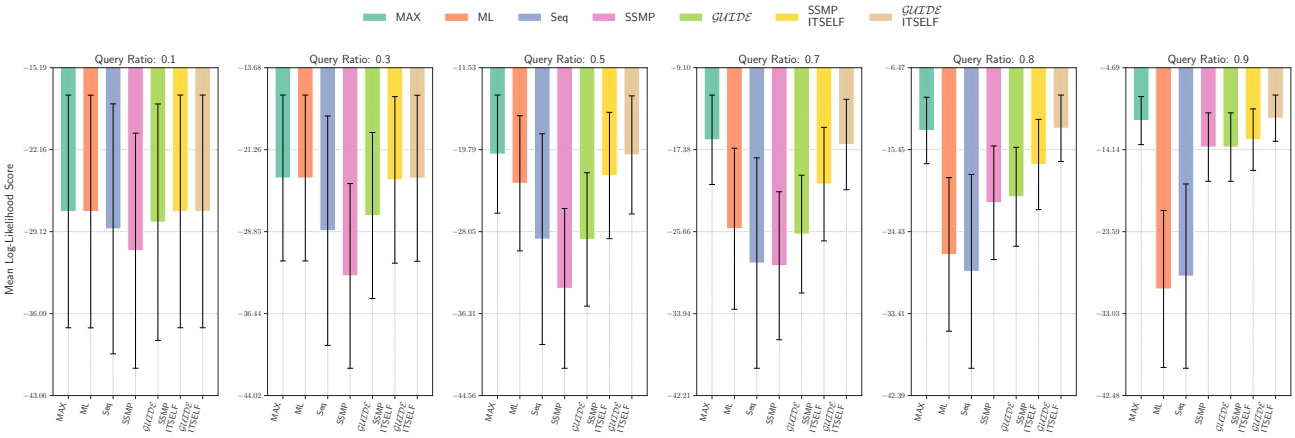


Figure 119: Log-Likelihood Scores on Ad for PCs. Higher Scores Indicate Better Performance.

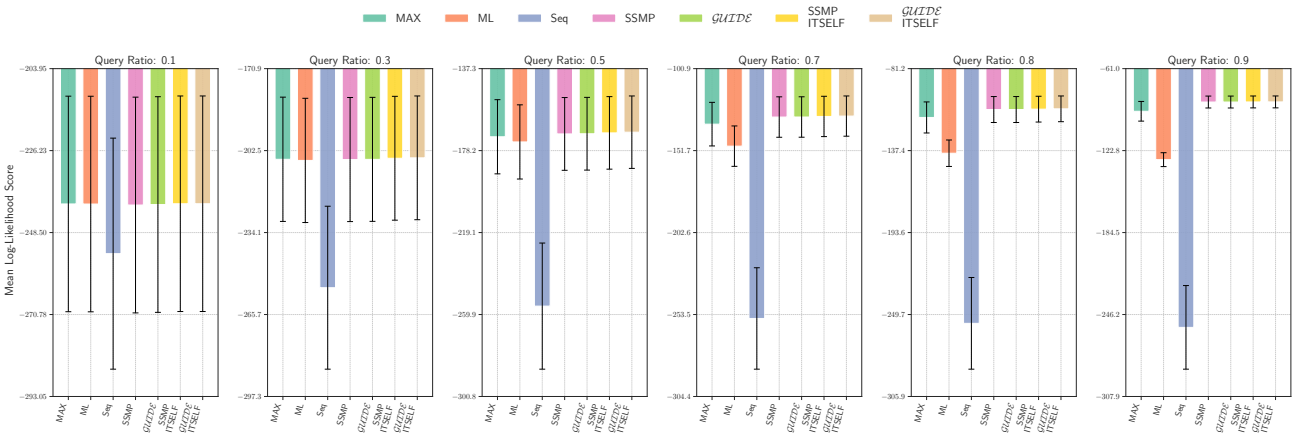


Figure 120: Log-Likelihood Scores on BBC for PCs. Higher Scores Indicate Better Performance.