PEIR: Modeling Performance in Neural Information Retrieval

Pooya Khandel[1(✉)], Andrew Yates[1,2], Ana-Lucia Varbanescu[3], Maarten de Rijke[1], and Andy Pimentel[1]

[1] University of Amsterdam, Amsterdam, The Netherlands
{p.khandel,m.derijke,a.d.pimentel}@uva.nl, andrew.yates@jhu.edu
[2] Johns Hopkins University, Baltimore, USA
[3] University of Twente, Enschede, The Netherlands
a.l.varbanescu@utwente.nl

Abstract. The efficiency of neural information retrieval methods is primarily evaluated by measuring query latency. In practice, measuring latency is highly tied to hardware configurations and requires extensive computational resources. Given the rapid introduction of retrieval models, achieving an overall comparison of their efficiency is challenging. In this paper, we introduce PEIR, a framework for hardware-independent efficiency measurements in Learned Sparse Retrieval (LSR). By employing performance modeling approaches from high-performance computing, we derive performance models for query evaluation approaches such as `BlockMax-MaxScore` (BMM) and propose to measure memory and/or floating-point operations while performing retrieval on input queries. We demonstrate that by using PEIR, similar conclusions on comparing the latency of retrieval models are obtained.

Keywords: Efficiency · Latency · Learned Sparse Retrieval · Performance Modelling

1 Introduction

Efficiency of methods in neural information retrieval (IR) has been chiefly investigated through measuring query latency [10,16] since previous studies show that slow search engines are detrimental to the user experience [27]. In contrast to effectiveness, due to high variety of possible configurations in models, software libraries and hardware, measuring efficiency in neural search is highly challenging [2]. The primary focus in the literature is to build a platform that allows for benchmarking of methods in a unified approach [7,15]. The main disadvantage of this approach is the need for extensive computational resources [2]. Yet, an important cause of challenges in efficiency evaluations is a high degree of hardware involvement and since there are many possible hardware configurations, efficiency comparisons between various methods are non-trivial.

In this paper, we introduce PEIR, a framework based on performance modeling to address the main challenges in measuring efficiency of LSR approaches.

C. Hauff et al. (Eds.): ECIR 2025, LNCS 15573, pp. 279–294, 2025.
https://doi.org/10.1007/978-3-031-88711-6_18

PEIR enables the construction of performance models that can act as proxy for hardware-agnostic efficiency analysis of retrieval methods. Employing PEIR is particularly beneficial since it allows for: (i) standardized efficiency comparisons of retrieval methods, (ii) reducing efficiency measurement dependency on hardware configurations, (iii) reducing the required computational resources for evaluating efficiency by integration of PEIR in existing IR benchmarking frameworks such as TIREX [7], and (iv) accelerating further research and development of new models as it removes the need to evaluate all previous models in terms of efficiency.

The more *work* an algorithm has to do to be completed, the more time taken and energy is consumed, though this relation is not necessarily linear. Regardless of what an algorithm does in practice, its work can be seen as a combination of *memory* and *floating-point* operations at its essence. This principle is the chief idea behind PEIR. We model the *performance* of the retrieval stage given a processed query and a pre-built index, i.e., to count how many operations are executed when relevant documents are retrieved given an input query. The number of operations in an algorithm does not vary by changes in hardware configurations, so making a single time measurement is sufficient. They can be an alternative to current efficiency evaluation metrics, in particular latency.

Performance models help with understanding the cost of all operations for different segments, *factors*, of an algorithm, and how frequent each segment is repeated, we denote them as *coefficients*. Performance models allow us to distinguish between three fundamental levels in efficiency analysis: (L1) algorithm definition; this levels associates with how theoretically efficient an algorithm is. (L2) algorithm implementation; at this level the efficiency is affected by the programming language and how well the algorithm is programmed. (L3) algorithm execution; this level concerns the hardware configuration such as processor choice (CPU or GPU) or use of parallelism. PEIR is built upon efficiency analysis in the second level (L2). We illustrate how applying PEIR allows for constructing performance models to compare the retrieval methods efficiency and detail the process to perform efficiency analysis in the final level (L3) for latency evaluation.

As a case-study, we focus on efficiency evaluation of learned sparse retrieval methods using the PISA [18] library. We take the `BlockMax-MaxScore` (BMM) [3] algorithm and derive performance models for it to calculate the number of memory and floating-point operations using MS MARCO v1 [1] evaluation queries. Further, we develop latency models and validate that efficiency evaluation by calculating the number of operations through the proposed performance models is robust and aligns closely with the findings from measuring the latency. In addition, we empirically show that latency models built based on the proposed performance models can be used to predict query latency based on the experimental results obtained from two different clusters with AMD and Intel CPUs. The contributions of our work are as follows:

(C1) We introduce PEIR, a framework that allows for building performance models for hardware-agnostic efficiency evaluation of retrieval methods. To

the best of our knowledge, we are the first to apply performance models in neural IR to address challenges in efficiency evaluation.

(C2) We validate the applicability of our proposed framework by deriving performance models for the BMM algorithm with the PISA library for learned sparse retrieval methods. We empirically demonstrate that comparing efficiency of retrieval methods with our proposed approach highly correlates with measuring latency across two different hardware platforms.

(C3) The repository including our instrumented PISA library and empirical analysis is publicly available at: https://github.com/po-oya/peir.

2 Background – Learned Sparse Retrieval (LSR)

In this section, we explain the operation of a Learned Sparse Retrieval (LSR) pipeline consisting of training and retrieval stages. LSR models may have different training recipes, as they may use, for example, different expansion schemes or optimizations, but they eventually encode queries and documents in sparse vectors of vocabulary size [21]. At the end of training stage, an inverted index of documents is created that is used in the retrieval stage. In the retrieval stage, we employ a query evaluation algorithm that, given an encoded input query and an inverted index, retrieves the most relevant documents from the collection as fast as possible. This query evaluation algorithm performs several optimizations, including early-exiting strategies, to decrease the retrieval latency with minor degradation in performance. These optimizations eventually make the retrieval latency variable per query.

Efficient LSR implies low-latency queries. The total query latency is the sum of query encoding latency and retrieval latency. Determining query encoding (processing) latency is less of a problem because the time to process a query mainly depends on the model size. In contrast, retrieval latency is highly affected by term weight distributions [16]. In this work, we focus on retrieval latency estimation, as the most expensive and least predictable of the pair.

3 Related Work

We identify three types of related work: efficiency metrics definition and assessment, efficiency measurements, and performance modeling.

Efficiency Metrics. Since users experience with search systems is affected by retrieval speed, i.e. how fast retrieved documents are presented, [27], latency is the most widely used efficiency metric in IR [2,25]. Achieving low latency is especially challenging given the large collections of documents and big retrieval models. Other work (particularly in the context of SPLADE models [6]) proposes to assess efficiency based on FLOPS, defined as the average number of floating-point operations between a query and a document [4]. However, these estimates are not good efficiency indicators because they consider the worst-case scenario where all query-document pairs are scored [4–6,10] - the very behaviour that LSR

pipelines optimize with techniques like early exit. Other metrics that have been proposed include the index size [10,16], and, more recently, measuring energy consumption [26], CO2 footprint [28]. In this work, we consider low-latency as the key indicator of efficiency for a LSR method.

Efficiency Benchmarking. There exist diverse benchmarking attempts for efficiency in the context of IR. For example, Hofstatter and Hanbury [9] measure and report throughput and latency, in addition to comparing models accuracy. Based on the Dynaboard [15] method, Santhanam et al. [25] combine several efficiency metrics to compare models together. Fröbe et al. [7] introduce TIREX, a standardized environment to run and evaluate models effectiveness and efficiency on a diversity of platforms. Our work is complementary to these approaches, because we *estimate* efficiency in a hardware-agnostic, model-based manner.

Modeling. Performance models are employed to understand and analyze the performance of various applications [20], typically expressed in terms of latency or throughput. Moreover, they help with exploring the applications performance bottlenecks [22,29] through profiling hardware performance counters. Particularly, Li et al. [11] and Lym et al. [14] successfully apply performance models to detect bottlenecks and predict latency of Convolutional Neural Networks (CNN). However, to the best of our knowledge, we propose the first performance modeling approach for the retrieval stage of LSR pipeline *and* we demonstrate how the model can be used successfully as proxy for latency comparison for different LSR methods.

4 Performance Modeling Principles

We detail our approach to performance modeling for latency estimation.

4.1 Performance Model $\mathcal{P}$

Consider the retrieval stage explained in Sect. 2. Although different query evaluation methods $\mathcal{R}$ have different complexity, evaluating a query ultimately translates into performing *memory operations* (Mop) and/or *floating-point operations* (Flop) to produce the list of retrieved documents $\mathcal{L}$. Memory operations are reading and writing (R/W) data from/to the memory hierarchy; addition and multiplication are examples of floating-point operations. The overall execution time depends on the number of memory and floating point operations, which in turn depend on the implementation of $\mathcal{R}$, and the latency of these operations, which in turn depends on the hardware specifications.

We define analytical performance models to estimate the number of different operations of a given implementation, which ultimately correlates to the expected execution time. Assuming f_i is the number of times an operation repeats in an algorithm, $\mathcal{P}_{\mathcal{R}}(f_i\text{Mop}_i, f_i\text{Flop}_i), \forall i$ is the performance model for $\mathcal{R}$. Given the input for the retrieval task, i.e., a query and an index, the output of this $\mathcal{P}_{\mathcal{R}}$ would be a number of operations; however, the model can be further adjusted to estimate latency or energy consumption.

To construct a performance model for retrieval in the context of learned-sparse methods, we make the following simplifications: (i) We assume **no parallelism** is used for the implementation of $\mathcal{R}$; we observe all efficiency evaluations are conducted in a single-core setting. Therefore, we can safely assume operations are executed sequentially. This will simplify the model and increase its interpretability, although it may be a source of error in model accuracy due to optimizations within the CPU. (ii) We do not distinguish between R/W operations within memory and cache(s). This is also an assumption in favor of interpretability, but it may also be a source of inaccuracy, especially for algorithms optimized for cache usage. The assumption can be removed for relevant cases, meaning additional parameters are needed for the performance model. (iii) We do not differentiate, in terms of latency, between different types of floating point operations (i.e., a sum and a multiplication are both one operation).

4.2 Constructing $\mathcal{P}$

Assuming an algorithm with i phases (i.e., code blocks or segments), we define our generic performance model, $\mathcal{P}$, as a *symbolic work model* where the contributions of both memory and floating-point operations are combined (hence the $\oplus$ operator). Such a generic model is presented in Eq. (1):

$$\mathcal{P} = \sum_i f_i \cdot (\text{Mop}_i \oplus \text{Flop}_i)\,. \tag{1}$$

In an ideal setting, when the computation is independent of the input data, the operation mix (i.e., the number of memory and floating point operations) can be determined by static analysis tools. However, when data-dependent code blocks exist, we need additional steps to collect such data.

We explain these concrete steps to construct such a performance model $\mathcal{P}_\mathcal{E}$ through an example algorithm $\mathcal{E}$, provided in pseudo-code in Algorithm 1. To define $\mathcal{P}_\mathcal{E}$ for algorithm $\mathcal{E}$, we rely on *static analysis* and instrumentation at source-code level (e.g., Algorithm 1). In the manual process, this entails collecting the *relevant data* directly from the code, line by line. For example, initialization of variables and arrays, i.e., lines 1–4. The cost of this initialization is negligible compared to the bulk of processing, and is therefore ignored. Lines 5–10 form the costly part of Algorithm 1. In this region we monitor the frequency of operations by adding the explicit counters *for_cnt* and *if_cnt* to count how many times the for loop and the branch, respectively, are executed. Expanding the model to use the additional data from *for_cnt* and *if_cnt*, Eq. (1) can be rewritten as:

$$\begin{aligned}\mathcal{P}_\mathcal{E} = {} & \mathit{for\_cnt} \cdot ((\text{Mop}_{line\ 6} + \text{Mop}_{line\ 7}) \oplus (\text{Flop}_{line\ 7} + \text{Flop}_{line\ 7})) \\ & + \mathit{if\_cnt} \cdot (\text{Mop}_{line\ 8} \oplus \text{Flop}_{line\ 8})\,.\end{aligned} \tag{2}$$

In line 6, the application performs one memory read and one floating-point operation; the cost for *Func* is one memory read. Finally, the cost of line 8 is one read from memory and one floating point operation. Thus, simplifying the terms in Eq. (2) results in:

$$\mathcal{P}_\mathcal{E} = \underbrace{2 \cdot \mathit{for\_cnt} + 1 \cdot \mathit{if\_cnt}}_{\text{Mop cost}} \ \oplus \ \underbrace{1 \cdot \mathit{for\_cnt} + 1 \cdot \mathit{if\_cnt}}_{\text{Flop cost}}\,. \tag{3}$$

Algorithm 1. Implementation of an example algorithm $\mathcal{E}$

Input: A dummy array A of size N randomly initialized with integers from 0 to $N-1$; A binary randomly initialized array S of size N.

```
1: Initialize an output vector V of size N.
2: for i = 0, 1, ..., N do
3:     V[i] = 0.
4: end for
5: for i = 0, 1, ..., N do          ▷ Instrumentation: update coefficient for_1++
6:     Temp = i * A[i].             ▷ Contributes to Mop and Flop
7:     if FUNC(A[i], S) then        ▷ Instrumentation: update coefficient if_1++
8:         Temp = A[i] * A[i]       ▷ Contributes to Mop and Flop)
9:     end if
10: end for
11: procedure FUNC(loc, S)
12:     if S[loc] == 1 then
13:         return True
14:     end if
15:     return False
16: end procedure
```

In our performance model $\mathcal{P}_{\mathcal{E}}$, we correlate/refer to counters (such as *for_cnt* and *if_cnt*) that indicate the frequency of operations to *coefficients*, and to the values multiplied with them as *factors*. The *factors* depend on the implementation (i.e., they (only) change with variations in implementation), while the *coefficients* are likely to vary based on the input data. E.g., in Algorithm 1, the main loop at line 5 occurs N times, meaning the coefficients $for_1 = N$ and $if_1 \leq N$ depend on the size and initialization of array S, while the factors depend on the actual operations in the implementation.

To determine unknown coefficients, we require a dedicated data collection step. First, we execute the algorithm with sufficient samples of input data, and measure coefficients per sample. Then, we estimate an average for each coefficient from the collected data if the variance is low. In more complex cases, where the relation between input characteristics and coefficient distribution is more complex, more analysis/modeling is needed - for example using regression or more advanced machine learning approaches - to create a representative function that estimates the coefficient sufficiently well.

5 PEIR: Performance Models for Neural IR

Our main goal is to propose a framework for estimating the efficiency of retrieval pipelines based on analytical performance models. This section focuses on the design and implementation of the framework, and its applicability for a given case-study: comparing the efficiency of LSR models with the BMM [3] algorithm. We show how to build a dedicated, hardware-agnostic performance model, $\mathcal{P}_{\mathcal{R}}$, for our use-case. We validate the model and framework further in Sect. 6.

5.1 The PEIR Framework

PEIR focuses on modeling performance of a query evaluation algorithm $\mathcal{R}$ in the retrieval stage that takes a processed query and an inverted index as inputs and produces the list of retrieved documents. Let $\mathcal{P}_{\mathcal{R}}$ be the analytical performance model for $\mathcal{R}$ that determines the total number of operations while executing $\mathcal{R}$. The processed queries and term weight distributions in the index vary across retrieval methods [16]; therefore, various retrieval methods will have a different $\mathcal{P}_{\mathcal{R}}$ w.r.t. coefficients.

Comparing the number of operations with $\mathcal{P}_{\mathcal{R}}$ across retrieval methods is a proxy for comparing their latency in a hardware-agnostic manner.

5.2 $\mathcal{P}_{\mathcal{R}}$ for the `BlockMax-MaxScore` (BMM) Algorithm in `PISA`

To demonstrate the feasibility of our modeling approach, we show how the performance model $\mathcal{P}_{\mathcal{R}}$ is constructed in practice for the BMM algorithm based on the `C++` implementation from the `PISA` [18] library. To derive the coefficients and factors for $\mathcal{P}_{\mathcal{R}}$, the code needs to be inspected. Due to space limitations, we show a simplified snippet of the code in Sect. 5.2.[1] This snippet is from the `block_max_maxscore_query.hpp`[2] file in the PISA library; this code executes the BMM algorithm on an input query.

```
1    void operator()(CursorRange&& cursors, uint64_t max_docid) {
2        // This part of the code is mainly initialization
3        while ( /* Main loop of the algorithm, the most costly part */) {
4            // w_cnt
5            // f1_cnt, f2_cnt, f3_cnt
6            // p1_cnt, ..., p6_cnt
7            // brz1_cnt, brz2_cnt
8        }
9    }
```

The first part of the implementation only includes variable initialization. Our analysis indicates that this initialization is negligible, time-wise, when compared to the *main loop*. Thus, $\mathcal{P}_{\mathcal{R}}$ only captures the main loop. To determine the *coefficients* in $\mathcal{P}_{\mathcal{R}}$, we define counters: w_cnt monitors how many times the `while` loop occurs; $f_i\_cnt, i \in \{1, \ldots, 3\}$ monitor the occurrence of three `for` loops; $p_i\_cnt, i \in \{1, \ldots, 6\}$ monitor the `if` statements' branching; and, finally, brz_i_cnt monitor the two `break` statements in the implementation; no extra operation is executed when break occurs and they do not affect $\mathcal{P}_{\mathcal{R}}$. Each counter x_cnt eventually corresponds to coefficient x in $\mathcal{P}_{\mathcal{R}}$.

To derive the *factors*, we inspected the code and extracted the number of Mop and Flop operations for any extra method called within this main loop. Table 1 presents the results of this inspection.

The symbolic model for $\mathcal{P}_{\mathcal{R}}$ is presented in Eq. (4) (for memory operations) and (5) (for floating-point operations). Note that Mop_{M_i} and Flop_{M_i} denote the

[1] The complete instrumented code available at https://github.com/po-oya/peir.
[2] Look at pisa/include/pisa/query/algorithm/block_max_maxscore_query.hpp.

factors from Table 1.

$$\begin{aligned}\mathrm{Mop}_{(\mathcal{P}_\mathcal{R})} = {} & f_1 \cdot \mathrm{Mop}_{(\mathrm{M}_2)} + p_1 \cdot \big(\mathrm{Mop}_{(\mathrm{M}_3)} + \mathrm{Mop}_{(\mathrm{M}_8)}\big) \\ & + f_2 \cdot \big(\mathrm{Mop}_{(\mathrm{M}_5)} + \mathrm{Mop}_{(\mathrm{M}_1)} + \mathrm{Mop}_{(\mathrm{M}_4)} + \mathrm{Mop}_{(\mathrm{M}_7)}\big) \\ & + f_3 \cdot \big(\mathrm{Mop}_{(\mathrm{M}_6)} + \mathrm{Mop}_{(\mathrm{M}_2)} + \mathrm{Mop}_{(\mathrm{M}_4)} + \mathrm{Mop}_{(\mathrm{M}_7)}\big) \\ & + p_3 \cdot \mathrm{Mop}_{(\mathrm{M}_9)} + p_5 \cdot \mathrm{Mop}_{(\mathrm{M}_3)} + w \cdot \big(\mathrm{Mop}_{(\mathrm{M}_7)}\big)\end{aligned} \tag{4}$$

Code analysis further reveals that p_2_cnt is already considered within the f_2 factors, and adds no additional costs. Moreover, p_6_cnt (in the last `if` statement of the main loop) is negligible, given its very small value relative to the rest of the coefficients. In addition, the constant factors in (5) are due to single floating-point operations in the main loop.

$$\mathrm{Flop}_{(\mathcal{P}_\mathcal{R})} = f_2 \cdot \big(\mathrm{Flop}_{(\mathrm{M}_1)} + \mathrm{Flop}_{(\mathrm{M}_4)} + 3\big) + f_3 \cdot \big(\mathrm{Flop}_{(\mathrm{M}_4)} + 2\big) + p_4 + p_5 + w. \tag{5}$$

Inserting the values from Table 1 into Eq. (4) and (5) determines the final value for $\mathcal{P}_\mathcal{R}$ as follows:

$$\mathcal{P}_\mathcal{R} = \underbrace{f_1 + 3 \cdot p_1 + 6 \cdot f_2 + 2 \cdot p_3 + w + 6 \cdot f_3 + p_5}_{\text{Mop cost}} \oplus \underbrace{5 \cdot f_2 + w + 3 \cdot f_3 + p_5 + p_4}_{\text{Flop cost}}. \tag{6}$$

Table 1. The Mop and Flop cost (in number of operations) of the functions employed in the implementation of BMM algorithm in `PISA` library

Function	M_1: `max_score`	M_2: `doc_id`	M_3: `score`	M_4: `block_max_score`	M_5: `block_max_doc_id`
Mop	1	1	1	2	2
Flop	1	–	–	1	–
	M_6: `next_geq`	M_7: `would_enter`	M_8: `next`	M_9: `block_max_next_geq`	
Mop	2	2	2	1	
Flop	–	–	–	–	

5.3 Latency Model Based on $\mathcal{P}_\mathcal{R}$

Our current $\mathcal{P}_\mathcal{R}$ model enables us to determine the number of operations, but does not translate to latency directly. We thus formulate a latency model based on $\mathcal{P}_\mathcal{R}$ with coefficients as input and latency as output. We expect the latency model based on $\mathcal{P}_\mathcal{R}$ to combine the effects of both memory and floating point operations. However, depending on whether the application is compute- or memory-bound, the impact of one of the two components might become negligible. Thus, we investigate three possible variants to convert our symbolic work model into a latency model: (i) considering only Mop operations; (ii) considering only Flop operations; and (iii) considering both Mop and Flop operations.

We also consider an additional scheme (iv) where we only consider the coefficients, and ignore the factors in $\mathcal{P}_\mathcal{R}$.

We simplify the latency model by assuming the latency of all operations is approximately the same: T_M for memory operations and T_F for floating-point operations. Hence, our latency model is formulated as:

$$\mathcal{L}_{\mathcal{P}_\mathcal{R}} = \underbrace{(f_1 + 3 \cdot p_1 + 6 \cdot f_2 + 2 \cdot p_3 + w + 6 \cdot f_3 + p_5) \cdot T_M}_{\text{Mop latency}} + \underbrace{(5 \cdot f_2 + w + 3 \cdot f_3 + p_5 + p_4) \cdot T_F}_{\text{Flop latency}}. \tag{7}$$

We randomly select a small percentage of the queries and collected their coefficients in $\mathcal{P}_\mathcal{R}$ as the training set for estimating T_M and T_F. With the single-latency schemes ((i) and (ii)), only one of them needs to be estimated. By considering either of Mop latency or Flop latency with the training set, we obtain the per-query estimates of T_M and T_F. Then, we set the final values of T in either case as the mean over all per-query estimates. For the third scheme, where we combine both latencies, we apply the least square regression method to solve T_M and T_F with the training set. Finally, we apply the linear regression method for the last latency model with the collected per-query coefficients as the model input and latency as its output.

Our latency models require the execution of the evaluation algorithm for all queries to collect the coefficients. However, in practice it is feasible to obtain accurate estimates of coefficients with a small subset of the data, as we demonstrate empirically in Sect. 6.4.

6 Evaluation

The evaluation of PEIR focuses on (1) validating the correctness of $\mathcal{P}_\mathcal{R}$; (2) evaluating the correlation between Mop and latency; and (3) assessing the data collection requirements and feasibility.

6.1 Experimental Setup

We use the MS-MARCO v1 development set for evaluating the efficiency of retrieval methods. The dataset contains 9.9M passages and 6,980 queries [1]. As we focus on retrieval efficiency, we follow the exact same experimental setup as in [16] and consider the following methods: (i) **BM25**: using the BM25 scoring function [24]. (ii) **BM25-T5**: documents are expanded with query predictions from T5 [23]. (iii) **DeepImpact**: based on [17]. (iv) **uniCOIL-T5**: based on COIL [8] model and with T5 expansions [12]. (v) **uniCOIL-TILDE**: based on [32], using COIL [8] but with TILDE [33] expansion. (vi) **SPLADEV2** [4]: adds additional optimization on SPLADEV1 [6].

Our experiments are performed using the PISA library [18], which provides efficient C++ implementations of several query evaluation algorithms; PISA outperforms other options such as Anserini [31] and JASS [13,30] w.r.t. query latency [16]. We focus on retrieval latency with pre-processed queries and a

pre-built index, and select the BMM [3] algorithm for query evaluation, given its superiority over BlockMaxWand [19]. However, PEIR is not limited to these particular choices of library or query evaluation algorithm. While executing BMM, we use $k = 1000$ and quantization (except for BM25 methods). We measure latency in milliseconds, run each experiment five times for every combination of settings, and report the mean value.

To evaluate the correctness of the latency models proposed in Sect. 5.3, we measure the R2 score, where values closer to 1 indicate that the model fits the data well, and negative values indicate a failure in the regression.

To confirm the portability of PEIR, we perform experiments on two machines with different CPUs: an AMD EPYC 7402P 24-core processor, and a 12-core Intel(R) Xeon(R) Gold 5118 CPU (Skylake). We run all experiments on a single core. We compile the PISA library in two different versions from scratch: an instrumented version (see Sect. 5.1) with performance counters to collect data for coefficient estimation, and a timed version, where we only add a few timers to collect the processing time of different code blocks of the BMM algorithm. This setup ensures that the data collection overhead does not affect the latency data.

Table 2. R2 scores for different performance models and retrieval methods.

Retrieval	**Intel**					**AMD**				
method	**Mop**	**Flop**	**MFlop**	**LR**	**PL**	**Mop**	**Flop**	**MFlop**	**LR**	**PL**
BM25	0.801	−0.259	0.969	0.990	0.589	0.463	−1.076	0.820	0.884	0.537
BM25-T5	0.968	0.367	0.989	0.997	−0.000	0.742	0.308	0.950	0.964	0.001
uniCOIL-TILDE	0.986	0.930	0.985	0.984	0.300	0.979	0.930	0.977	0.977	0.302
DeepImpact	0.969	0.469	0.987	0.993	0.444	0.957	0.485	0.970	0.977	0.429
uniCOIL-T5	0.971	0.939	0.985	0.977	0.261	0.966	0.941	0.981	0.973	0.255
SPLADEV2	0.988	0.859	0.990	0.997	0.617	0.983	0.856	0.986	0.994	0.605

6.2 Validating the Correctness of $\mathcal{P}_{\mathcal{R}}$

As explained in Sect. 5.3, we first create a training set by randomly selecting 1% of the collected data and estimate T_M and/or T_F in the latency models.

The first four columns in Table 2, with both CPUs (Intel and AMD), show the R2 scores for all the retrieval methods all using the latency models based on $\mathcal{P}_{\mathcal{R}}$ (For brevity, we represent **Mop** + **Flop** as **MFlop**). To provide more elaborate comparisons, we also present the results for a simple baseline **PL** in the fifth column. **PL** is a linear regression model to predict retrieval latency based on the sum of posting-lists size per query.

We note that as expected, predicting latency through features such as posting-list size (PL column in Table 2) is not effective and reliable. Even after increasing the training data size, improvements were minor and we achieved

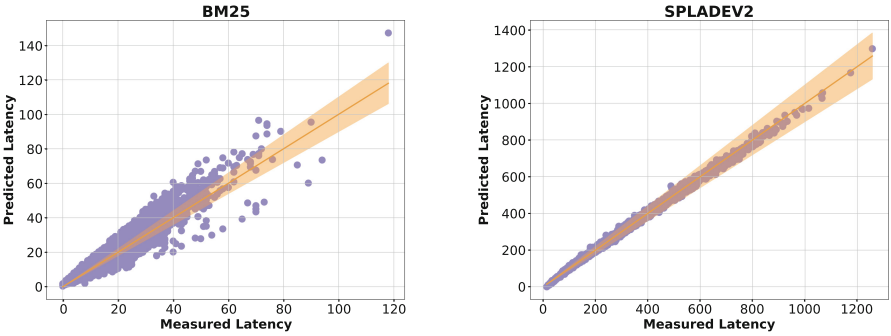

Fig. 1. Measured vs. predicted latency based on **LR** model.

similar observations. This is primarily due to algorithmic optimizations during retrieval, such as the ordering and skipping of documents.

The majority of other latency models achieve high R2 scores (more than 0.95 in many cases), validating the correctness of $\mathcal{P}_{\mathcal{R}}$. The notable exception are the Flop-based latency models, which perform poorly because, with the pre-built indexes and processed queries, most floating-point operations are completed before the retrieval stage, and the query evaluation algorithm is heavily memory-bound. Combining floating-point operations together with memory operations results in mild improvements in R2 scores. The latency models that only use coefficients as input (LR column in Table 2), achieve the best overall result since they benefit from ground-truth latency measurements, but they lack the interpretability of Mop-based models.

In Fig. 1, we illustrate the measured vs. predicted latency with a LR latency model for SPLADEV2 and BM25 and an AMD CPU. The straight lines in these graphs indicate the target predictions, the shaded areas represent 10% error region, and the dots represent the different queries. These figures illustrate that the majority of errors for the proposed performance models are due to queries with a lower latency. The lower the latency, the harder it is to make simplifying assumptions in the performance models: even a few milliseconds of difference in predictions translates to higher error rate. Moreover, by observing Table 2, we notice that the error rate for BM25 is more pronounced in Mop-based models than in LR models which is due to Mop-based models considering the *factors* in latency prediction.

In summary, our validation experiments demonstrate that the Mop-based latency model $\mathcal{L}_{\mathcal{R}}$ offers a very good trade-off between interpretability and accuracy. The remainder of our evaluation focuses on this model. Additionally, for brevity, and given that we observe the same behaviour on both Intel and AMD, we present results only from the Intel system for the rest of evaluation.

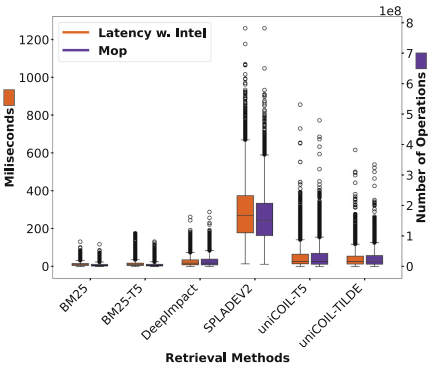

Fig. 2. Comparison between distributions of latency and number of memory operations for all retrieval methods.

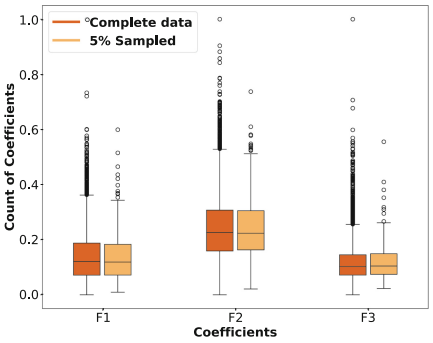

Fig. 3. Comparison between distributions of coefficients for all queries and a randomly sampled subset of them.

6.3 Correlation Between Mop and Latency

The aim of PEIR is to provide an evaluation framework in which the efficiency (in terms of latency) of retrieval methods is compared in a hardware-agnostic manner. To this end, we constructed symbolic performance models $\mathcal{P}_{\mathcal{R}}$ and Mop−based latency models for six retrieval methods. We use these models to compare the methods using both the measured latency and the predicted indicators (i.e., Mops and predicted latency). We aim to answer two questions: (i) How does the distribution of predicted Mops correlate to the distribution of measured latency per method, across all the evaluation queries? (ii) How accurate is the relative efficiency comparison based on predicted Mops in comparison to the actual comparison based on measured latency?

Figure 2 illustrates the memory operations distribution for all six retrieval methods. We observe that the two distributions closely match each other. Additionally, the data for each method, for total and average Mop, measured latency, and their relative ratio to BM25 are presented in Table 3. As expected, the measured latency is different across different CPUs. Mop is independent of what CPU is used. We also note that *the relative comparisons between any pair of retrieval methods yields similar conclusions for the predicted* Mop *and the measured latency.* As shown in Fig. 2, this statement holds not only for the mean, but for all distribution quantiles. These observations confirm that using PEIR to define performance models, and using Mop as efficiency indicator successfully allows for hardware-agnostic efficiency comparison of retrieval methods.

6.4 Data Collection Requirements and Feasibility

We have successfully demonstrated that comparing retrieval models using predicted Mop instead of measured latency is feasible (Sect. 6.3). However, to calculate Mop or latency, model coefficients are determined by collecting data from the

Table 3. Comparison of retrieval methods: Average Latency (L) vs. Average Mops

R. method	L_{AMD}	L_{Intel}	Mop $[\times 10^6]$	L_{AMD}^{BM25}	L_{Intel}^{BM25}	Mop^{BM25}
BM25	13.90	12.26	5.4116	1.00	1.00	1.00
BM25-T5	17.15	18.32	7.3242	1.23	1.49	1.35
DeepImpact	24.50	26.37	17.9721	2.49	2.60	2.70
SPLADEV2	277.97	290.97	161.8180	28.24	28.66	24.28
uniCOIL-T5	48.92	50.69	32.4773	4.97	4.99	4.87
uniCOIL-TILDE	40.88	41.32	26.5466	4.15	4.07	3.98

evaluation algorithm executed for all queries (Sect. 5.3). This can be a lengthy, possibly inefficient process.

To improve data collection efficiency, we propose sampling: we randomly select 5% of the queries and compare the distribution of three high-impact coefficients in the Mop part of Eq. (4). The results, shown in Fig. 3 for SPLADEV2 method, indicate very similar distributions.[3] Thus, by collecting only coefficients for a small set of queries, we can more efficiently derive the needed model coefficients, with a minimal reduction in accuracy.

7 Conclusion

We have proposed PEIR, a framework that enables deriving performance models for query evaluation methods in the context of LSR. The models estimate the performance of the retrieval stage w.r.t. the number of operations in a LSR pipeline. Considering several LSR methods, the `BlockMax-MaxScore` (BMM) algorithm in PISA, the MS MARCO dataset, and two hardware configurations, we demonstrated that we can successfully use the number of memory operations in a query evaluation method as a reliable proxy to estimate latency. The limitation of PEIR is the possible overhead of instrumenting query evaluation libraries.

Employing our proposed performance models in practice could be highly beneficial, as it provides standardization of efficiency analysis within LSR and facilitates reproducible and comparable efficiency measurements across various settings. Integrating PEIR in existing IR benchmarking frameworks such as TIREX [7] would considerably reduce the required computational resources.

For future research in this direction, we focus on (i) improving further the instrumentation and calibration of the models, (ii) improving the accuracy of the performance models further, (iii) integrating query encoding latency into PEIR, to consider the influence of model complexity on query latency analysis, and (iv) extending the models to other metrics, like energy consumption, thus enabling more thorough evaluation of LSR methods.

[3] We observe the same behaviour with other retrieval methods.

Acknowledgments. This research was (partially) supported by the Dutch Research Council (NWO), under project numbers 024.004.022, NWA.1389.20.183, KICH3.LTP.20.006, and VI.Vidi.223.166, and the European Union's Horizon Europe program under grant agreement No 101070212.

We thank Mackenzie et al. [16] for sharing the index files used in their paper with us. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

References

1. Bajaj, P., et al.: MS marco: a human generated machine reading comprehension dataset (2018). https://arxiv.org/abs/1611.09268
2. Bruch, S., Mackenzie, J., Maistro, M., Nardini, F.M.: A proposed efficiency benchmark for modern information retrieval systems. Technical report (2023). Presented at ReNeuIR 2023 (at SIGIR 2023, Tapei)
3. Chakrabarti, K., Chaudhuri, S., Ganti, V.: Interval-based pruning for top-k processing over compressed lists. In: Proceedings of the 2011 IEEE 27th International Conference on Data Engineering, pp. 709–720 (2011)
4. Formal, T., Lassance, C., Piwowarski, B., Clinchant, S.: Splade v2: sparse lexical and expansion model for information retrieval (2021). https://arxiv.org/abs/2109.10086
5. Formal, T., Lassance, C., Piwowarski, B., Clinchant, S.: From distillation to hard negative sampling: making sparse neural ir models more effective (2022). https://arxiv.org/abs/2205.04733
6. Formal, T., Piwowarski, B., Clinchant, S.: Splade: sparse lexical and expansion model for first stage ranking. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 2288–2292 (2021)
7. Fröbe, M., et al.: The information retrieval experiment platform. In: Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 2826–2836 (2023)
8. Gao, L., Dai, Z., Chen, T., Fan, Z., Van Durme, B., Callan, J.: Complement lexical retrieval model with semantic residual embeddings. In: Hiemstra, D., Moens, M.-F., Mothe, J., Perego, R., Potthast, M., Sebastiani, F. (eds.) ECIR 2021. LNCS, vol. 12656, pp. 146–160. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72113-8_10
9. Hofstatter, S., Hanbury, A.: Let's measure run time! Extending the IR replicability infrastructure to include performance aspects. In: Proceedings of the Open-Source IR Replicability Challenge co-located with 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, CEUR Workshop Proceedings, vol. 2409, pp. 12–16 (2019)
10. Lassance, C., Clinchant, S.: An efficiency study for splade models. In: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2022, pp. 2220–2226 (2022)
11. Li, J., et al.: Towards an accurate latency model for convolutional neural network layers on gpus. In: MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM), pp. 904–909 (2021)
12. Lin, J., Ma, X.: A few brief notes on deepimpact, coil, and a conceptual framework for information retrieval techniques (2021). https://arxiv.org/abs/2106.14807

13. Lin, J., Trotman, A.: Anytime ranking for impact-ordered indexes. In: Proceedings of the 2015 International Conference on The Theory of Information Retrieval, pp. 301–304 (2015)
14. Lym, S., Lee, D., O'Connor, M., Chatterjee, N., Erez, M.: DeLTA: GPU performance model for deep learning applications with in-depth memory system traffic analysis. In: Proceedings - 2019 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2019, pp. 293–303 (2019)
15. Ma, Z., et al.: Dynaboard: an evaluation-as-a-service platform for holistic next-generation benchmarking. In: Proceedings of the 35th International Conference on Neural Information Processing Systems (2024)
16. Mackenzie, J., Trotman, A., Lin, J.: Efficient document-at-a-time and score-at-a-time query evaluation for learned sparse representations. ACM Trans. Inf. Syst. **41**(4), 1–28 (2023)
17. Mallia, A., Khattab, O., Suel, T., Tonellotto, N.: Learning passage impacts for inverted indexes. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2021, pp. 1723–1727 (2021)
18. Mallia, A., Siedlaczek, M., Mackenzie, J., Suel, T.: PISA: performant indexes and search for academia. In: Proceedings of the Open-Source IR Replicability Challenge co-located with 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 50–56 (2019)
19. Mallia, A., Siedlaczek, M., Suel, T.: An experimental study of index compression and DAAT query processing methods. In: Azzopardi, L., Stein, B., Fuhr, N., Mayr, P., Hauff, C., Hiemstra, D. (eds.) ECIR 2019. LNCS, vol. 11437, pp. 353–368. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-15712-8_23
20. Mathis, M.M., Amato, N.M., Adams, M.L.: A general performance model for parallel sweeps on orthogonal grids for particle transport calculations. In: Proceedings of the 14th International Conference on Supercomputing, ICS 2000, pp. 255–263 (2000)
21. Nguyen, T., MacAvaney, S., Yates, A.: A unified framework for learned sparse retrieval. In: Advances in Information Retrieval: 45th European Conference on Information Retrieval, ECIR 2023, pp. 101–116 (2023)
22. Pompougnac, H., Dutilleul, A., Guillon, C., Derumigny, N., Rastello, F.: Performance bottlenecks detection through microarchitectural sensitivity. arXiv preprint arXiv:2402.15773 (2024)
23. Raffel, C., et al.: Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res. **21**(1), 1–67 (2020)
24. Robertson, S., Zaragoza, H.: The probabilistic relevance framework: BM25 and beyond. Found. Trends Inf. Retr. **3**(4), 333–389 (2009)
25. Santhanam, K., et al.: Moving beyond downstream task accuracy for information retrieval benchmarking. In: Findings of the Association for Computational Linguistics: ACL 2023, pp. 11613–11628 (2023)
26. Scells, H., Zhuang, S., Zuccon, G.: Reduce, reuse, recycle: green information retrieval research. In: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 2825–2837 (2022)
27. Schurman, E., Brutlag, J.: Performance related changes and their user impact. In: Velocity Web Performance and Operations Conference (2009)
28. Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in NLP. In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pp. 3645–3650 (2019)

29. Treibig, J., Hager, G., Wellein, G.: Best practices for HPM-assisted performance engineering on modern multicore processors. arXiv preprint arXiv:1206.3738 (2012)
30. Trotman, A., Crane, M.: Micro-and macro-optimizations of SaaT search. Softw. Pract. Exp. **49**(5), 942–950 (2019)
31. Yang, P., Fang, H., Lin, J.: Anserini: reproducible ranking baselines using lucene. J. Data Inf. Qual. **10**(4), 1–20 (2018)
32. Zhuang, S., Zuccon, G.: Fast passage re-ranking with contextualized exact term matching and efficient passage expansion (2021)
33. Zhuang, S., Zuccon, G.: TILDE: term independent likelihood model for passage re-ranking. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 1483–1492 (2021)