An Efficient and Robust Semantic Hashing Framework for Similar Text Search

LIYANG HE, ZHENYA HUANG, ENHONG CHEN, QI LIU, SHIWEI TONG, HAO WANG, and DEFU LIAN, University of Science and Technology of China and State Key Laboratory of Cognitive Intelligence
SHIJIN WANG, iFLYTEK Co., Ltd and State Key Laboratory of Cognitive Intelligence

Similar text search aims to find texts relevant to a given query from a database, which is fundamental in many information retrieval applications, such as question search and exercise search. Since millions of texts always exist behind practical search engine systems, a well-developed text search system usually consists of recall and ranking stages. Specifically, the recall stage serves as the basis in the system, where the main purpose is to find a small set of relevant candidates accurately and efficiently. Towards this goal, deep semantic hashing, which projects original texts into compact hash codes, can support good search performance. However, learning desired textual hash codes is extremely difficult due to the following problems. First, compact hash codes (with short length) can improve retrieval efficiency, but the demand for learning compact hash codes cannot guarantee accuracy due to severe information loss. Second, existing methods always learn the unevenly distributed codes in the space from a local perspective, leading to unsatisfactory code-balance results. Third, a large fraction of textual data contains various types of noise in real-world applications, which causes the deviation of semantics in hash codes. To this end, in this article, we first propose a general unsupervised encoder-decoder semantic hashing framework, namely MASH (short for Memory-bAsed Semantic Hashing), to learn the balanced and compact hash codes for similar text search. Specifically, with a target of retaining semantic information as much as possible, the encoder introduces a novel relevance constraint among informative high-dimensional representations to guide the compact hash code learning. Then, we design an external memory where the hashing learning can be optimized in the global space to ensure the code balance of the learning results, which can promote search efficiency. Besides, to alleviate the performance degradation problem of the model caused by text noise, we propose an improved SMASH (short for denoiSing Memory-bAsed Semantic Hashing) model by incorporating a noise-aware encoder-decoder framework. This framework considers the noise degree for each text from the semantic deviation aspect, ensuring the robustness of hash codes. Finally, we conduct extensive experiments in three real-world datasets. The experimental results clearly demonstrate the effectiveness and efficiency of MASH and SMASH in generating balanced and compact hash codes, as well as the superior denoising ability of SMASH.

This research was partially supported by grants from the National Key Research and Development Program of China (Grant No. 2021YFF0901005) and the National Natural Science Foundation of China (Grants No. 61922073, 62106244, and U20A20229).

Authors' addresses: L. He, Z. Huang (corresponding author), E. Chen, Q. Liu, S. Tong, H. Wang, and D. Lian, Anhui Province Key Laboratory of Big Data Analysis and Application, University of Science and Technology of China, Hefei 230027, China and State Key Laboratory of Cognitive Intelligence, Hefei 230088, China; emails: heliyang@mail.ustc.edu.cn; {huangzhy, cheneh, qiliuql}@ustc.edu.cn, tongsw@mail.ustc.edu.cn, {wanghao3, liandefu}@ustc.edu.cn; S. Wang, iFLYTEK AI Research (Central China), iFLYTEK Co., Ltd, China and State Key Laboratory of Cognitive Intelligence, Hefei 230088, China; email: sjwang3@iflytek.com.

1046-8188/2023/03-ART91 $15.00
https://doi.org/10.1145/3570725

CCS Concepts: • **Information systems** → *Retrieval models and ranking*; **Document representation;**

Additional Key Words and Phrases: Semantic hashing, similarity search, efficient codes, robust codes

ACM Reference format:
Liyang He, Zhenya Huang, Enhong Chen, Qi Liu, Shiwei Tong, Hao Wang, Defu Lian, and Shijin Wang. 2023. An Efficient and Robust Semantic Hashing Framework for Similar Text Search. *ACM Trans. Inf. Syst.* 41, 4, Article 91 (March 2023), 31 pages.
https://doi.org/10.1145/3570725

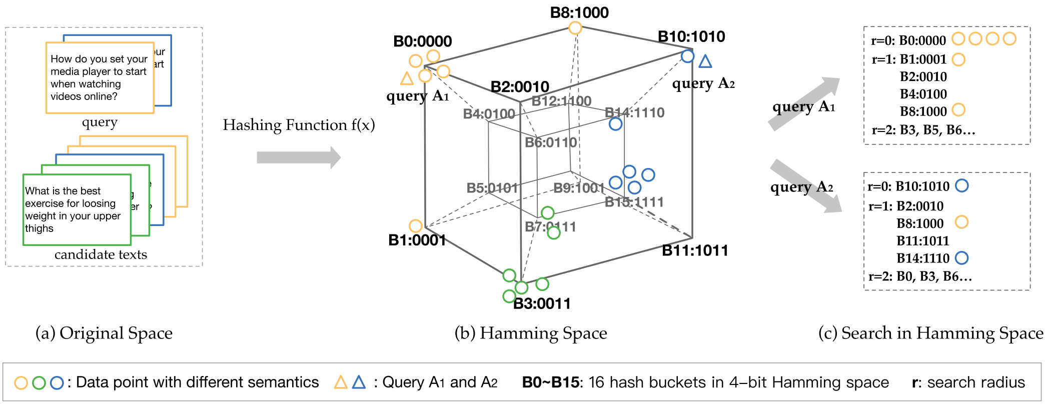
1 INTRODUCTION

Similar text search is a fundamental information retrieval task that aims at finding texts similar to a given query, which has been applied to many practical applications. For example, in **community-based question answering** (**CQA**) sites like Quora,[1] people can search for similar questions they are interested in and find desired answers [2, 53, 77]. Meanwhile, in online intelligent education systems like Khan Academy,[2] students usually consolidate knowledge by finding similar exercises based on their wrong exercises [44, 59, 70]. Since the number of collected texts is rapidly growing, the modern industrial similar text search system follows the *recall-then-rank* two-stage paradigm. Specifically, given a user query, the *recall* stage tends to retrieve an initial subset of relevant candidates from the extensive database in low latency, which usually contains lightweight algorithms or models assisted by some indexing structure. Then the *ranking* stage elaborately scores the candidates with more complicated algorithms or models to achieve better-sorted results [29, 41, 51]. As the bottleneck, the recall stage is non-negligible in this pipeline with the goal of effectiveness and efficiency, i.e., to return a set of texts covering relevant texts as many as possible within a short time-span [20].

To achieve satisfying performance in the recall stage, earlier search techniques apply various term matching methods [81] for the cases of keyword match and build the inverted index [80] for efficiency. However, they cannot address desired results that do not exactly match the query text but can satisfy users' search intent, which is also called the mismatch problem and may block the ranking stage from relevant texts at the beginning [91]. Thus it remains a problem for semantic retrieval [39]. With the significant progress of deep learning in various domains [28, 31, 38, 43, 45, 55, 64], representation learning, which uses embeddings to represent items, has been proven to be a successful technique contributing to the semantic retrieval [21, 27, 68]. In the recall stage, the application of representation learning is to use embeddings to represent queries and items. Then it converts the retrieval problem into a nearest neighbor search problem in the embedding space [29].

In this setting, semantic hashing [47, 61, 72], which enables a very efficient search and learns semantic embeddings of texts (or other types of data objects [9, 57, 89]), is suitable for the recall stage and has attracted a great deal of attention. Concretely, as shown in Figure 1(a) and (b), semantic hashing projects texts in the original space into a much smaller compact Hamming space [25], where texts are turned into binary representations of remaining semantics (e.g., question's topic). The binary representations are called hash codes and fall into corresponding hash buckets. On the one hand, semantic hashing can utilize the text's semantics to ensure candidates' relevance. On the other hand, it can also handle the efficiency problem on two sides. First, storing hash codes is significantly storage-efficient. For example, suppose there are 10 million texts with 512-dimensional

[1] https://www.quora.com/.
[2] https://www.khanacademy.org/.

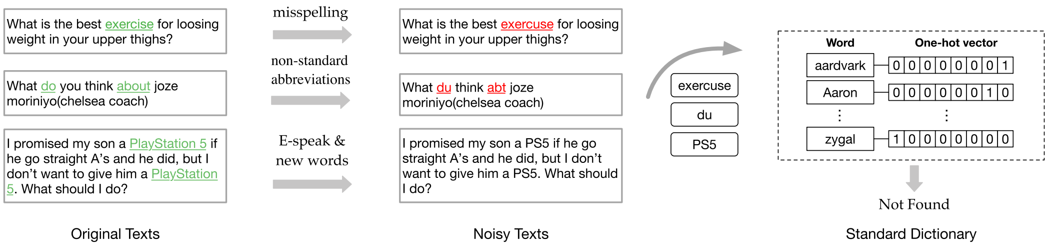


(a) Original Space (b) Hamming Space (c) Search in Hamming Space

Fig. 1. A showcase of similar text search in text semantic hashing. (a) and (b) show the hashing function projects original texts into the binary Hamming space. (b) illustrates hash codes in a 4-bit Hamming space, which can be seen as a hypercube in geometric space. Each circle represents a text in the database, and triangles represent queries. Different colors mean a specific semantic. (c) plots the search procedure in the Hamming space. For query A, we first set the search radius $r = 0$ to access bucket $B0$ and get four related texts. Then we set search radius $r = 1$ to access buckets $B1, B2, B4, B8$, and get one related text and so on.

double-precision float vectors. Their representations will take over 19 GB of storage space, which is hard to be deployed into general devices with limited memory. However, only 19 M of storage is needed to store these texts if we use 16-bit hash codes for representations. Second, the binary codes are used as direct indices (addresses) into a hash table to realize search efficiency, yielding a remarkable increase in search speed compared to an exhaustive linear scan [52]. As illustrated in Figure 1(c), we generally explore the Hamming space to acquire candidate texts by increasing the search radius r around the query. Based on the above search procedure and according to the different termination conditions, it introduces two kinds of problems. The first is to find exact K texts closest to Hamming distance to a given query, namely the **k-nearest neighbor** (**KNN**) search. The second is to find all texts within a fixed Hamming distance of a query, namely **Point Location in Equal Balls** (**PLEBs**) search [33].

To generate effective hash codes, early works [78, 86] are developed to preserve similar structures of the original texts inspired by spectral clustering [50] and latent semantic indexing [13]. With the progress of deep learning, modern approaches typically use unsupervised encoder-decoder architectures to retain the textual semantics [6, 8, 15, 23, 24, 63, 82]. Though this line achieves exemplary achievements with different targets, they usually suffer from the following three deficiencies in the practical recall system. First, compactness is a crucial factor in hash codes [72], which means the hash codes should be as short as possible. On the one hand, short hash codes significantly promote search efficiency. Briefly, the search cost mainly depends on the number of lookups, i.e., the number of buckets examined, which will exponentially explode as the code length b increases [18] (a detailed explanation is presented in Section 2.2). On the other hand, with the explosive growth of social media and e-commerce, more and more short text data (e.g., questions and tweets) are generated every day on the Web. It would be a colossal waste to represent them as long hash codes [82]. Second, hash codes should ensure the property of code balance, which means hash codes are evenly spread throughout the Hamming space [25]. Specifically, code balance is important because it helps to reduce information redundancy in hash codes, then makes it better to preserve the original locality structure of the data [15]. In addition, code balance narrows the search latency gap between bad cases and ordinary cases, therefore improving user



What is the best exercise for loosing weight in your upper thighs?

What do you think about joze moriniyo(chelsea coach)

I promised my son a PlayStation 5 if he go straight A's and he did, but I don't want to give him a PlayStation 5. What should I do?

What is the best exercuse for loosing weight in your upper thighs?

What du think abt joze moriniyo(chelsea coach)

I promised my son a PS5 if he go straight A's and he did, but I don't want to give him a PS5. What should I do?

Fig. 2. The common text noise in a similar search. These noise always leads to out-of-vocabulary words.

experience. For example, assume we want to acquire four relevant texts in the KNN search. As shown in Figure 1(c), for query A_1, we only need to access the hash bucket $B0$ to reach four texts. However, we must access 6 to 11 hash buckets for query A_2. The huge gap between A_1 and A_2 brings users a poor experience. In the PLEB search, please note that the lookups are equal for each query in the recall stage using the same search radius r, but the number of returned candidates impacts the ranking stage's efficiency. For example, suppose we set search radius $r = 0$, still in Figure 1(c), query A_1 and query A_2 will retrieve four and one candidate texts respectively. That means, compared to A_2, A_1 will spend about four times as long in the rank stage to score candidate texts and even cost more time in sorting. It is clear that worst search cases can be minimized if the hash buckets contain an equal number of texts in both search ways [25]. That is also the code-balance target. Third, the hash code should keep the semantics of text as much as possible for reliable retrieval results.

Unfortunately, there are many technical and domain challenges to achieving the ideal hash codes described above. First, it is difficult to ensure effectiveness when the hash code is short because of information loss, which is also called the few-bits hashing problem [82]. Second, generating uniformly distributed hash codes with the code-balance target are virtually impossible in a large-scale dataset. For example, some works [15, 63] notice the code-balance problem, but they optimize it from a local perspective, which means they only include very few hash codes to estimate the distribution. (e.g., in a training batch). For this reason, they cannot ensure code balance in the entire space encountering a large-scale dataset. It essentially limits their performance on efficiency for bad search cases. Last but not least, as illustrated in Figure 2, a large fraction of textual data contains various types of noise in real-world applications [3, 22, 40], such as misspelling, non-standard abbreviations, and E-speak & new words. For example, Cucerzan et al. [11] report that the frequency of misspelling occurs in 10%–15% of search engine queries. Besides, current social communities (e.g., Quora) contain frequent use of non-standard abbreviations (e.g., "du" for "do you", "abt" for "about") or E-speak & new words (e.g., "PS5" for "PlayStation 5"). The text noise creates legal or legitimate words not presented in standard dictionaries, which are also called **out-of-vocabulary words** (**OOVs**) [79]. The out-of-vocabulary words in queries will cause the semantics' deviation in their hash codes, leading to undesired search results. Thus, ensuring the robustness of hash codes against text noise brings us a critical challenge for designing a semantic hashing model in practice.

To address the first two more general challenges, we propose a **Memory-bAsed Semantic Hashing** (**MASH**) model for deep semantic hashing. In MASH, we aim at achieving both efficiency and effectiveness in hash codes. Specifically, MASH follows the encoder-decoder framework in an unsupervised manner, where the encoder generates the hash codes from texts, and the decoder reconstructs the texts on their own. To tackle the few-bits hashing problem, we first propose a novel relevance propagation objective in the encoder to use the relevance constraints among informative high-dimensional representations to guide the short hash code learning for retaining semantic information. Then, to achieve the code-balance target in the entire Hamming space, we introduce an external memory to store all hash codes in the training procedure. Besides, we design a strategy to select representative hash codes stored in the external memory to precisely adjust the distribution of new hash codes.

Furthermore, to generate robust hash codes with noisy texts in real-world applications, we extend MASH and propose a **denoiSing Memory-bAsed Semantic Hashing** (**SMASH**) framework. Specifically, we incorporate a noise-aware encoder-decoder framework into our model. The basic idea is to reconstruct the corrupted text as the clean one in the encoder-decoder framework. Considering the various semantic deviation in each corrupted text, we assign latent weights to them for generating more robust hash codes. Besides, we design an adaptive noise rate generation strategy to release the manual setting and improve the generalization ability in our framework.

Finally, we conduct extensive experiments and evaluate both MASH and SMASH frameworks on three real-world datasets. The experimental results in both general and noise scenarios demonstrate the effectiveness and efficiency of the two proposed frameworks in large-scale recall systems.

2 RELATED WORK

Similar search aims at searching similar data from a large-scale database, which is fundamental in many information retrieval applications [4, 19, 49, 76, 87, 92], such as searching similar questions in **community-based question answering** (**CQA**) sites [2, 53, 77] and finding similar exercises in online intelligent education systems [44, 59, 70]. A similar search system usually comprises *recall* and *ranking* stages to respond to users' requests in real-time. In this article, we mainly focus on similar text search methods in the *recall* stage, which aims at proposing a small set (such as hundreds to thousands) of relevant candidates with the goal of effectiveness and efficiency. In the early years, people usually used word matching methods like the well-known BM25 model [60] with the inverted index [80] for textural data. Due to the limitation of the semantic match, some deep learning methods [30, 32, 42, 48, 54] use the embedding to represent data. Meanwhile, approximate search [1, 34, 36, 61, 67] has been incorporated into this field for fast search, where semantic hashing [61] can achieve both effectiveness and efficiency by learning semantic binary representation. This section summarizes related works in the following categories, including semantic hashing and characteristics of hash codes for the reader to better understand our work.

2.1 Semantic Hashing

The most representative class for similarity search is hash, which projects the data from the original space into Hamming space. In the past, **Locality Sensitive Hashing** (**LSH**) [12] is a convenient and popular hashing method with complete theoretical guarantees [74]. However, it is data-independent, which is unable to capture the semantics of texts. To tackle this problem, semantic hashing [47, 61, 72] has been proposed to leverage machine learning or deep learning techniques to learn data-dependent hashing functions. It aims at generating semantic hash codes that maintain the text's structure. Then the nearest neighbor search on hash codes leads to a search result similar to the nearest neighbor search on the original text space.

In general, semantic hashing methods can be broadly classified into supervised and unsupervised methods. In supervised methods [37, 46, 73, 75], scholars try to use relevance labels to learn hashing function and achieve better performance. However, such supervised methods suffer from human-annotated costs. They are not suitable for similar search systems nowadays that contain millions of textual resources. Therefore, unsupervised hash approaches learn hashing functions without any supervised signal, which are more feasible for large-scale corpora.

There are many notable unsupervised shallow-hashing methods [16, 26, 50, 78, 86] in early works. For instance, **Spectral Hashing** (**SPH**) [78] can be viewed as an extension of spectral clustering [50]. It aims at learning hash codes that preserve the global similarity structures of the original text. **Self-Taught Hashing** (**STH**) [86] has the objective of preserving the local similarities between samples found via a k-nearest neighbor search. However, this is done by computing the bit vectors by considering text connectivity without learning text features. From the aspect of quantization loss, **Spectral Hashing** (**SphcHash**) [26] learns compact hash codes by solving the Eigenvector problem, which achieves significant performance. However, it strongly assumes that the data points are uniformly distributed in a hyper-rectangle, limiting its application potential. On the other hand, **Spherical Hashing** (**SpheHash**) [26] employs hypersphere-based partitioning to achieve better and more coherent quantization. Furthermore, **Iterative Quantization** (**ITQ**) [16] additionally minimizes the quantization error to generate hashing functions that better preserve the original locality structure of the input data. However, these shallow models are linear and cannot be applied to real-world high-dimensional datasets.

With the increasing attention of deep learning, many research efforts [6–8, 15, 23, 24, 63, 82, 90, 94] have been devoted to deep semantic hashing methods that display remarkable performance improvements. In this article, we focus on text-based unsupervised deep methods since they are suitable for large-scale text collections. For example, Chaidaroon et al. [8] propose the first work in this direction, named **Variational Deep Semantic Hashing** (**VDSH**). They employ **variational autoencoders** (**VAE**) [58] framework to learn hash codes in an unsupervised way. Later, Chaidaroon et al. [6] expanded their model and proposed NbrReg. It forces the hash codes to be able to reconstruct unique words occurring in both the text and its neighbors in the original text space (found using BM25 [60]). Both of them achieve improvement but cannot generate hash codes end-to-end. To obtain hash codes directly, **Neural Architecture for Semantic Hashing** (**NASH**) [63] proposes an end-to-end trainable semantic hashing framework. NASH adopts a similar variational autoencoder architecture as VDSH but models the hash codes as Bernoulli latent-variable. Because this binarization is discrete and thus not differentiable, a straight-through estimator [83] is used when optimizing the model. Analogously, Doc2hash [90] is proposed to solve the gradient problem using the Gumbel-Softmax trick instead of using the straight-through estimator. Similar to NbrReg and STH, some recent works [23, 24] take advantage of neighborhood knowledge. **Ranking-based Semantic Hashing** (**RBSH**) [23] incorporates pseudo labels learned from weak supervised into the hash code generation. Before long, the authors of RBSH propose Semantic Hashing with **Pairwise Reconstruction** (**PairRec**) [24] to improve it by using a pairwise reconstruction object. Considering the relationships between documents, node2hash [7] first incorporate graph context into hash modeling to achieve better performance. Recently, Few-bits Semantic Hashing (WISH) [82] notices the severe information loss on short hash codes and leverages auxiliary implicit topic vectors to address it. The above works are based on a variational autoencoder architecture. Some of them [24, 63] use the Bernoulli latent variable to achieve code balance. While in [15], the authors employ an adversarial training procedure to the encoder-decoder framework to ensure code balance. However, current deep models achieve code balance from a local perspective, which is ineffective in large-scale databases.

Our work improves such studies for text-based semantic hashing as follows. First, different from [82] implicitly employing topic vectors to save information, we propose an explicit propagation relevance regularization to transfer the correlations among informative high-dimensional space to compact Hamming space, which sufficiently addresses the information loss with a clear goal. Second, we aim at maintaining code balance from a global perspective to generate highly efficient hash codes in large-scale corpus rather than local preserve like previous models [15, 23, 24, 63]. Third, text noise often occurs in practical applications, which seriously damages the performance of hash codes. However, most present works ignore this problem. Thus we introduce a noise-aware encoder architecture to extend the basic model and generate more robust hash codes.

2.2 Characteristics of Hash Codes

Compared with other types of data structures like the kd-tree [65], hash codes usually achieve significant improvement in storage and search efficiency [25], whose speed is almost independent of the number of data points [52]. In the similar search task, most related works [6, 8, 15, 23, 24, 63, 82] use a brute-force linear scan of all the hash codes to find desired results. While the highly efficient Hamming distance enables large-scale search using linear scans [62], significantly faster alternatives exist for the recall stage. A quick way to find all r-neighbors of a hash code query is to use a hash table and examine all hash buckets whose indices are within r bits of the query. For hash codes of b bits, the number of hash buckets to be examined is called lookups and can be formulated as

$$lookups(r, b) = \sum_{k=0}^{r} \binom{b}{k}. \tag{1}$$

A major drawback of this method is that the search efficiency will break down for long hash codes since the lookups will explosion with the code length b increases [18]. One way to solve this problem is restricting a search to exact matches (i.e., $r = 0$), thus avoiding the need for Hamming ball exploration [25]. However, it is not a standard method for semantic hashing because it ignores the semantics between nearby hash codes. Another way is to use multi-index hashing [52], which splits the hash codes into m substrings and builds hash tables for each substring. This way can significantly reduce the lookups but is also related to code length b:

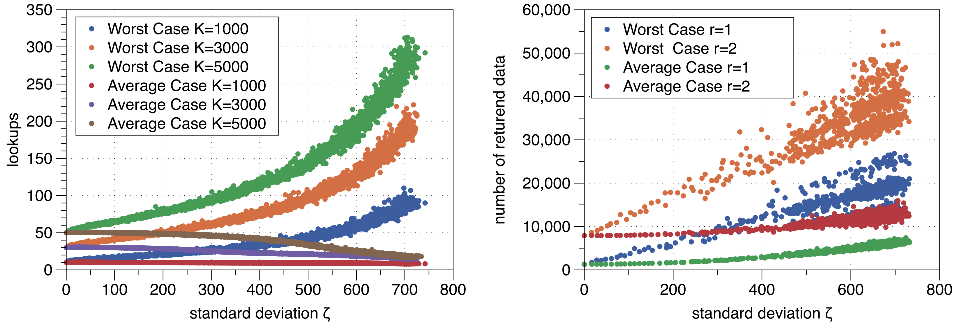
$$lookups(r, b, m) \leq m2^{H(\frac{r}{b})\frac{b}{m}}, \tag{2}$$

where $H(a) \equiv -alog_2 a - (1-x)log_2(1-a)$ is the entropy of a Bernoulli distribution with probability a. When m increases, we must consider the candidates' test time, which is practically equivalent to ranking stage cost time in our scenario. For example, if we set $m = b$, the returned texts will include the entire database, making the search procedure ineffective. Thus we cannot infinitely increase m. When fixing m and increasing the code length b, it still leads to a rapid expansion of lookups. Therefore, generating compact hash codes with short bits is a crucial target for semantic hashing in practical applications [72].

Besides, code balance is a desired characteristic for hash codes to promote search efficiency in bad cases. To illustrate the importance of code balance on both search problems more clearly, we conducted experiments using simulation data. Figure 3(a) and (b) depict two concrete examples for both KNN search and PLEB search in the simulation data. We use ζ, the standard deviation of the number of texts in all hash buckets, to reflect the degree of code balance. In general, the smaller the standard deviation ζ, the higher the degree of code balance, which means the more uniformly distributed texts are in the Hamming space. Figure 3(a) illustrates the average and worst lookups (the accessed number of hash buckets) in different data distributions to reach exact K texts in KNN search. The result demonstrates that the worst-case lookups are 15 times more than the average case in a poor code balance situation. Such a huge gap between the worst and average cases will bring users a bad experience, even when the average lookups decrease. In the PLEB search, each query has the same lookups using the same search radius. However, the number of returned texts greatly impacts the ranking stage. Figure 3(b) illustrates the average and worst returned texts in PLEB search in different search radius r. We find that code balance reduces the returned texts of worst and average cases, alleviating the time complexity of the ranking stage. Thus code balance is a crucial factor for generating efficient hash codes.

3 PRELIMINARIES

In this section, we formally present our problem of semantic hashing and similar text search problem in the recall stage.



(a) KNN search

(b) PLEB search

Fig. 3. A synthetic example of 409,600 (100×2^{12}) data points assigned to 4,096 (2^{12}) hash buckets in 12-bit Hamming space. We use the standard deviation ζ of the number of texts in all hash buckets to evaluate the degree of code balance. A search procedure is conducted on the synthetic data with KNN search and PLEB search, respectively. (a) plots the average and worst number of access hash buckets in KNN search. (b) shows the average and worst number of returned texts in the PLEB search. This result shows that efficient hash codes should achieve code balance to mitigate bad search cases.

3.1 Semantic Hashing

In general, there is a text database $X = \{x_1, x_2, \ldots, x_N\}$ where x_i is the ith text and N is the number of texts. Semantic hashing aims at learning a hashing function $f(x) : x \rightarrow s$, which can be implemented by complex algorithms or models, to project the original text x into a binary representation s. We term the representation $s \in \{-1, 1\}^b$ as hash code, and it falls into a corresponding hash bucket, where the notation b is the dimension (or cardinality) of the hash code s. Then Hamming distance is used as the distance metric to evaluate the similarities between two hash codes, which is defined as the number of different bits between two codes:

$$d_{Hamming}(s_1, s_2) = \sum_{i=1}^{b} 1_{s_{1,i} \neq s_{2,i}} = SUM(s_1\, XOR\, s_2). \tag{3}$$

Since learning a discrete-output function $f(x)$ in the deep network is intractable, whose gradient always vanishes in the *sign* function, a convenient way is to learn a continuous function instead, from which s is obtained by using the median method [78]. Specifically, we first compute the median value of the semantic vector in the training set as the threshold. Then we set the kth element of s to 1 if the kth dimension of s is larger than the threshold; otherwise, we set the kth bit to −1.

3.2 Similar Search Problem in Recall Stage

In the similar text search problem, given a query text q, our target is to find a subset of texts $Set(q)$ from the database. We should make as many texts x in set $Set(q)$ as possible to satisfy a similarity score $sim(q, y) \leq sim(q, x)$, where $y \in \{X - Set(q)\}$.

While using semantic hashing in the recall stage, there are two phases, including indexing phase and search phase. In the indexing phase, texts set X are indexed in the database by using a hashing function $f(x)$ to extract hash codes as representation. In the search phase, the system uses the hashing function $f(x)$ to extract the hash code from a query text q. Then it uses the code of the query as the index to access hash buckets and find candidate texts. In this setting, we have two search problems according to the termination conditions, including KNN search and PLEB search.

Table 1. NOTATION

Notation	Description
D_x	The data distribution
X	The dataset of texts
N	The number of texts in X
V	The set of vocabulary
M	The external memory
$\hat{M}$	The set of selected hash codes from external memory
$\mathcal{P}(x)$	The corrupt texts set of x
x_i	The i-th text in dataset X
$x, \hat{x}, x'$	The original,corrupted and reconstruction text
$x^{(i)}, \hat{x}^{(i)}$	The actual and predicted words at position i in x
w_i	The i-th word in vocabulary V
s	The hash code of x
$\hat{s}$	The latent representation of x
l	The high-dimensional representation of x
$\mathcal{B}$	The training mini-batch
M_i	The i-th slot in external memory M
v_i	The i-th timer corresponds to M_i
K	The number of returned texts in KNN search
r	The search radius of the Hamming-ball
b	The hash code length
$\lvert \cdot \rvert$	The length of a representation or the number of a set
θ_E, θ_D	Parameters of the encoder and the decoder

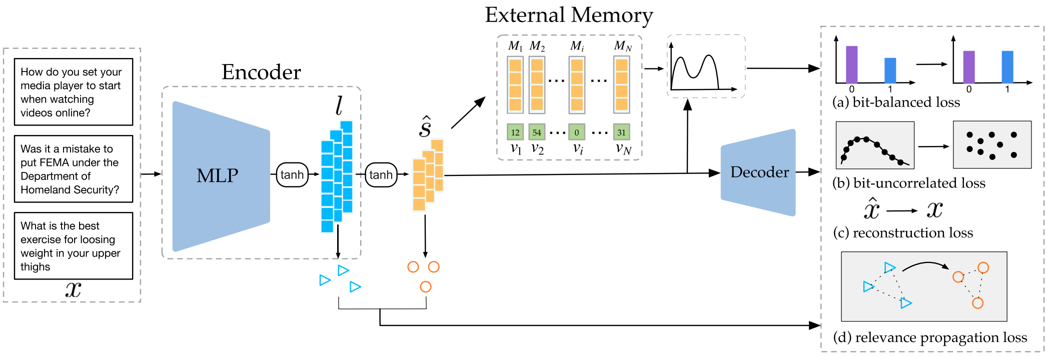
Specifically, Given a set of hash codes $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$, KNN search is to find exact K codes in $\mathcal{S}$ that are closest in Hamming distance to a given query. While in the PLEB search, we pre-define a search distance and then find all codes in $\mathcal{S}$ within the distance of a query. The notations used in this paper are given in Table 1.

4 MEMORY-BASED SEMANTIC HASHING

In this section, we will introduce the MASH model. The overview of the proposed Memory-Based Semantic Hashing model is presented in Figure 4, which is an encoder-decoder framework and consists of two auxiliary components. (1) A relevance propagation regularization aims at tackling the information loss in short hash codes, where we design an objective for the encoder network to propagate the correlations between pairs of texts from high-dimensional informative space to compact Hamming space. (2) An external memory storing hash codes of all texts during the training process aims at achieving the code balance objective from a global perspective. In the following subsections, we will explain the details of each part of the MASH framework.

4.1 Encoder-Decoder based Semantic Hashing Framework

As illustrated in Figure 4, we use an encoder-decoder framework to generate hash codes from the texts in an unsupervised way. Given a dataset $X = \{x | x \sim D_x\}$, where D_x is the data distribution, the encoder-decoder framework is defined as an encoding function $E(x, \theta_E) : x \rightarrow z$ that maps each text x into a latent representation z with the parameter θ_E, and a decoding function $D(x, \theta_D) : z \rightarrow \hat{x}$ that reconstructs the latent representation z to $\hat{x}$ with the parameter θ_D. Considering $x = \sum_i x^{(i)}$ is the bag-of-words representation for text x and $x^{(i)}$ is the actual word at position i in x, we write


How do you set your media player to start when watching videos online?

Was it a mistake to put FEMA under the Department of Homeland Security?

What is the best exercise for loosing weight in your upper thighs

Fig. 4. The overview of the MASH model.

the reconstruction loss function of all texts as word-wise negative likelihood:

$$L_{rec} = E_{x \sim D_x} \left[\frac{1}{|V|} \sum_{i=1}^{|V|} -\log p(\hat{x}^{(i)} = x^{(i)}) \right], \tag{4}$$

where $|V|$ is the number of word in the vocabulary set V and $\hat{x}^{(i)}$ corresponds the ith word in the reconstructed text $\hat{x}$.

In deep semantic hashing, the encoding function $E(x, \theta_E)$ is the hashing function $f(x) : x \rightarrow s$ aiming at generating hash code s from the text x. However, as mentioned before, learning a discrete-output function directly is an intractable problem because the discrete layer is usually incompatible with the backward propagation and thus causes a gradient flow problem [15]. We instead learn a continuous function $\hat{s} = f(x) \in [-1, 1]^b$ for convenience. Concretely, we use the smooth function $tanh(a) = \frac{2}{1+e^{-2a}} - 1$ as the activation function to approximate binary representation. Then the hash code s is obtained by using median method [78]. Note that the encoding function $f(x)$ can be specified with several models like RNN [10] and CNN [88]. We do not emphasize their differences and implement it by MLP architecture to make a fair comparison with previous works [6, 8, 23, 24, 63, 82]. Besides, the decoder function $D(s, \theta_E)$ is to reconstruct the binary-like representation $\hat{s}$ as $\hat{x}$. We use a softmax layer to predict discrete words. Then, the decoder can be defined as

$$p(x^{(i)} = w|\hat{s}) = \frac{exp(\hat{s}^T W x^{(w)} + b_w)}{\sum_{j=1}^{|V|} exp(\hat{s}^T W x^{(j)} + b_j)}, \tag{5}$$

where $p(x^{(i)} = w|\hat{s})$ is the predicted probability of each word $w \in V$ with parameter $\theta_E = \{W, b_1, b_2, \ldots, b_{|V|}\}$ and $x^{(w)}$ is the one-hot representation of word w. Here, $W \in R^{b \times |V|}$ can be interpreted as a word embedding matrix to be learned, and $\{b_i\}_{i=1}^{|V|}$ denote bias terms.

4.2 Relevance Propagation Regularization

Here, we have obtained the hash code through the encoder network. As discussed in Section 2.2, efficient hash codes should be as short as possible. However, when we set a short length for hash code s, the original encoder network cannot handle severe information loss in s. Some related works [8, 24, 63] show that in a suitable range, with a growing number of dimension, the quality of binary representations (hash codes) increases but it will lead to higher search latency. Therefore, it inspires us to utilize high-quality information in high-dimensional representation.

Specifically, we introduce a relevance propagation objective to build a bridge between high-dimensional representation l and short binary-like representation $\hat{s}$ by using a regularization loss function, which is equivalent to propagating Hamming distances between two layers in the encoder network. Any of the intermediate representations in the encoder network can be chosen as the high-dimensional representation l, where we choose the representation before the short latent representation $\hat{s}$ as Figure 4 illustrates. In order to make l and $\hat{s}$ comparable, we use the same activation function *tanh* for l. Another referential property is the hamming distance between two binary vector $a_1, a_2 \in \{-1, 1\}^b$ of the same length can be written as $d_h(a_1, a_2) = -0.5(a_1^T a_2 - |a_1|)$. Therefore, remote hash codes are characterized by low values and close hash codes are characterized by high values. Thus we define the relevance propagation objective as follows:

$$L_{rp} = \frac{1}{|\mathcal{B}|(|\mathcal{B}| - 1)} \sum_{k,j=1,k\neq j}^{|\mathcal{B}|} \left| \frac{l_k l_j}{|l|} - \frac{\hat{s}_k \hat{s}_j}{|\hat{s}|} \right|, \tag{6}$$

where the notation $\mathcal{B}$ is a mini-batch with $|\mathcal{B}|$ texts in it, and $|l|$ and $|\hat{s}|$ is the dimension of l and $\hat{s}$ respectively, where we usually ensure $|l| \gg |\hat{s}|$. In terms of the optimization procedure, we guarantee that texts nearby in high-dimensional Hamming space are also nearby in latent Hamming space. We normalize Hamming distances between $d_h(l_k, l_j)$ and $d_h(\hat{s}_k, \hat{s}_j)$ by dividing the corresponding representation dimensions to make them comparable. In general, the role of the relevance propagation objective is to map the relevance between embeddings from the informative high-dimensional space to the compact Hamming space.

4.3 External Memory

Code balance is a crucial factor in generating highly effective hash codes as analyzed in Section 2.2, which means hash codes should be evenly spread throughout the entire Hamming space and alleviates the search time delay. However, previous deep models [15, 23, 24, 63, 82] only optimize them from a local perspective and ensure them in a small subset of texts (e.g., in a mini-batch), which are not suitable for large-scale databases. To address this problem, we propose an external memory $M \in \{-1, 1\}^{N\times b}$ in MASH to record all hash codes of N texts during the training process. These stored hash codes describe the entire distribution and we will use them to adjust the distribution of new hash codes in the training procedure. Theoretically, the external memory does not occupy many GPU or CPU memory due to the storage efficiency of the hash code. Next, we first introduce the updated details of the external memory, then describe the optimization of new hash codes.

Figure 4 illustrates the details of external memory. Each text x has its storage location. For example, the hash code s_i generated from the ith training text x_i will be stored in the external memory's ith slot M_i. Therefore, we can view all N training texts globally rather than only select a subset of texts to optimize. A natural way is to use all stored hash codes to evaluate the current distribution. However, some hash codes stored in the external memory will become stale with the training process because of substantial training texts and the finite batch size. If we use all hash codes in the external memory, the optimization may be biased. Although we can update all hash codes in the external memory after each training step, it will cause intolerable time costs. Thus we need to choose "fresh" hash codes in the memory before optimizing the code-balance objective.

Intuitively, there are two crucial factors in identifying which hash code should be chosen. From the individual perspective, more recently updated hash codes are more likely to be "fresh", so we should choose the lately modernized hash codes in the external memory. From the integrated perspective, if the network's parameters have little evolution in the current training step, most of the texts' hash codes will remain unchanged. In general, at the beginning of the training process, the network parameters change fast with a large loss and most of the hash codes in the external memory become stale. As the update progresses of the network, the parameters become consistent with small loss and most of the texts' hash codes will remain unchanged, which means more stored hash codes are “fresh”. Based on the above ideas, we design a heuristic method to choose “fresh” hash codes. Specifically, we assign a timer v_i to each slot M_i in the external memory. At the beginning of tth training step, each timer will be updated by $v^{(t)} = v^{(t-1)} + 1$. Then we choose M_i if $v_i^{(t)}$ satisfy following inequality:

$$v_i^{(t)} \leq \gamma \frac{|L_{max} - L_{t-1}|}{|L_{max}|} * \frac{N}{|\mathcal{B}|}, \tag{7}$$

where L_{t-1} is the loss in time $t-1$, L_{max} is the largest loss in the training history, and $\gamma \in (0, 1]$ is a hyper-parameter. We use the first term $\frac{|L_{max}-L_{t-1}|}{L_{max}}$ to measure the changes of network's parameters and the second term $\frac{N}{|\mathcal{B}|}$ is equivalent to the total amount of timer increments in a training epoch. In backward propagation, we update the ith timer $v_i^{(t)} = 0$ if $x_i \in \mathcal{B}$ and correspondingly update the ith slot in the external memory by $M_i = sign(s_i)$, where $sign$ is the signum function.

After choosing the “fresh” hash codes matrix $\hat{M} = \{\hat{M}_1, \hat{M}_2, \ldots, \hat{M}_{|\hat{M}|}\} \in \{-1, 1\}^{|\hat{M}| \times b}$, where $|\hat{M}|$ is the number of selected hash codes, MASH model begins to optimize code balance from a global perspective. Technically, code balance includes two optimization targets: *bit-balanced* and *bit-uncorrelated* [78].

First, *bit-balanced* means each bit has the same probability of appearing. To achieve the goal of bit-balanced, each bit in the hash code should have a 50% chance of being 1 or −1. Thus we introduce the bit-balanced loss function as follows:

$$L_{bb} = \frac{1}{b} \sum_{j=1}^{b} \alpha_j \left| \sum_{i=1}^{|\mathcal{B}|} s_{ij} \right|, \tag{8}$$

where s_{ij} is the jth bit value of ith hash codes in a training batch $\mathcal{B}$ and α_j is a weight coefficient assigned to each bit:

$$\alpha_j = \frac{exp(|\sum_{i=1}^{|\hat{M}|} \hat{M}_{ij}|)}{\sum_{j=1}^{b} exp(|\sum_{i=1}^{|\hat{M}|} \hat{M}_{ij}|)}, \tag{9}$$

where $\hat{M}_{ij}$ is the jth bit value of ith hash codes in matrix $\hat{M}$. This regularization function can be understood as making the mathematical expectation of each bit in the training batch zero with an important weight. The weight α_j reflects the degree of each bit's bit-balanced in the selected hash codes from the external memory. Suppose the mathematical expectation of a bit in the external memory deviates from zero. In that case, a relatively large penalty term will be added to the corresponding bit in the current training batch and guides the new hash codes to the desired distribution from a global perspective.

Second, *bit-uncorrelated* means different bits should be as irrelevant as possible. We implement orthogonal on our model to achieve this goal. To be specific, the premise of the orthogonal method is that if two variables are orthogonal and the expectation of any one of them is zero, then they are irrelevant. For convenience, we assume $S = \{\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_{|\mathcal{B}|}\} \in R^{|\mathcal{B}| \times b}$ is the matrix of short binary-like representation generated from mini-batch $\mathcal{B}$. Similarly, we also add a weight coefficient A in the following regularization function:

$$L_{bd} = \frac{1}{b^2} \left\| A \cdot \left(\frac{S^T S}{|\mathcal{B}|} - I \right) \right\|_F^2, \tag{10}$$

where I is identity matrix, $|| \cdot ||_F$ denotes frobenius norm and A is defined as follow:

$$A = softmax\left(\left| \frac{\hat{M}^T \hat{M}}{|\hat{M}|} - I \right| \right). \tag{11}$$

With these two objectives, MASH can optimize code balance from a global perspective. Furthermore, there is a training trick for external memory. In the early training, because of extensive updates, the hash code in external memory is unstable and the loss values usually dramatically change. Using such hash codes to measure distribution is meaningless and the loss value may be tremendous to make $\frac{|L_{max}-L_{t-1}|}{L_{max}}$ always be approximately 1 in later training. Thus we need to warm up the external memory first. Precisely, at the beginning of training, we do not compute the bit-balanced objective L_{bb}, bit-uncorrelated objective L_{bd} and do not record the largest loss value but only update timers and slots.

4.4 Model Objective Function

By combining the bit-balance loss function and bit-uncorrelated loss function, the objective function Equation (4) can be transformed into the final formulation:

$$L_{final} = L_{rec} + \beta_1 L_{rp} + \beta_2 L_{bb} + \beta_3 L_{bd}, \tag{12}$$

where β_1, β_2 and β_3 are the hyper-parameters. By the values of β_2 and β_3, MASH can balance the efficiency and effectiveness of retrieval in the different practical application situations. We summarize the training algorithm of MASH in Algorithm 1.

ALGORITHM 1: Parameter Learning of MASH

```
Initialization: $\theta_E, \theta_D, M, L_{max}$ ; $v_k \leftarrow \frac{N}{|\mathcal{B}|}$
repeat
    draw a mini-batch $\mathcal{B}$ from $X$
    Update all $v_i \in \{v_1, v_2, \ldots, v_N\}$ by $v_i = v_i + 1$
    $\hat{M} \leftarrow \{\}$
    for each $i \in \{1, 2, \ldots, N\}$ do
        if $v_i$ satisfy Equation (7) then
            $\hat{M} \leftarrow \hat{M} \cup \{M_i\}$
        end if
    end for
    Update $\theta_E, \theta_D$ by minimizing $L_{final}$ in Equation (12)
    if $L_{final} > L_{max}$ then
        $L_{max} \leftarrow L_{final}$
    end if
    for each $i \in \{1, 2, \ldots, N\}$ do
        if $x_i \in \mathcal{B}$ then
            $M_i \leftarrow sign(s_i)$
            $v_i \leftarrow 0$
        end if
    end for
until converged
Output: $\{\theta_E\}$ parameters of the encoder
```

5 DENOISING MEMORY BASED SEMANTIC HASHING

MASH can effectively deal with the problem of information loss in short hash codes and generate uniformly distributed hash codes. It is a general framework for many applications. However, in the recall scenario, a large of textual data contains various types of noise such as misspelling,

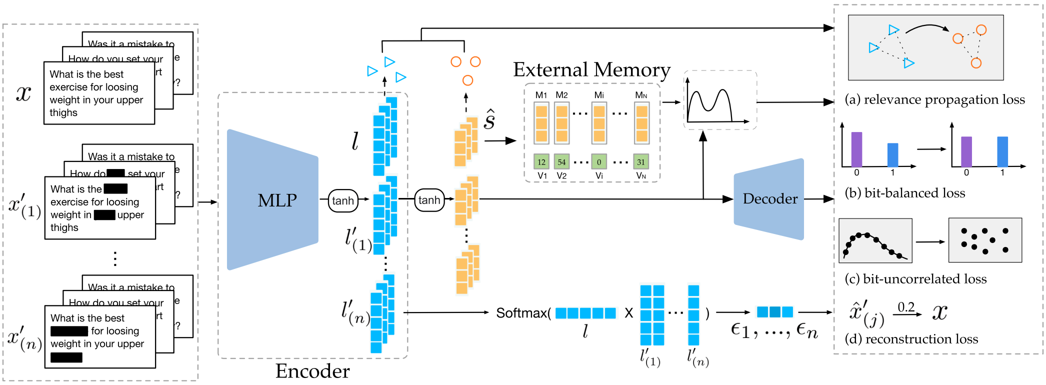


Fig. 5. The overview of SMASH model.

non-standard abbreviations, and E-speak & new words and so on [3, 22, 40], as shown in Figure 2. Although we can clean the noisy text in the database, the noise can also be introduced by user queries (e.g., misspelling) [11]. For example, a user may misspell "exercise" for "exercuse" in the query text. In general, "exercuse" is absent in standard dictionaries and causes semantics deviation, damaging the retrieval performance. Therefore, to alleviate this problem and improve the performance of our MASH model in the noisy scenario, we further consider the robustness of hash codes. Intuitively, when a text includes non-dictionary words, it should still fall into the same or close hash bucket as the clean one. Based on the above consideration, in this section, we further address the problem of noise texts by generating robust hash codes. We extend the current MASH framework and propose a SMASH framework by incorporating a noise-aware encoder-decoder framework in our model as illustrated in Figure 5. In the following subsections, we will explain the details of each part in the SMASH framework.

5.1 Noise-aware Encoder-decoder Framework

To generate robust hash codes, the model targets to reconstruct the corrupted text x' as the clean one x. A straightforward way is randomly removing each word in the text x with a pre-defined noise rate p_n, which is the fraction of the words of the input texts that need to be removed [15]. Then, suppose the reconstructed text is $\hat{x}'$, the reconstruction loss Equation (4) can be rewritten as follows:

$$L_{rec} = E_{x \sim D_x}\left[\frac{1}{|V|}\sum_{i=1}^{|V|} -\log p(\hat{x}'^{(i)} = x^{(i)})\right], \tag{13}$$

where $\hat{x}'^{(i)}$ corresponds the ith word in the reconstructed text $\hat{x}'$. This method is also known to help learn a smooth, low-dimensional manifold of the data [5].

However, two problems will affect the performance and convenience of that framework. First, we argue that not each removed word has the same importance in a sequence, leading to various semantic changes in each corrupt text. For example, the original text is *"What is the best exercise for losing weight in your upper thighs"*. We randomly remove the word and may get a new text "What is the ### exercise for losing weight in ### upper thighs", which without changing the original semantic practically and should be put into the same hash bucket. However, the corrupt text also could be "What is the best ### for losing ### in your upper thighs". In this scenario, the semantics of this new text are far from the original one because we do not know what the subject of this question is? For example, it may be surgery for losing the fat in the upper thighs. Second, the choice of noise value p_n for all texts has a significant impact on performance, which is a time-consuming process because of the manual setting and we need to fine-tune it for different datasets.

To tackle the first problem, we propose to assign a coefficient to each text in the reconstruction loss, which makes each corrupt text created by the same text has different impacts on the model's update. Specifically, for the original text x, we construct its corrupted text set $\mathcal{P}(x) = \{x'_{(1)}, x'_{(2)}, \ldots, x'_{(n)}\}$, where $x'_{(j)}$ is the jth corrupted text. Each corrupted text $x'_{(j)}$ is created by a specific noise rate, which we will discuss later. In the training procedure, we input the original text x and corrupted text set $\mathcal{P}(x)$ to the encoder and will get corresponding high-dimensional representation l and $\mathcal{P}(l) = \{l'_{(1)}, l'_{(2)}, \ldots, l'_{(n)}\}$. Considering the high-dimensional representation can be used as the indicator in the Hamming space and reflect the semantic similarity between two texts, we calculate the semantic similarity coefficient as follows:

$$\epsilon_j = \frac{exp(l^T l'_{(j)})}{\sum_{k=1}^{n} exp(l^T l'_{(k)})}, \tag{14}$$

where ϵ_j can also be view as a latent weight variable for the jth text. Thus we use the latent weight variable ϵ_j to rewrite the reconstruction loss Equation (4) as follows:

$$L'_{rec} = E_{x \sim D_x}\left[\frac{1}{|V|}\left(\sum_{j=1}^{n}\sum_{i=1}^{|V|} -\epsilon_j \log p\left(\hat{x}'^{(i)}_{(j)} = x^{(i)}\right)\right)\right]. \tag{15}$$

To solve the second problem, we intend to traverse various noise rates in the training process. An intuitive solution is to view the noise rate $\widetilde{p}_n$ from a Bayesian perspective and assume that it is a random variable obeying a given distribution q. Specifically, in our work, we suppose q as a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. The reason is if we set μ to be p_n and σ to be 0, then q degrades to a definite point, which is equivalent to setting p_n as a hyper-parameter. Therefore, our method generally covers traditional ways. Under this setting, the reconstruction loss function is as follows:

$$\begin{aligned} E_{\widetilde{p}_n \sim \mathrm{q}}[L'_{rec}] &= \int \mathrm{q}(\widetilde{p}_n) L'_{rec} \mathrm{d}\widetilde{p}_n \\ &= \int \mathrm{q}(\widetilde{p}_n) \int_{x \sim D_x} \frac{1}{|V|} \sum_{j=1}^{n}\sum_{i=1}^{|V|} -\epsilon_j \log p\left(\hat{x}'^{(i)}_{(j)} = x^{(i)}\right) \mathrm{d}x \mathrm{d}\widetilde{p}_n \\ &= \int_{x \sim D_x} \int_{\widetilde{p}_n \sim \mathrm{q}} \frac{1}{|V|} \sum_{j=1}^{n}\sum_{i=1}^{|V|} -\epsilon_j \log p\left(\hat{x}'^{(i)}_{(j)} = x^{(i)}\right) \mathrm{d}\widetilde{p}_n \mathrm{d}x. \end{aligned} \tag{16}$$

However, Equation (16) is intractable due to the integration of $\widetilde{p}_n$. To tackle this problem, we sample $\widetilde{p}_n$ from the distribution q for each text x to create noisy text x', which means $x' = \widetilde{p}_n(x)$. It should be noticed that we may have different $\widetilde{p}_n$, rather than setting the same rate for different texts, which is capable of dealing with different training samples independently and thus brings better generalization ability to our model. To be more specific, in each training step, we sample the noise rate for each text in every training step. The reconstruction loss function turns to:

$$L'_{rec} = E_{x' = \widetilde{p}_n(x), x \sim D_x}\left[\frac{1}{|V|} \sum_{j=1}^{n}\sum_{i=1}^{|V|} -\epsilon_j \log\left(\hat{x}'^{(i)}_{(j)} = x^{(i)}\right)\right]. \tag{17}$$

Please note that although the obtained corrupt texts can be used as supplementary training samples, we only use them to optimize the reconstruction loss Equation (17) to generate robust hash codes and optimize relevance propagation regularization Equation (6) to propagate relevance between representations. The bit-balanced loss Equation (8) and bit-uncorrelated loss Equation (10) are still optimized by the original text because the external memory is used to analyze the distribution of original texts' hash codes. We summarize the training algorithm of SMASH in Algorithm 2.

ALGORITHM 2: Parameter Learning of SMASH

```
Initialization: $\theta_E, \theta_D, M, L_{max}; v_k \leftarrow \frac{N}{|\mathcal{B}|}$
repeat
    draw a mini-batch $\mathcal{B}$ from $X$
    create the corrupt text set $\mathcal{P}(x)$ for each $x \in \mathcal{B}$ with noise rate $\widetilde{p}_n \sim \mathcal{N}(\mu, \epsilon)$
    Update all $v_i \in \{v_1, v_2, \ldots, v_N\}$ by $v_i = v_i + 1$
    $\hat{M} \leftarrow \{\}$
    for each $i \in \{1, 2, \ldots, N\}$ do
        if $v_i$ satisfy Equation (7) then
            $\hat{M} \leftarrow \hat{M} \cup \{M_i\}$
        end if
    end for
    Update $\theta_E, \theta_D$ by minimizing Equation (4), Equation (6), Equation (8), Equation (10) with $\mathcal{B}$
    Update $\theta_E$ by minimizing Equation (17), Equation (6) with $\mathcal{P}(x)$
    if $L_{final} > L_{max}$ then
        $L_{max} \leftarrow L_{final}$
    end if
    for each $i \in \{1, 2, \ldots, N\}$ do
        if $x_i \in \mathcal{B}$ then
            $M_i \leftarrow sign(s_i)$
            $v_i \leftarrow 0$
        end if
    end for
until converged
Output: $\{\theta_E\}$ parameters of the encoder
```

6 EXPERIMTENTS

In this section, we conduct extensive experiments to validate the performance of our proposed MASH and SMASH frameworks. Specifically, we first describe the datasets and introduce the experimental setup (Sections 6.1 and 6.2). Then, we demonstrate the effectiveness and efficiency of our models on three real-world datasets (Sections 6.3 and 6.4). At last, we provide further analyses on different aspects, including code balance (Section 6.5), external memory (Section 6.7), text noise (Section 6.6), and Hamming space visualization (Section 6.8).

6.1 Datasets

We utilize the following three datasets from different domains for evaluation: searching similar questions in CQA (community-based question answering) sites, finding similar exercises in the educational platform, and searching similar news.

- Yahooanswer.[3] It is a large collection of 1,460,000 questions obtained through the Yahoo! Research Alliance Webscope program with the ten largest main categories. We use it as the coarse-grained semantic dataset.

[3] https://www.kaggle.com/soumikrakshit/yahoo-answers-dataset.

Table 2. The Statics of the three Datasets

Dataset	Yahooanswer	EduMath	20Newsgroups
domain	CQA	Education	News
# of texts	1,460,000	748,554	18,828
# of classes	10	175	20
average of text length	10.8	69.2	206.2
word error rate	0.552%	0.251%	0.418%

Table 3. A Practical Example of Noisy Text in Yahooanswer Dataset

Noisy text example	Classes of OOV Words
What is the difference betwen a close knit family and extended family?	misspelling
What du think abt joze moriniyo (chelsea coach)	abbreviation
I have a desktop with a 82845 GL Intel integrated video card 64MB w/ no AGP slot. Should I upgrade via PCI?	E-speak & new words

- EduMath. This dataset collects extensive mathematical exercises for Chinese high school students from an authoritative institution mainly for education. It contains 748,554 real-world math exercises with 85 knowledge concepts,[4] which are seen as classes. We use it as the fine-grained semantic dataset.
- 20Newsgroups.[5] This dataset collects 18,828 newsgroup posts, partitioned into 20 different newsgroups/categories. It has become a popular dataset for experiments in text applications of machine learning techniques, and we use it as the small texts dataset.

Note that three datasets come from real world and contain various text noise. We utilize pycorrector,[6] which is a text error correction tool, to roughly evaluate the word error rate [35] in these datasets. Tables 2 and 3 respectively present the basic statistics of three datasets and an interpretation example about noisy text in Yahooanswer dataset.

For the three datasets, we split them into three subsets with roughly 80% for training, 10% for validation, and 10% for the test. Besides, we filter all texts by removing short texts with less than 6, 20, and 100 words and long ones with more than 100, 200, and 500 words in Yahooanswer, EduMath and 20Newsgroups datasets respectively. In the EduMath dataset, we utilize jieba[7] Chinese text segmentation tool to get the word list. Moreover, we apply stopword removal using the NLTK stopword list[8] for Yahooanswer and 20Newsgroups datasets. The used datasets are originally created for text classification, but we can define two texts to be similar if they have the same class label. This is the only time that the class labels are used. The definition of similarity is also used by related works [6, 8, 15, 23, 24, 63, 82].

6.2 Experimental Setup

6.2.1 Settings of Short Code Length. In our work, we focus on generating short hash codes. However, there is no clear definition of the short hash code currently. With the efficiency target, we consider that the code length should ensure that exploring the Hamming space by increasing

[4] knowledge concepts are previously labeled (e.g., in ASSISTments [56]).
[5] https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html.
[6] https://github.com/shibing624/pycorrector.
[7] https://github.com/fxsjy/jieba.
[8] https://www.nltk.org/nltk_data/.

the search radius r around the query faster than the linear scan. That means the number of hash buckets 2^b should be less than the number of texts N in the database. Besides, the Hamming space should ensure that different semantics have unique codes. Based on the above understanding, we think if the code length b satisfy:

$$log_2 c \leq b \leq log_2 N, \tag{18}$$

where c is the dataset's semantic (e.g., class) number, we say it is a short hash code. In our experiment, we set different datasets with a corresponding code length according to this definition. Specifically, we choose code length in $\{16, 18\}$ for Yahooanswer and EduMath, $\{10, 12\}$ for 20Newsgroups in the following experiments.

6.2.2 Settings of MASH and SMASH. There are some hyper-parameters in our MASH and SMASH models, and they should be initialized in the experiment. Similar to almost previous works [6, 8, 23, 24, 63, 82], our model use two **Rectified Linear Unit** (**ReLU**) layers of 1,000 hidden nodes as the first two layers of the encoder, then followed by a layer of 128 hidden nodes with $tanh$ activation to get the high-dimensional representation l and a layer of hash code length with $tanh$ activation to get the latent representation $\hat{s}$. We set the learning rate as 0.001 and consider the hyper-parameters β_1 in set $\{1, 5, 10\}$, β_2 in $\{0.001, 0.005, 0.01\}$ and β_3 in $\{0.001, 0.005, 0.01\}$ (Equation (12)). Besides, we set γ (Equation (7)), μ as 0.7 and 0, and choose σ in $\{0.2, 0.3\}$. We perform the grid search method on different datasets to get their best combination.[9]

6.2.3 Baseline Approaches. To compare the performance of our proposed MASH and SMASH models with other hashing methods, we borrow some baselines from various perspectives. The details of them are as follows:

- Spherical Hashing (**SPH**) [78]: SPH is a hypersphere-based space-partitioning shallow hashing method that preserves the similarity between texts.
- Self-taught Hashing (**STH**) [86]: STH is a two-component method that generates binary code via a k-nearest neighbor search and then learns hashing function by preserving the local similarities between samples.
- Variational Deep Semantic Hashing (**VDSH**) [8][10]: VDSH is a deep hashing method that learns the hashing function based on variational autoencoders.
- Neural Architecture for Semantic Hashing (**NASH**) [63][11]: NASH is also based on variational autoencoders but models the hash codes as a Bernoulli latent-variable and optimizes code balance from a local perspective.
- Neighborhood Recognition Model (**NbrReg**) [6][12]: NbrReg is a deep hashing method using supplementary neighborhood information to generate hash codes. Like STH, it mitigates the lack of labeled data by using an unsupervised ranking to approximate the true text space.
- Pairwise reconstruction based hashing (**PairRec**) [24][13]: PairRec is a weak supervised semantic hashing approach. It uses pairwise reconstruction loss to reconstruct the original text from the input text and its neighbor texts.
- Few-bits Semantic Hashing(**WISH**) [82][14]: WISH is a deep hashing method on few-bits hashing problem. It leverages auxiliary implicit topic vectors to generate compact hash codes.

[9] We make our code publicly available at https://github.com/hly1998/SMASH.
[10] https://github.com/bayesquant/VDSH.
[11] https://github.com/donggyukimc/nash.
[12] https://github.com/yfy-/nbrreg.
[13] https://github.com/casperhansen/PairRec.
[14] https://github.com/smartyfh/WISH.

Specifically, the chosen baselines are all widely used in the text semantic hashing area. Besides, we also design MASH w/o RP and MASH w/o EM for the ablation experiment. Specifically, MASH w/o RP means not using relevance propagation regularization but using the external memory to store hash codes and optimize the code-balance objective. On the contrary, MASH w/o EM means not using external memory but using relevance propagation regularization to solve the few-bits hashing problem.

In the following experiments, all models are implemented by using Pytorch. Specifically, VDSH, NbrReg, PairRec, and WISH use two layers of 1,000 hidden nodes for the encoder part. At the same time, NASH has two hidden layers with 500 units as suggested in [63]. We conduct all experiments on a Linux server with four 2.0 GHz Intel Xeon E5-2620 CPUs and a Tesla K80 GPU. For fairness, all parameters in these baselines are tuned to have the best performances.

6.3 Performance on Effectiveness

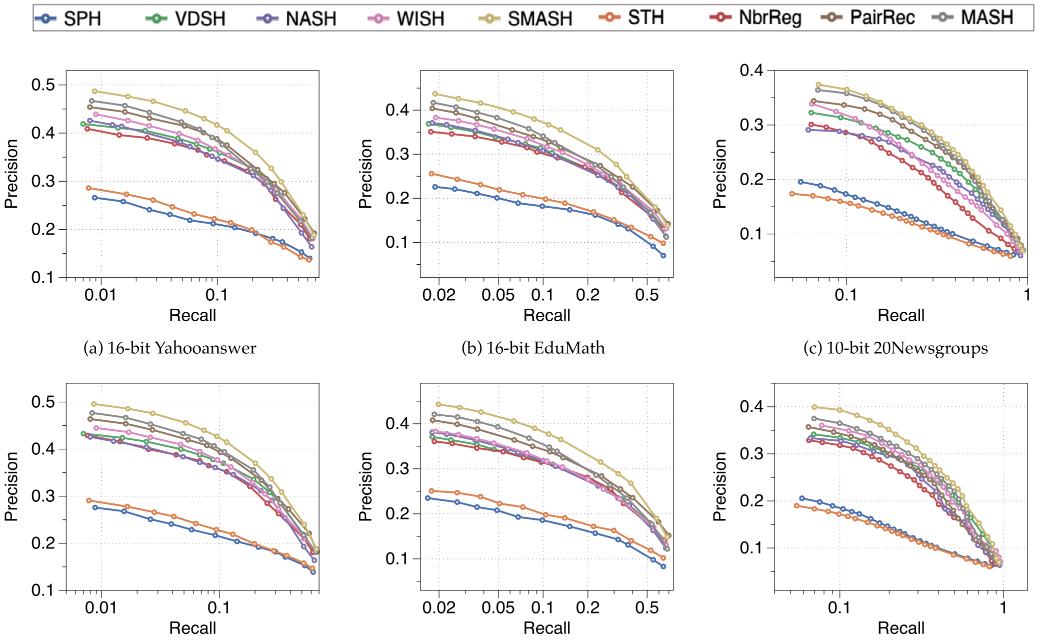
Effectiveness is a crucial factor in the recall stage, which aims at retrieving more relevant texts in the database while maintaining accuracy in the returned texts set. In this experiment, we use *Recall@K* and *Precision@K* as metrics to measure various semantic hashing models, which we define as:

$$Recall@K = \frac{Number\ of\ relevant\ texts\ in\ K\ retrieved\ texts}{Total\ number\ of\ all\ relevant\ texts}, \tag{19}$$

$$Precision@K = \frac{Number\ of\ relevant\ texts\ in\ K\ retrieved\ texts}{Total\ number\ of\ retrieved\ texts}. \tag{20}$$

We choose code length b in $\{16, 18\}$ for Yahooanswer and EduMath, $\{10, 12\}$ for 20Newsgroups in this experiment, and gradually increase the number of retrieval texts to plot the Precision-Recall curves. In Figure 6, we have several observations from these results. First, traditional shallow-hashing methods SPH and STH always have worse performance than other deep hashing methods, revealing the importance of deep semantic information in hash codes. Second, compared with other deep semantic hashing models, both MASH and SMASH clearly perform better on three datasets, followed by WISH. These phenomena prove the effectiveness of our proposed models by incorporating the factors of generating effective short hash codes. Third, SMASH consistently improves the retrieval quality compared to MASH in three real-world datasets, which include text noise. This phenomenon indicates that SMASH could handle text noise problems well by using the noise-aware encoder-decoder framework. A further study of noise will present in Section 6.6.

Besides, to further explore the effect of the different components in our model, we also conduct an ablation study with MASH w/o EM (MASH without external memory) and MASH w/o RP (MASH without relevance propagation regularization) models. Specifically, we set code length b as 16 for two large datasets Yahooanswer and EduMath to retrieve 1,000 candidates and code length as 10 for 20Newsgroups to retrieve 100 candidates. Then we use the Recall and Precision metrics for evaluation. As illustrated in Table 4, we find that compared with MASH and MASH w/o EM, MASH w/o RP gets poor performance, demonstrating that the relevance propagation objective can sufficiently address the few-bits hashing problem. Furthermore, MASH beats MASH w/o EM by additionally leveraging the external memory. We conjecture that the superior performance of MASH is due to the global optimization of code balance by using external memory, which reduces the redundancy among texts and a subsequent study about the de-redundancy ability will be discussed in Section 6.5.



(a) 16-bit Yahooanswer

(b) 16-bit EduMath

(c) 10-bit 20Newsgroups

(d) 18-bit Yahooanswer

(e) 18-bit EduMath

(f) 12-bit 20Newsgroups

Fig. 6. Precision-recall curves in three datasets using KNN search.

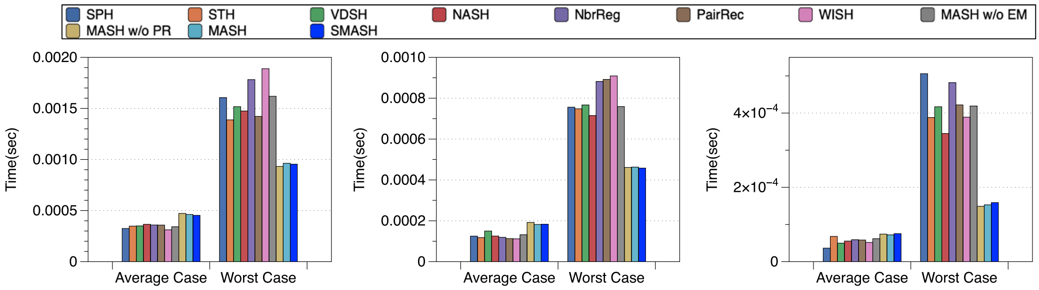
Table 4. Ablation Study of MASH in three Datasets

Model	Yahooanswer		EduMath		20Newsgroups	
	Recall	Precision	Recall	Precision	Recall	Precision
MASH w/o EM	0.026	0.428	0.033	0.375	0.066	0.348
MASH w/o RP	0.026	0.405	0.032	0.369	0.064	0.336
MASH	0.026	0.443	0.037	0.395	0.069	0.364
SMASH	**0.028**	**0.466**	**0.038**	**0.416**	**0.070**	**0.371**

6.4 Performance on Efficiency

In addition, efficiency is also a crucial metric for the recall stage given a query, including the average search case and the worst search case. Considering the different search termination conditions between KNN search and PLEB search, we conduct two experiments for each.

6.4.1 Efficiency on KNN Search. In the KNN search, we aim at retrieving the exact K nearest neighbor texts around the query. The time delay depends on the bucket examination in the search procedure. Here, we compare MASH and SMASH models with other deep semantic hashing models in retrieval time delay. Concretely, we set code length b as 16 for Yahooanswer and EduMath with K = 1,000 and code length b as 10 for 20Newsgroups with K = 100 to retrieve candidate texts and record the average and worst time cost per query. Figure 7 shows the average and worst time results for different models in three datasets, and we have some findings from them. First, we observe no apparent gap between these methods in average time, where MASH and SMASH are slightly worse than other baselines. However, we find an apparent distinction between our models and other baselines in the worst situation. Compared to other baselines, MASH and SMASH achieve about



(a) 16-bit Yahooanswer (b) 16-bit EduMath (c) 10-bit 20Newsgroups

Fig. 7. Average and worst search case in three datasets using KNN search.

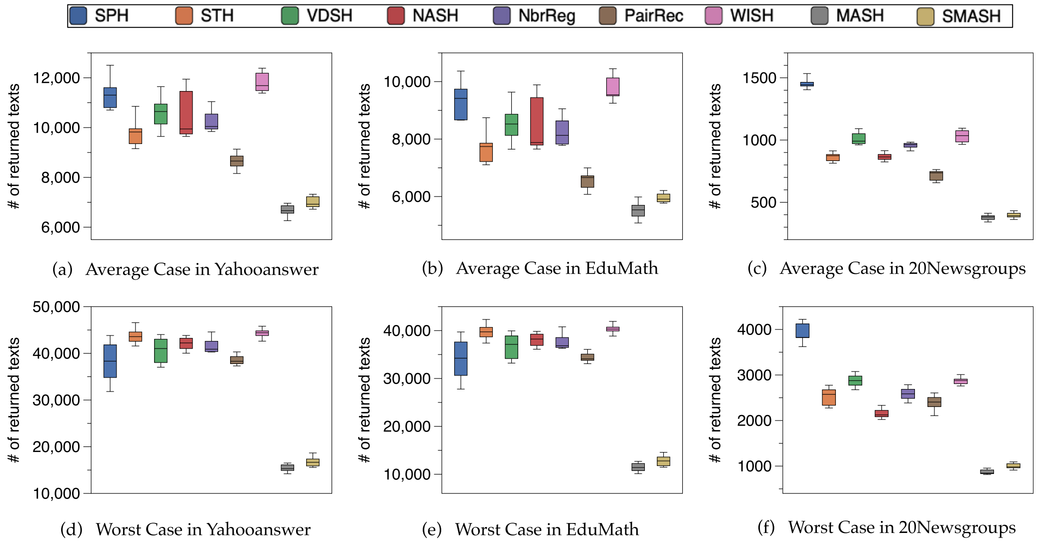
2–5 times more time-saving promotions on the worst search time. The results confirm the efficiency of our approaches. Second, compared with MASH and MASH w/o RP, MASH w/o EM cannot achieve good performance in the worst time case, which confirms the importance of generating a uniform distribution of data points as mentioned before. This uniform distribution benefits from the external memory, which estimates the distribution from a global perspective to achieve code balance. Furthermore, these results show a large gap between the average and worst cases in the search procedure. For example, the worst case takes 4–15 times as long as the average case, which has a harmful impact on users' experience. However, the gap between average and worst cases is only about two times in MASH and SMASH. Thus it is crucial to generate code-balance hash codes to relieve such issues.

6.4.2 *Efficiency on PLEB Search.* In PLEB search, we specify a Hamming search radius r and retrieve near neighbors. The search time is almost equal in the recall stage because we use the same search radius r in the PLEB search. However, as mentioned before, the number of returned texts will affect the efficiency of the ranking stage. Thus, we use the number of returned texts in the recall stage to measure the efficiency and do not care about the specific model implemented in the ranking stage. Expressly, we set code length b as 16 for Yahooanswer and EduMath, and 10 for 20Newsgroups to retrieve texts with a radius $r = 2$. We record the average and worst number of returned texts per query, whose result is shown in Figure 8. These figures show that our proposed models MASH and SMASH return fewer texts than selected baseline models, both in average and worst cases. These results mean our models will process faster than other baselines in the ranking stage. Recalling the synthetic analysis in Section 2.2, MASH and SMASH can achieve such good performance because they can generate code-balance hash codes. These results demonstrate the efficiency of our model in the realistic scenario from the perspective of the entire retrieval system.

6.5 Code Balance Evaluation

In the previous analysis, we evaluate the search effectiveness and efficiency of various semantic hashing models, where we find that code balance plays a significant role. To further study the code balance on different models, we choose MASH as a representative and other deep semantic hashing models to conduct the following experiments.

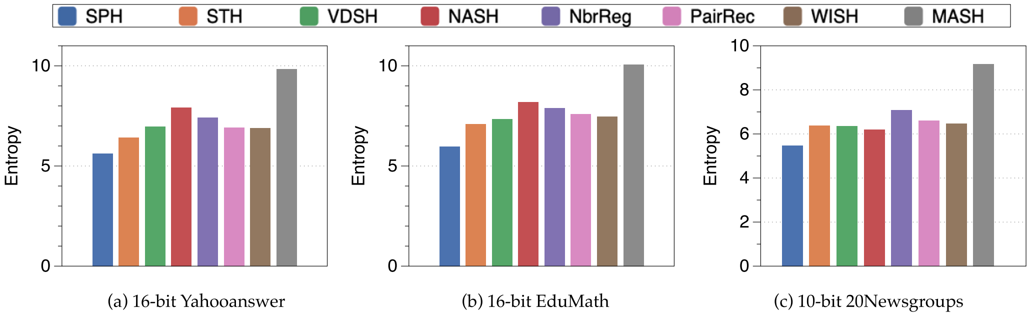
6.5.1 *Entropy Analysis.* As mentioned before, code balance means texts are uniformly distributed in each hash bucket. To evaluate the ability of different models in this aspect, we calculate the entropy of hash codes' distribution. The higher the entropy value is, the more uniform the texts in each hash bucket are. We set code length b as 16 for Yahooanswer and EduMath and 10 for 20Newsgroups. All texts in each dataset are fed into the trained model and get corresponding



(a) Average Case in Yahooanswer (b) Average Case in EduMath (c) Average Case in 20Newsgroups

(d) Worst Case in Yahooanswer (e) Worst Case in EduMath (f) Worst Case in 20Newsgroups

Fig. 8. Average and worst search case in three datasets using PLEB search.

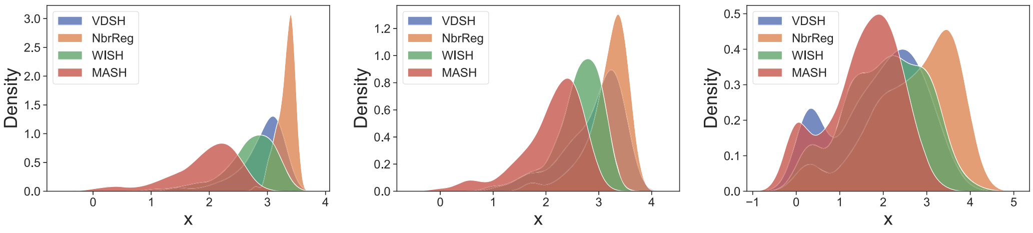


(a) 16-bit Yahooanswer (b) 16-bit EduMath (c) 10-bit 20Newsgroups

Fig. 9. The entropy of hash codes on three datasets.

hash codes. Then we calculate the entropy of all hash codes and plot the result in Figure 9, from which MASH apparently exceeds other baselines by 25%–60%. This result shows that MASH can generate more uniformly distributed hash codes in the Hamming space. In other words, MASH outperforms other baselines on code balance. We think this is because external memory can use a global perspective to guide the hash code learning and scatter texts across the entire Hamming space, rather than estimating the distribution of hash codes using mini-batch data.

6.5.2 Redundancy Analysis. According to [15], another code-balance effect is the deredundancy ability, which can better preserve the original locality structure of the data. To validate this fact, we select some representative models and set code length b as 16 for Yahooanswer and EduMath, and 10 for 20Newsgroups to conduct the following experiment. Specifically, we first calculate the entropy of classes in each hash bucket. If the entropy value is low, then the classes in the hash bucket are pure, which means that the model has good discrimination ability. Next, we use KDE [69], a non-parametric way to estimate the probability density function of a random variable, to compute the distribution of entropy in all hash buckets. Figure 10 report the result. In this Figure, if the distribution is proximate to the left, then the model has a solid ability to



(a) 16-bit Yahooanswer

(b) 16-bit EduMath

(c) 10-bit 20Newsgroups

Fig. 10. The distribution of some representative deep semantic hashing models on Yahooanswer dataset and 20Newsgroups dataset with kernel density estimation.

distinguish semantics in different classes. Compared with other baselines, the entropy distribution created by MASH is the closet on the far left. This phenomenon demonstrates that MASH has better de-redundancy ability than other baselines credited with better code balance by leveraging the external memory to optimize code balance globally, i.e., the bit-balanced objective in Equation (8) and the bit-uncorrelated objective in Equation (10).

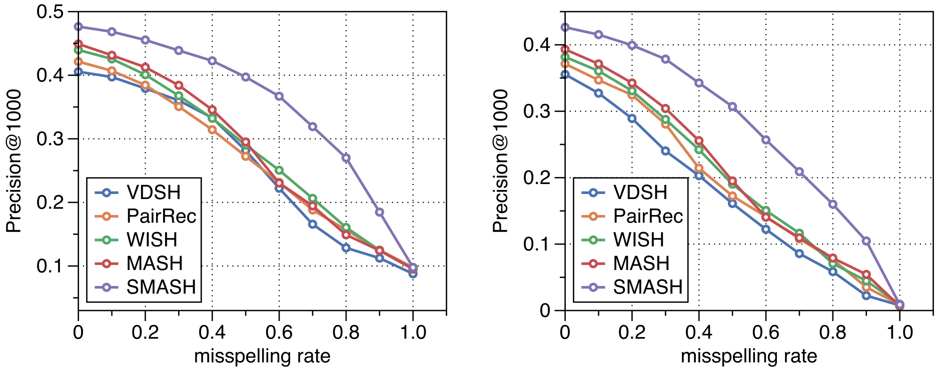
6.6 Performance on Noisy Data

In this experiment, we evaluate the performance of various models when the data contain noise. There are various text noises in real-world applications. Among them, misspelling is a common problem that frequently occurs in search engines [11]. Thus we focus on the noise introduced by spelling mistakes and typos but can also be applied to other noise types.

We simulate the presence of misspellings in the user query, where we use original texts to train different models and use noisy texts as the query. Referring to [66], we first create the noisy text dataset. Specifically, to create noisy text from original text x, we randomly choose one character from the text of x and replace it with nearby characters in a qwerty keyboard. For example, if character h is selected, it will be replaced by a character randomly chosen from y, u, j, n, b, g, surrounding the key h in a qwerty keyboard. In the Yahooanswer and EduMath datasets, we randomly modify a pre-defined fraction of the words ranging from 0% to 100%, which is called misspelling rate, for each text query according to the above procedure. Then we evaluate the performance, specifically $Precision@1000$, of various representative deep semantic hashing models.

Figure 11 shows the performance of the methods at various misspelling rates. We observe that SMASH has a slower decline in $Precision@1000$ with misspelling rate increases. The result demonstrates that SMASH is more robust than other methods except for the misspelling rate approaching 100%, which becomes almost a random guess for all methods. The results also demonstrate the robustness of employing the noise-aware encoder-decoder framework in SMASH toward noise texts, a realistic scenario when deploying the similarity search system in a real-world environment.

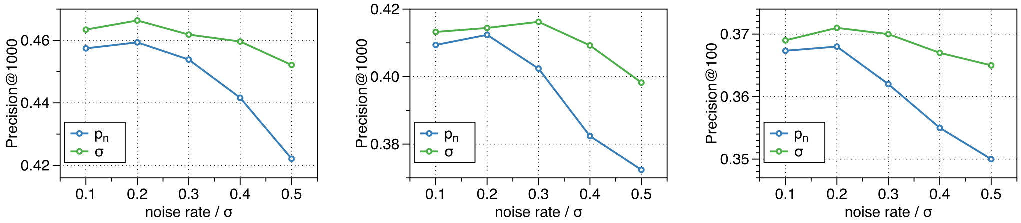
Besides, as discussed before, we draw a noise rate $\widetilde{p}_n$ from Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ $(\mu = 0)$ in each training step rather than assign a fixed p_n by manual setting. To evaluate the difference between them, we conduct the following experiment. Specifically, we use the fix noise rate p_n and adaptive $\widetilde{p}_n$ to train SMASH respectively. Figure 12 visualizes the performance with the increasing values of fix p_n and standard variation σ from 0.1, 0.2, 0.3, 0.4, 0.5 in datasets Yahooanswer, EduMath, 20Newsgroups respectively. In these figures, as p_n increases, the performances of SMASH firstly increase but decrease dramatically when p_n surpasses 0.1 or 0.2 in the corresponding datasets. Besides, we also observe that the performance of SMASH keeps stable with different σ and will achieve better performance better than the fixed p_n. This result demonstrates that



(a) 16-bit Yahooanswer

(b) 16-bit EduMath

Fig. 11. Precision@1000 of various deep models on Yahooanswer and EduMath dataset.



(a) 16-bit Yahooanswer

(b) 16-bit EduMath

(c) 10-bit 20Newsgroups

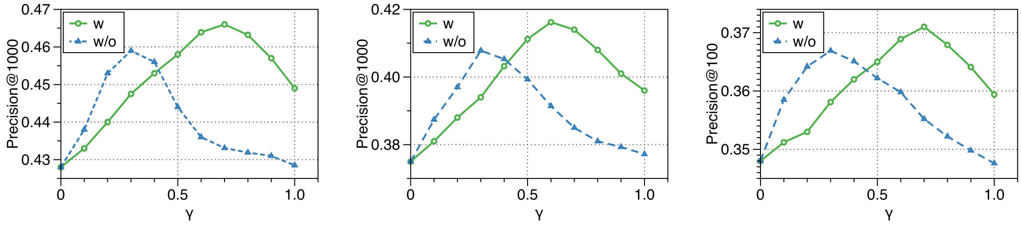
Fig. 12. The impact of fixed p_n and flexible σ on three datasets.

training the model adaptively by drawing the noise rate from the Gaussian distribution will achieve better results in our model.

6.7 External Memory Analysis

Here, we find that the external memory is helpful in our model, which provides a global perspective to guide the hash code training. To further explore the proposed select strategy in the external memory, we analyze the effects of γ and the loss change term $\frac{|L_{max}-L_{t-1}|}{L_{max}}$. To be specific, we set code length as 16 for Yahooanswer and EduMath and 10 for 20Newsgroups with $K = 1{,}000$ to retrieve candidate texts and use the *Precision* as the metric. We change γ from 0 to 1 with or without loss change term $\frac{|L_{max}-L_{t-1}|}{L_{max}}$ to train SMASH model. Note that when $\gamma = 0$, the external memory will not affect the training procedure; thus, we remove the loss term related to the external memory.

Figure 13 illustrates the effects of the above factors, where the green line uses the loss change term and the blue line does not. We can draw the following conclusions from these pictures. First, whether the loss change term is used or not, as γ increases, precision growths at the beginning and then decreases. These phenomena demonstrate that the strategy of choosing "fresh" hash codes is useful and is affected by γ, which controls the number of selected hash codes set $\hat{M}$. Second, in terms of the growth trend, the blue line grows faster than the green line. We think this is because the ratio of selected hash codes should be around a value. Influenced by the loss change term, the green line needs a larger γ to reach. Third, in terms of the best precision, using the loss change



(a) 16-bit Yahooanswer

(b) 16-bit EduMath

(c) 10-bit 20Newsgroups

Fig. 13. The impact of γ and loss change term on three datasets. The green line means using the loss change term and the blue line means not using the loss change term.

term is better than without the loss change term. This result demonstrates that the loss change term can make the model achieve better results by dynamically adjusting the ratio of the selected hash codes during the training process.

6.8 Hamming Space Manifold

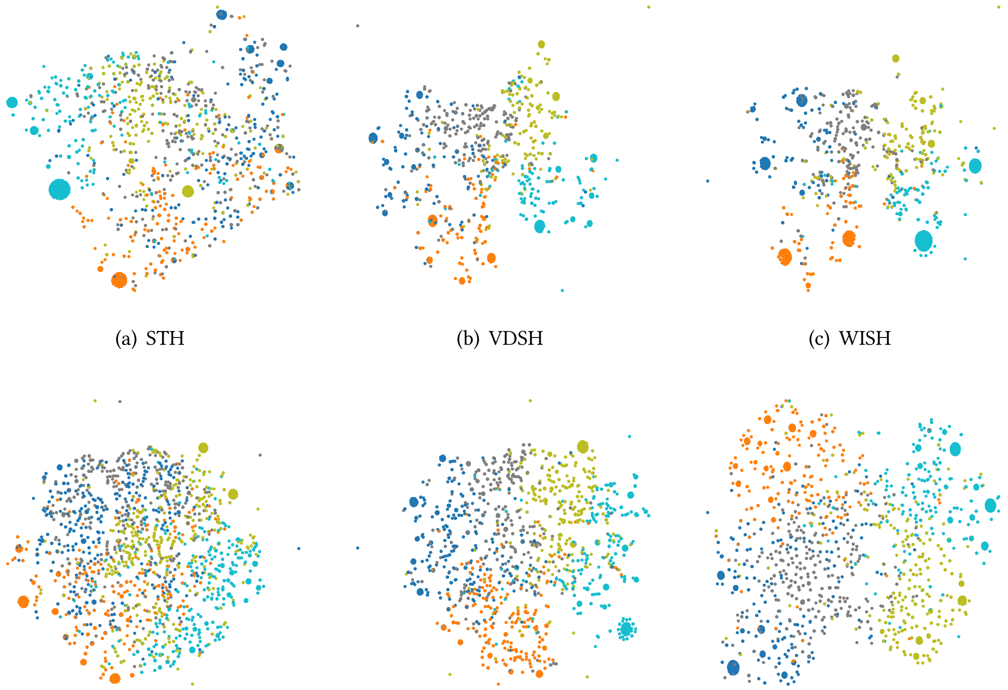
To intuitively see if the learned hash codes preserve the semantics of the original texts and if the Hamming space is fully utilized, we plot the embedding of the texts in the Hamming space using the 2-D t-SNE projection [71]. Specially, we compare our proposed model SMASH with other representative semantic models in the 20Newsgroups dataset and set the code length b to 12. In Figure 14, each point denotes a text and the texts belonging to the same class are represented using the same color. We randomly select five classes to display. Implications of pronounced colors are indicated in the figure's description.

Compared to other deep semantic models, the shallow-hashing method STH can only distinguish a small number of samples. Besides, as observed in VDSH and WISH, texts belonging to the same class are clustered together. However, the text embedding does not fully use the space and many texts overlap at the same point. In contrast, NASH and PairRec allow the texts to be dispersed in the space due to their local optimization of code balance. However, the effect of clustering is not significant. In the SMASH results, the texts belonging to the same class are clustered together and well scattered throughout the space credited to optimizing code balance globally. The visualization analysis verifies the effectiveness of our approach again and demonstrates that our approach preserves code balance, which ensures efficiency in the retrieval process.

7 DISCUSSION

In this section, we comprehensively discuss the advantages and some available research directions in the future. In this article, we explore the semantic hashing technology in the similar text search problem. Here, we maintain the semantic information in short hash codes using explicit propagation relevance regularization. Besides, the external memory in our model provides a global perspective to optimize the code balance target, which is sufficiently demonstrated in Section 6.5. Although the external memory will increase the training time and adds parameters, it is only considered in the training procedure, where we usually have more time tolerance in practice. When the model is used as the inference procedure, we only apply the encoder network, which does not include the external memory, as the hash function to map the original texts to hash codes. Thus the external memory will not have severe impacts on practical applications. Moreover, considering the text noise problem makes our model more suitable for real-world application scenarios.

Meanwhile, there are still some critical issues that can be explored in the future. On the one hand, in current text-based semantic hashing works [6, 8, 15, 23, 24, 63, 82], most of them are applied to



(a) STH (b) VDSH (c) WISH

(d) NASH (e) PairRec (f) SMASH

Fig. 14. Visualization of the 12 bits hash codes of 20Newsgroups generated by some representative models. The main classes in the picture include electronics(light blue), hardware(orange), electronics(brown), windows x(Orange), hardware(navy blue), christian(yellow).

similar text search problems and have achieved great success. These frameworks use one encoder as the hash function with homogeneous textual inputs. However, the similar text search task is one of the problems in the information retrieval domain. As for heterogeneous input tasks such as ad-hot and QA tasks, there is little work in the semantic hashing applied to them. These semantic hashing frameworks may not guarantee desired performance because the difference between the input texts is not considered. How to introduce the ad-hot and QA model to the semantic hashing field is still an open direction.

On the other hand, BOW and TF-IDF are two common text representations in text-based semantic hashing. Still, it may lose some vital information like the word order and dependency structure. Although [15] considers this problem, they only use simple RNN and CNN to replace the MLP networks. Nowadays, many pre-trained large models have achieved excellent performance in different fields [14, 84]. In our opinion, information retention is a crucial factor in semantic hashing, and these large pre-trained models may give help. However, large pre-trained models will increase the inference time and the storage overhead, which contradicts the original goal of semantic hashing. Along this line, some model compression methods, such as knowledge distillation [17] and network quantization [85, 93], could be potentially helpful.

8 CONCLUSION

In this article, we presented a focused study on semantic hashing learning. Specifically, we designed two lightweight and practical unsupervised semantic hashing models, the MASH and the

SMASH model incorporating important factors on hash codes. Specifically, the MASH model uses relevance propagation regularization to ensure the accuracy of short hash codes. With external memory, the MASH model can generate code-balance hash codes from a global perspective, promoting the effectiveness and efficiency of hash codes. Furthermore, against the noise texts, the improved SMASH model applies a novel noise-aware encoder-decoder framework to generate robust hash codes. Finally, extensive experiments on three real-world datasets demonstrated the effectiveness and efficiency of MASH and SMASH. We hope this work will lead to more studies in the future.

REFERENCES

[1] Aris Anagnostopoulos, Luca Becchetti, Ilaria Bordino, Stefano Leonardi, Ida Mele, and Piotr Sankowski. 2015. Stochastic query covering for fast approximate document retrieval. *ACM Transactions on Information Systems* 33, 3 (2015), 1–35.

[2] Tao Lei Hrishikesh Joshi Regina Barzilay, Tommi Jaakkola, Katerina Tymoshenko, and Alessandro Moschitti Lluıs Marquez. 2016. Semi-supervised question retrieval with gated convolutions. In *Proceedings of the NAACL-HLT*. 1279–1289.

[3] Guilherme Torresan Bazzo, Gustavo Acauan Lorentz, Danny Suarez Vargas, and Viviane P. Moreira. 2020. Assessing the impact of ocr errors in information retrieval. In *Proceedings of the European Conference on Information Retrieval*. Springer, 102–109.

[4] Saddam Bekhet and Amr Ahmed. 2018. An integrated signature-based framework for efficient visual similarity detection and measurement in video shots. *ACM Transactions on Information Systems* 36, 4 (2018), 1–38.

[5] David Berthelot, Colin Raffel, Aurko Roy, and Ian Goodfellow. 2018. Understanding and improving interpolation in autoencoders via an adversarial regularizer. In *Proceedings of the International Conference on Learning Representations*.

[6] Suthee Chaidaroon, Travis Ebesu, and Yi Fang. 2018. Deep semantic text hashing with weak supervision. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1109–1112.

[7] Suthee Chaidaroon, Dae Hoon Park, Yi Chang, and Yi Fang. 2020. node2hash: Graph aware deep semantic text hashing. *Information Processing and Management* 57, 6 (2020), 102143.

[8] Suthee Chaidaroon and Fang Yi. 2017. Variational deep semantic hashing for text documents. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 75–84.

[9] Miaomiao Cheng, Liping Jing, and Michael K. Ng. 2020. Robust unsupervised cross-modal hashing for multimedia retrieval. *ACM Transactions on Information Systems* 38, 3 (2020), 1–25.

[10] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. 1724–1734.

[11] Silviu Cucerzan and Eric Brill. 2004. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. 293–300.

[12] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th Annual Symposium on Computational Geometry*. 253–262.

[13] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science* 41, 6 (1990), 391–407.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics. 4171–4186.

[15] Khoa D. Doan and Chandan K. Reddy. 2020. Efficient implicit unsupervised text hashing using adversarial autoencoder. In *Proceedings of the Web Conference 2020*. 684–694.

[16] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2012. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 12 (2012), 2916–2929.

[17] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision* 129, 6 (2021), 1789–1819.

[18] Kristen Grauman and Rob Fergus. 2013. Learning binary hash codes for large-scale image search. In *Proceedings of the Machine Learning for Computer Vision*. Springer, 49–87.

[19] Yanhui Gu, Zhenglu Yang, Guandong Xu, Miyuki Nakano, Masashi Toyoda, and Masaru Kitsuregawa. 2014. Exploration on efficient similar sentences extraction. *World Wide Web* 17, 4 (2014), 595–626.

[20] Jiafeng Guo, Yinqiong Cai, Yixing Fan, Fei Sun, Ruqing Zhang, and Xueqi Cheng. 2022. Semantic models for the first-stage retrieval: A comprehensive review. *ACM Transactions on Information Systems* 40, 4 (2022), 66:1–66:42.

[21] Lei Guo, Hongzhi Yin, Tong Chen, Xiangliang Zhang, and Kai Zheng. 2021. Hierarchical hyperedge embedding-based representation learning for group recommendation. *ACM Transactions on Information Systems* 40, 1 (2021), 1–27.

[22] Raiza Hanada, Maria da Graça C. Pimentel, Marco Cristo, and Fernando Anglada Lores. 2016. Effective spelling correction for eye-based typing using domain-specific information about error distribution. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management.* 1723–1732.

[23] Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2019. Unsupervised neural generative semantic hashing. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval.* 735–744.

[24] Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2020. Unsupervised semantic hashing with pairwise reconstruction. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval.* 2009–2012.

[25] Junfeng He, Shih-Fu Chang, Regunathan Radhakrishnan, and Claus Bauer. 2011. Compact hashing with joint optimization of search accuracy and time. In *Proceedings of the CVPR 2011.* IEEE, 753–760.

[26] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon. 2012. Spherical hashing. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition.* IEEE, 2957–2964.

[27] Wei Hua, Yulei Sui, Yao Wan, Guangzhong Liu, and Guandong Xu. 2020. Fcca: Hybrid code representation for functional clone detection using attention networks. *IEEE Transactions on Reliability* 70, 1 (2020), 304–318.

[28] Jizhou Huang, Haifeng Wang, Yibo Sun, Miao Fan, Zhengjie Huang, Chunyuan Yuan, and Yawen Li. 2021. HGAMN: Heterogeneous graph attention matching network for multilingual POI retrieval at baidu maps. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.* 3032–3040.

[29] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* 2553–2561.

[30] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management.* 2333–2338.

[31] Zhenya Huang, Binbin Jin, Hongke Zhao, Qi Liu, Defu Lian, Tengfei Bao, and Enhong Chen. 2022. Personal or general? A hybrid strategy with multi-factors for news recommendation. *ACM Transactions on Information Systems* (2022). Just Accepted.

[32] Zhenya Huang, Xin Lin, Hao Wang, Qi Liu, Enhong Chen, Jianhui Ma, Yu Su, and Wei Tong. 2021. Disenqnet: Disentangled representation learning for educational questions. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.* 696–704.

[33] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing.* Jeffrey Scott Vitter (Ed.), ACM, 604–613.

[34] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2010), 117–128.

[35] Catherine Kobus, François Yvon, and Géraldine Damnati. 2008. Normalizing SMS: Are two metaphors better than one?. In *Proceedings of the 22nd International Conference on Computational Linguistics.* 441–448.

[36] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. 2012. Efficient retrieval of recommendations in a matrix factorization framework. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management.* 535–544.

[37] Brian Kulis and Trevor Darrell. 2009. Learning to hash with binary reconstructive embeddings. *Advances in Neural Information Processing Systems* 22 (2009), 1042–1050.

[38] Dan Li, Tong Xu, Peilun Zhou, Weidong He, Yanbin Hao, Yi Zheng, and Enhong Chen. 2021. Social context-aware person search in videos via multi-modal cues. *ACM Transactions on Information Systems* 40, 3 (2021), 1–25.

[39] Hang Li and Jun Xu. 2014. Semantic matching in search. *Foundations and Trends in Information Retrieval* 7, 5 (2014), 343–469.

[40] Jing Li, Dafei Yin, Haozhao Wang, and Yonggang Wang. 2021. DCSpell: A detector-corrector framework for chinese spelling error correction. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval.* 1870–1874.

[41] Sen Li, Fuyu Lv, Taiwei Jin, Guli Lin, Keping Yang, Xiaoyi Zeng, Xiao-Ming Wu, and Qianli Ma. 2021. Embedding-based product retrieval in taobao search. In *Proceedings of the KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.* ACM, 3181–3189.

[42] Yawen Li, Di Jiang, Rongzhong Lian, Xueyang Wu, Conghui Tan, Yi Xu, and Zhiyang Su. 2021. Heterogeneous latent topic discovery for semantic text mining. *IEEE Transactions on Knowledge and Data Engineering* 35, 1 (2021), 533–544.

[43] Xin Lin, Zhenya Huang, Hongke Zhao, Enhong Chen, Qi Liu, Hao Wang, and Shijin Wang. 2021. Hms: A hierarchical solver with dependency-enhanced understanding for math word problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 4232–4240.

[44] Qi Liu, Zai Huang, Zhenya Huang, Chuanren Liu, Enhong Chen, Yu Su, and Guoping Hu. 2018. Finding similar exercises in online education systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1821–1830.

[45] Qi Liu, Zhenya Huang, Yu Yin, Enhong Chen, Hui Xiong, Yu Su, and Guoping Hu. 2019. Ekt: Exercise-aware knowledge tracing for student performance prediction. *IEEE Transactions on Knowledge and Data Engineering* 33, 1 (2019), 100–115.

[46] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. 2012. Supervised hashing with kernels. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2074–2081.

[47] Xiao Luo, Haixin Wang, Daqing Wu, Chong Chen, Minghua Deng, Jianqiang Huang, and Xian-Sheng Hua. 2020. A survey on deep hashing methods. *ACM Transactions on Knowledge Discovery from Data* (2020). Just Accepted.

[48] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International Conference on World Wide Web*. 1291–1299.

[49] Sara Morsy and George Karypis. 2016. Accounting for language changes over time in document similarity search. *ACM Transactions on Information Systems* 35, 1 (2016), 1–26.

[50] Andrew Ng, Michael Jordan, and Yair Weiss. 2001. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems* 14 (2001), 849–856.

[51] Priyanka Nigam, Yiwei Song, Vijai Mohan, Vihan Lakshman, Weitian Allen Ding, Ankit Shingavi, Choon Hui Teo, Hao Gu, and Bing Yin. 2019. Semantic product search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2876–2885.

[52] Mohammad Norouzi, Ali Punjani, and David J. Fleet. 2012. Fast search in hamming space with multi-index hashing. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 3108–3115.

[53] Nouha Othman, Rim Faiz, and Kamel Smaïli. 2019. Manhattan siamese LSTM for question retrieval in community question answering. In *Proceedings of the OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*. Springer, 661–677.

[54] Hongbin Pei, Bingzhe Wei, Kevin Chang, Chunxu Zhang, and Bo Yang. 2020. Curvature regularization to prevent distortion in graph embedding. In *Proceedings of the Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020*.

[55] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric graph convolutional networks. In *Proceedings of the 8th International Conference on Learning Representations*. Retrieved from OpenReview.net. https://openreview.net/forum?id=S1e2agrFvS.

[56] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J. Guibas, and Jascha Sohl-Dickstein. 2015. Deep knowledge tracing. In *Proceedings of the NIPS*.

[57] Zongyue Qin, Yunsheng Bai, and Yizhou Sun. 2020. Ghashing: Semantic graph hashing for approximate similarity search in graph databases. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2062–2072.

[58] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. 2014. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the International Conference on Machine Learning*. PMLR, 1278–1286.

[59] Jirí Rihák and Radek Pelánek. 2017. Measuring similarity of educational items using data on learners' performance. In *Proceedings of the 10th International Conference on Educational Data Mining*. 16–23.

[60] Stephen E. Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the SIGIR'94*. Springer, 232–241.

[61] Ruslan Salakhutdinov and Geoffrey Hinton. 2009. Semantic hashing. *International Journal of Approximate Reasoning* 50, 7 (2009), 969–978.

[62] Ying Shan, Jian Jiao, Jie Zhu, and J. C. Mao. 2018. Recurrent binary embedding for gpu-enabled exhaustive retrieval from billion-scale semantic vectors. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2170–2179.

[63] Dinghan Shen, Qinliang Su, Paidamoyo Chapfuwa, Wenlin Wang, Guoyin Wang, Ricardo Henao, and Lawrence Carin. 2018. NASH: Toward end-to-end neural architecture for generative semantic hashing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2041–2050.

[64] Yansong Shen, Lin Li, Qing Xie, Xin Li, and Guandong Xu. 2022. A two-tower spatial-temporal graph neural network for traffic speed prediction. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 406–418.

[65] Robert F. Sproull. 1991. Refinements to nearest-neighbor searching ink-dimensional trees. *Algorithmica* 6, 1 (1991), 579–589.

[66] Ankit Srivastava, Piyush Makhija, and Anuj Gupta. 2020. Noisy text data: Achilles' heel of BERT. In *Proceedings of the 6th Workshop on Noisy User-generated Text (W-NUT 2020)*. 16–21.

[67] Yukihiro Tagami. 2017. Annexml: Approximate nearest neighbor search for extreme multi-label classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 455–464.

[68] Lynda Tamine, Laure Soulier, Gia-Hung Nguyen, and Nathalie Souf. 2019. Offline versus online representation learning of documents using external knowledge. *ACM Transactions on Information Systems* 37, 4 (2019), 1–34.

[69] George R. Terrell and David W. Scott. 1992. Variable kernel density estimation. *The Annals of Statistics* 20, 3 (1992), 1236–1265.

[70] Wei Tong, Shiwei Tong, Wei Huang, Liyang He, Jianhui Ma, Qi Liu, and Enhong Chen. 2020. Exploiting knowledge hierarchy for finding similar exercises in online education systems. In *Proceedings of the 20th IEEE International Conference on Data Mining*. IEEE, 1298–1303.

[71] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, 11 (2008), 2579–2605.

[72] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. 2015. Learning to hash for indexing big data-A survey. *Proc. IEEE* 104, 1 (2015), 34–57.

[73] Jun Wang, Wei Liu, Andy X Sun, and Yu-Gang Jiang. 2013. Learning hash codes with listwise supervision. In *Proceedings of the IEEE International Conference on Computer Vision*. 3032–3039.

[74] Jingdong Wang, Ting Zhang, Nicu Sebe, and Heng Tao Shen. 2017. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 4 (2017), 769–790.

[75] Qifan Wang, Dan Zhang, and Luo Si. 2013. Semantic hashing using tags and topic modeling. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 213–222.

[76] Xin Wang, Wei Huang, Qi Liu, Yu Yin, Zhenya Huang, Le Wu, Jianhui Ma, and Xue Wang. 2020. Fine-grained similarity measurement between educational videos and exercises. In *Proceedings of the 28th ACM International Conference on Multimedia*. 331–339.

[77] Zizhen Wang, Yixing Fan, Jiafeng Guo, Liu Yang, Ruqing Zhang, Yanyan Lan, Xueqi Cheng, Hui Jiang, and Xiaozhao Wang. 2020. Match^2: A matching over matching model for similar question identification. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 559–568.

[78] Yair Weiss, Antonio Torralba, and Rob Fergus. 2008. Spectral hashing. *Advances in Neural Information Processing Systems* 21 (2008), 1753–1760.

[79] Philip C. Woodland, Sue E. Johnson, Pierre Jourlin, and K. Spärck Jones. 2000. Effects of out of vocabulary words in spoken document retrieval. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 372–374.

[80] Hao Yan, Shuai Ding, and Torsten Suel. 2009. Inverted index compression and query processing with optimized document ordering. In *Proceedings of the 18th International Conference on World Wide Web*. 401–410.

[81] R. Baeza Yates and B. Ribeiro Neto. 2011. *Modern Information Retrieval: The Concepts and Technology Behind Search*. Pearson Education Ltd., Harlow.

[82] Fanghua Ye, Jarana Manotumruksa, and Emine Yilmaz. 2020. Unsupervised few-bits semantic hashing with implicit topics modeling. In *Proceedings of the EMNLP (Findings)*, Vol. 20. Association for Computational Linguistics (ACL), 2566–2575.

[83] Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. 2018. Understanding straight-through estimator in training activation quantized neural nets. In *Proceedings of the International Conference on Learning Representations*.

[84] Yu Yin, Qi Liu, Zhenya Huang, Enhong Chen, Wei Tong, Shijin Wang, and Yu Su. 2019. Quesnet: A unified representation for heterogeneous test questions. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1328–1336.

[85] Chunyu Yuan and Sos S. Agaian. 2021. A comprehensive review of binary neural network. arXiv:2110.06804. Retrieved from https://arxiv.org/abs/2110.06804.

[86] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. 2010. Self-taught hashing for fast similarity search. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 18–25.

[87] Jing Zhang, Jie Tang, Cong Ma, Hanghang Tong, Yu Jing, Juanzi Li, Walter Luyten, and Marie-Francine Moens. 2017. Fast and flexible top-k similarity search on large networks. *ACM Transactions on Information Systems* 36, 2 (2017), 1–30.

[88] Yizhe Zhang, Dinghan Shen, Guoyin Wang, Zhe Gan, Ricardo Henao, and Lawrence Carin. 2017. Deconvolutional paragraph representation learning. In *Proceedings of the Advances in Neural Information Processing Systems*. Neural information processing systems foundation, 4170–4180.

[89] Yan Zhang, Ivor Tsang, Hongzhi Yin, Guowu Yang, Defu Lian, and Jingjing Li. 2020. Deep pairwise hashing for cold-start recommendation. *IEEE Transactions on Knowledge and Data Engineering* 34, 7 (2020), 3169–3181.

[90] Yifei Zhang and Hao Zhu. 2019. Doc2hash: Learning discrete latent variables for documents retrieval. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2235–2240.

[91] Le Zhao and Jamie Callan. 2010. Term necessity prediction. In *Proceedings of the 19th ACM Conference on Information and Knowledge Management.* ACM, 259–268.

[92] Meng Zhao, Hao Wang, Liangliang Cao, Chen Zhang, Hongzhi Yin, and Fanjiang Xu. 2015. Lsif: A system for large-scale information flow detection based on topic-related semantic similarity measurement. In *Proceedings of the 2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. IEEE, 417–424.

[93] Wenyu Zhao, Teli Ma, Xuan Gong, Baochang Zhang, and David Doermann. 2020. A review of recent advances of binary neural networks for edge computing. *IEEE Journal on Miniaturization for Air and Space Systems* 2, 1 (2020), 25–35.

[94] Lin Zheng, Qinliang Su, Dinghan Shen, and Changyou Chen. 2020. Generative semantic hashing enhanced via boltzmann machines. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics.* 777–788.

Received 31 May 2022; revised 14 September 2022; accepted 20 October 2022