Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 09, No. 02, June 2023.

SEMANTIC RETRIEVAL FOR INDONESIAN QURAN AUTOCOMPLETION

Rian Adam Rajagede, Kholid Haryono and Rizan Qardafil

(Received: 7-Dec.-2022, Revised: 1-Feb.-2023 and 22-Feb.-2023, Accepted: 5-Mar.-2023)

ABSTRACT

Attending lectures is a common way to learn Islamic knowledge. The speaker talks in front of the forum and participants take notes on the lecture material. Many participants listen to the lecture while taking notes either in books or on other digital devices to avoid forgetting the discussed topics. However, note-taking during the lecture can be challenging, with no complementing module from the speaker. Lecturers have different paces and varying ways of delivering. In addition, sometimes, participants cannot always focus during the lecture. Those factors can cause problems in the note-taking process: some details can be lost or even shift the meaning. For note-taking on sensitive topics, such as verses from the Quran, the note-taking process must be done carefully and avoid mistakes. In this study, we proposed an autocomplete system for the Indonesian translation of the Quran that will help the user in note-taking in Islamic lectures. The user writes down words, the parts of the Quran verse that he/she hears and the system will retrieve the most similar verses. With semantic retrieval, the user does not need to write down the exact words of the verses he/she heard. The system can also handle typographical-errors that usually occur in note-taking. We use FastText and calculate the cosine distance between the query and verses for the retrieval process. We also performed several optimization steps to create a robust system for the production stage. The system is evaluated by comparing how close the returned verse is with the ground truth. The proposed method's result in terms of accuracy reached 70.59% for the top 5 retrieved verses and 76.47% for the top 10 retrieved verses.

KEYWORDS

Semantic retrieval, Quran auto-completion.

1. INTRODUCTION

Islamic knowledge is commonly communicated through lectures, in which the speaker engages and informs a large group of listeners. This type of Islamic lecture is fairly common in Muslim-majority countries, such as Indonesia. Islamic lectures are held in mosques across the country. Because these lectures are given in a variety of settings, they are frequently delivered without supporting materials, such as presentation slides or audience modules. Most of the time, lecturers give content from their notes, Islamic textbooks or from the holy book Quran. As a result, some people listen to lectures while taking notes on books or other digital devices to help them remember the information. However, without the use of a module, the process of taking notes on Islamic lectures could be more difficult. Some participants might be unable to maintain their concentration throughout the lecture or some lecturers might deliver the content at a fast pace, which could affect the difficulty of taking notes. Particularly, for participants who took notes using mobile devices, in general, people were slower to take notes through mobile applications than through writing by hand on paper [1].

This situation can affect participants to overlook some important aspects or make some errors when taking notes. It is quite risky when it comes to noting the snippets of Quran verses; errors in note-taking can lead to different meanings or different verses. Another strategy is only to write the surah's number and the verse number. However, it will make the notes become difficult to read. One solution that can be used is to create a system that is able to suggest the suitable choice of Quran verses based on user notes.

We can formulate this problem as an information-retrieval problem in the domain of Quran verses. In this research, we tackle this problem by creating an auto-completion system in a mobile note-taking application. In the auto-completion system, when the user types in the application, the system typically suggests a word or sentence before the user completes it. Instead of getting a word or sentence recommendation, we can suggest Quran verses during user typing. The user's incomplete sentence can be seen as a query and the system will retrieve recommended verses as a suggestion to complete the

R. A. Rajagede, K. Haryono and R. Qardafil are with Department of Informatics, Universitas Islam Indonesia, Sleman, Indonesia. Emails: rian.adam@uii.ac.id, kholid.haryono@uii.ac.id and rizan.qardafil@students.uii.ac.id

note.

The creation of an information retrieval system for the Quran or Hadith has previously been researched with several different approaches [2]–[4]. Most of the methods used in that research rely on stemming algorithms, such as the Nazief-Adriani algorithm [4]-[5] or ECS (Enhance Confix Stripping) [6]. Other studies uses corpus and thesaurus to ensure input from the user is in the database [2]-[3]. However, those approaches will be difficult to use with note-taking in Islamic lectures. This is because when note-taking, users can make writing errors caused by several factors that have been mentioned previously. These writing errors can bring up words that do not exist in the database. Those words are commonly known as Out-of-Vocabulary (OOV) words. Two approaches can be taken to overcome OOV errors: using edit distance [7] or semantic embedding [8]-[9].

In the edit-distance approach, the system measures word similarity if the word is not found in the database, while the semantic-embedding approach uses an additional algorithm to convert words into vectors, then compares the similarity of documents in vector space. Several methods that can be used to convert words into vectors are Word2Vec [10], FastText [11] and BERT [12]. In previous studies, researchers have compared Word2Vec and FastText for the Quran and Hadith [13]. In that research, Word2Vec outperformed FastText both intrinsically and extrinsically. However, among the three embedding methods previously mentioned, Word2Vec cannot handle OOV words, which is essential in auto-completion.

In this study, we propose a Quran auto-completion system that will be embedded in a mobile application for note-taking in Islamic lectures. This auto-completion system focuses on retrieving the Indonesian translation of Quran verses based on words that are fragments of the Quran's verse. The system will use FastText to get the semantic embedding of the user query, then look for the most suitable verse.

This paper will be organized as follows. We will briefly introduce FastText, Sentence Embedding and Compress FastText in Section 2. Section 3 outlines the methodology used in the study, including the data preparation, similarity search algorithm and evaluation methods. In Sections 4 and 5, we present the results of the research and discuss the findings. Section 4 will focus more on sentence-embedding model evaluation, while Section 5 will focus more on the retrieval-system evaluation. Finally, in Section 6, we provide concluding remarks and recommendations for further research.

2. RELATED WORKS

2.1 FastText

FastText is an open-source, free, lightweight library by Meta that allows users to represent words in a continuous vector space, called word embedding [11]. Word embedding uses machine learning, particularly neural networks, to convert the representation of a word into a vector. Words with similar meanings are close together in their vector space. For instance, the word "fun" will be closer to "joy" than "fan" even though it has similar letters. The use of word embedding as a feature has improved the performance of several models in many NLP tasks, such as sentiment analysis [14], sentence similarity [15], emotion classification [16] and essay scoring [17].

The key idea behind FastText is to represent words as a bag of character n-grams (sub-strings of length n) instead of using a single vector for each word. This approach helps in handling Out-Of-Vocabulary (OOV) words effectively, as these words can still be represented by the character n-grams even if they were not seen during training.

2.2 Sentence Embedding

Sentence embedding converts a sentence consisting of several words into a vector. This method expands word-embedding methods, such as Word2Vec or FastText, changing the representation from the word level to the sentence level.

Similar to word embedding, sentence embedding will make sentences with semantic similarities have closeness in their vector space. There are various techniques to obtain sentence-embedding vectors. One way is to calculate the average word vector in a sentence. This method is used in FastText. The disadvantage of this method is that it ignores the word order, which may give some contextual meaning. Another method is to map the entire sentence directly to get its embedding vector. This method is used

in BERT Sentence Embedding (SBERT) [18]. SBERT is an extension of the Bidirectional Encoder Representations from Transformers (BERT) [12] architecture that uses the Siamese network model. SBERT constructs vectors by paying attention to the word's context in the sentence. Previous studies have shown that this method is superior to FastText [19].

2.3 Compress FastText

One of the challenges in developing models for embedding is the model's size. The size of this model will affect the system at the production stage, because it must provide more extensive storage or memory capacity. Although large storage is easy to achieve in the production stage, large-memory requirements are still quite expensive. In previous studies, the Compress FastText model [20] has been developed, which has a much smaller size than the original size with a less significant decrease in performance. In this study, we focus on using a compress FastText model, which is more efficient in memory usage.

Compress FastText was developed by utilizing several optimization methods applied to the vocab and n-gram matrix of the original FastText model. In this study, we examine several optimization parameters and see the effect on the size and performance of the model. The first optimization method we tested was using half-precision, which changed the original FastText data type from initially used 32-bit floats to 16-bit floats. In addition, we reduce the size of the matrix by eliminating less-frequent words and n-grams. We will also compare it with the smallest model created by the Compress FastText author. The model is obtained with additional optimization methods, such as matrix factorization and product quantization.

3. METHODS

This research consisted of two major parts. The first part focuses on converting the query and the verses into embedding vectors, q and v. Then, at the next stage, a vector-similarity search is carried out between the query and all verses in the Quran. In general, the process of the system in this study is shown in Figure 1.

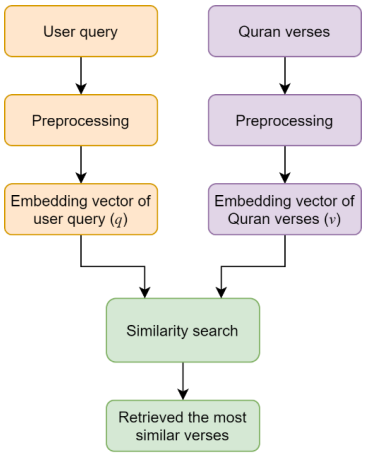

Figure 1. Steps to retrieve Quran verses in the aut-ocompletion system.

3.1 Pre-processing

In this stage, we pre-process the data before calculating the embedding vector, which includes case folding, stemming and stop-word removal. The data used in this study is a public Quran dataset taken from various sources and has been combined in a single JSON file [21]. The dataset contains data for the Arabic text of the Quran, transliteration, English and Indonesian translation of the Quran. This study only uses the Indonesian translation part for the retrieval process.

Each word in the database is processed through the following pre-processing stages:

1. Case folding; we change all letters into lowercase
2. Word stemming using the Nazief-Adriani algorithm [5] on PySastrawi [22]
3. Stop-word removal. Because currently, there is no publicly available data for this domain, we

choose the stop word based on word occurrences in the Quran. We try different threshold values and evaluate the model to the evaluation set. We explain this step later in Section 5.

3.2 Word and Sentence Embedding

For this research, we need to compute both word and sentence-embedding vectors of the user query and the Quran verses. We use the Compress FastText Model to calculate the word embedding for each query. To get the sentence embedding, first, we choose some words in the query and the Quran verses and calculate their word-embedding vectors. Then, we get the sentence-embedding vector by calculating the average of the word vectors. The pre-processing steps are performed before calculating the sentence vector query, as illustrated in Figure 2. Later in sub-section 3.3 will be explained how we select words for each query and verse to calculate sentence embedding.

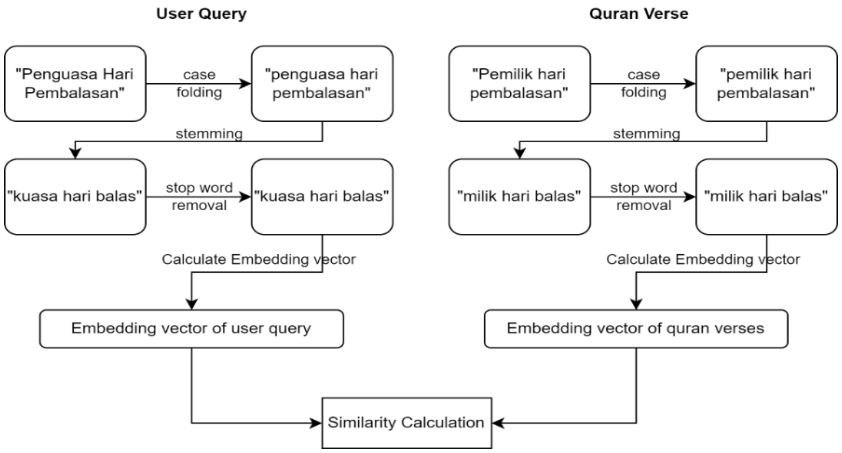

Figure 2. Similarity process using a single Indonesian verse (Quran 1:4) as an example.

To speed up the computing process, we store a dictionary file that contains all word embedding vectors of processed unique words in the Quran. This dictionary will help map a word with its vector representation faster. This additional data adds around 5 MB in size.

3.3 Similarity Search Algorithm

In this step, we measure the distance of the sentence vector of the user query (q) to the sentence vector of each verse (v) in the Quran. The smaller the distance between q and v^t, the more similar the meaning of the query to the t^{th} verse. The similarity is calculated using the cosine distance between vectors using:

$$\text{dist}(q, v^t) = \cos(\theta) = \frac{q \cdot v^t}{\|q\|\|v^t\|}$$

One characteristic of Quran retrieval is that the size of the search document is not too large and will not change over time. Therefore, we propose to combine sentence vector and word vector by aligning the words in the query and the words in the verse before averaging to get better performance.

We choose the corresponding word in the verse for each word in the query by calculating the distance between q_k and v_i^t, the word vectors of the k^{th} word in the query and the i^{th} word in the t^{th} verse. Then, the chosen words from the verse are averaged to get the sentence vector to represent the verse. This step is to avoid some words that may frequently appear in the verse, but not in the user query, affecting the average word vector of the verse. This method gives better results than directly calculating the distance between a sentence vector query and a verse. The explained retrieval algorithm can be seen in Table 1.

The drawback of this retrieval method is that the time complexity increases over the length of the query and the length of the verse. The additional step that iterates all words in the query for each verse increases the time complexity from $O(NM)$, where N is the number of verses and M is the length of the sentence vector into $O(NKLM)$, where K is the number of the word in the query and L is the number of the word in the verse. To address this problem, we speed up the distance calculation by leveraging parallelization calculation between cores using the Python multi-processing package. The processes at lines 3-8 from Table 1 will be computed in parallel.

Table 1. Similarity search algorithm.

Algorithm 1: Sentence + word vector retrieval

Input: query, verses

Output: List of distance *d* between query and verses

1: $q \leftarrow \frac{1}{|\text{query}|}\sum_k \text{word\_emb}(\text{query}_k)$ ▷ sentence embedding of the query

2: **for** t ← 1, |verses| **do** ▷ iterate for each verse in the Quran

3: **for** k ← 1, |query| **do** ▷ iterate for each word in the query

4: $q_k \leftarrow \text{word\_emb}(\text{query}_k)$ ▷ get the word embedding for the k[th] word in the query (q_k)

5: $l_k \leftarrow \underset{i=1,\ldots,|\text{verses}[t]|}{\text{argmin}} \text{dist}(q_k, v_i^t)$ ▷ calculate the distance between q_k and the word embedding for the i^{th} word in the t^{th} verses (v_i^t)

$$v_i^t \leftarrow \text{word\_emb}(\text{verses}_i^t)$$

▷ store the index of the closest word in the verse *i* to l_k

6: **end for**

7: $v^t \leftarrow \frac{1}{|\text{query}|}\sum_k \text{word\_emb}(v_{l_k}^t)$ ▷ sentence embedding of verses is calculated using only the closest word stored

8: $d^t \leftarrow \text{dist}(q, v^t)$ ▷ distance between sentence embedding of query and sentence embedding of verses

9: **end for**

3.4 Mobile Note-taking Application

As mentioned earlier, we can formulate this problem as an information-retrieval problem in the domain of Quran verses. However, in the user point-of-view, we format the retrieval process as an auto-completion system that suggests Quran verses during user typing. The user's incomplete sentence can be seen as a query and the system will retrieve recommended verses as a suggestion to complete the note.

We created a prototype mobile note-taking application based on Android to evaluate the system. Our models are stored on the virtual private server with 8 virtual CPU cores and communicated to the front-end *via* a Python API. The retrieval process is illustrated in Figure 3.

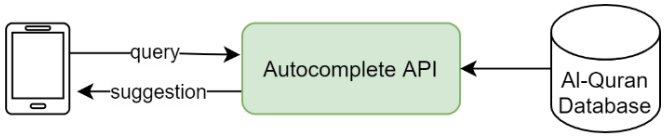

Figure 3. Illustration of auto-completion using API.

During writing in the note-taking application, the user starts writing a snippet of a verse using the "~" (tilde) symbol to trigger the auto-completion system. When the auto-completion system is triggered, the application will record what user type and use it as a retrieval query. After stopping typing for 1.5 seconds, the system will assume that the user completes the query and then the query will be sent to the API *via* the Internet. By using the algorithm presented in Table 1, the API will return ten recommended verses with high similarity with the query. We retrieved ten recommended verses to give users more options, as some verses may have high similarities.

3.5 Evaluation

In this study, we evaluate two parts. The first evaluation is the sentence-embedding model and the second part is the retrieval system. We dedicated Section 4 to explain in detail the evaluation steps of the embedding model.

For the retrieval process, we evaluate the retrieval system in two ways. The first method is done automatically by creating test data from the English translation of the Quran that is translated into Indonesian. We use English translation to create synthetic data that can be seen as a way to paraphrase the Indonesian translation of the Quran. This method is a common technique to enrich data for low-resource language tasks called back-translation [27]-[28]. The English translation was translated into Indonesian using Google Translate API.

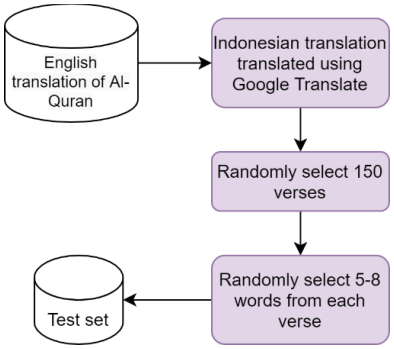

Figure 4. Steps to build test data to evaluate the retrieval system.

The test will be conducted by selecting 150 translated verses randomly, removing some words of each verse and then asking the system to retrieve the requested verses. The steps are illustrated in Figure 4.

The second evaluation method we do is to create test data containing Out-of-Vocabulary (OOV) words. This test data was made from the respondent's lecture notes, which contained verses from the Quran. Apart from lecture notes, this test data is also made by asking respondents who have memorized the Quran to recall the translation of a verse in the Quran and write it down.

We will compare the three retrieval methods at this evaluation stage. "Sentence + word vector" is the method we propose in Table 2. This method combines sentence and word-vector embedding. The second model we compared is "Sentence Vector" retrieval. This method will directly calculate the distance between the verse and query sentence vector and then return the verses that have the closest distance. The "Word matching" method is done by matching how many words in the query are found in the verse. The more query words are found in a verse, the higher the score returned for that verse. The details of the three models are shown in Table 2.

Table 2. Evaluated retrieval methods.

Model	Method
Sentence + vector retrieval	As explained in Table 1
Sentence retrieval	1. Calculate sentence vector for query and verse.
	2. Find the closest distance between query and verse.
Word matching	1. Count how many query words exist in each verse.
	2. Find the verse with the most number of query words found.

4. EMBEDDING MODEL EVALUATION

Before we implement the retrieval system, we need to evaluate the embedding model we will use in the system. In this section, we evaluated the FastText models after they had been compressed. It is worth noting that neither the FastText model nor Compress FastText used in this research is explicitly trained on Quranic verses. Therefore, we evaluated the model using general Indonesian text and expect that the model should give the best of its performance on the common words in the Indonesian language. We evaluated the sentence-embedding model both using intrinsic evaluation and extrinsic evaluation. In addition, we also evaluated the memory required by the system to run the model. The latter is quite essential for the production stage, as providing a system with a large RAM capacity is very expensive.

Intrinsic evaluation will measure how well the vector representation of the words generated by the model is. In this evaluation, we tested the aspect qualitatively by randomly taking Indonesian and Quran words and then looking at each model's five closest words in the vector space. For example, we try to find the

five closest words in the embedding space using the single word “rendang,” which is the traditional food name in Indonesia. We evaluate those words qualitatively with the word “rendang.”

For extrinsic evaluation, we also test the embedding-vector representation using an NLP task. The task that we chose was essay scoring using the Ukara dataset [24]. Ukara dataset is a dataset published by the Indonesian government for automatic essay scoring challenge for Indonesian language in 2019. The dataset contains student short answer and binary label for the correctness of the student answer. We evaluate the classifier performance when using embedding vector generated by the sentence-embedding model. We chose this task as the best model in that challenge is to use the FastText embedding model for Indonesian language.

Three models that compress FastText produced are compared here. Table 4 presents the compared FastText models as well as the optimization parameters that were utilized. Compress Model 1 is a model generated using default settings of the Compress Fasttext model [20]. We developed Compress Model 2 and Compress Model 3 by modifying the original FastText model for Indonesian language [23] using the parameters set up in Table 3 to compress the model size. We chose different vocab sizes and n-gram sizes to evaluate their effects on the model performance. The difference between the three models, apart from the vocab size and n-gram size, is that Compress Model 1 uses a matrix-quantization technique to reduce the model's size as suggested in [20]. Compared to the original model, compressed models use 16-bit floating points instead of 32-bit floating points.

Table 3. FastText and Compress FastText models comparison.

Model	Floating points	Matrix quant.	Vocab size	n-gram size
FastText Original Model [23]	32-bit	No	2,000,000	2,000,000
Compress Model 1	16-bit	Yes	20,000	100,000
Compress Model 2	16-bit	No	100,000	100,000
Compress Model 3	16-bit	No	100,000	300,000

The first thing that we look at is how much space the model takes up in storage, as well as the amount of memory that is required to transform words into vectors. The filprofiler package[1] is used to calculate the amount of memory capacity that is being used. Table 4 presents the findings of the study. We also compared the FastText compressed models with the SBERT model that was used in previous research [19].

Table 4. Storage and memory usage by FastText models.

Method	Storage (MB)	Memory (MB)
SBERT [19]	1075	2389
FastText Original Model [23]	6902	14484
Compress Model 1	14	77
Compress Model 2	117	254
Compress Model 3	233	384

4.1 Intrinsic Evaluation for Compressed Model

We also compared the quality of the four FastText models using intrinsic evaluation. Models retrieve the closest or similar words in their vector representation to the query words. We find the closest words using the most_similar() function from the Gensim package for FastText. This function will return the closest words whose distance is calculated using the cosine distance.

The model is tested by providing several Indonesian words and Quran words from different domains and assessing the returned words (human judgment). We use the same evaluation scheme from previous research, including how to calculate the model accuracy [13] by testing whether the most similar word returned is a synonym, antonym, related word or derivative form of the query. In this experiment, we found that the quality of the words returned by Compress Model 1 is sometimes very irrelevant to the

[1] https://pythonspeed.com/fil/

given query.

In Table 5, we show the query results on the word “Rendang”, one of Indonesia's popular foods. We chose the word “rendang” as illustration in Table 5, as that word is more common than the Quranic word. However, we evaluate the model using the Quranic word, as shown in Table 6. From the table, it can be seen that Compress Model 1 returns numerous words that are not categorized as synonyms, antonyms, related words or derived words to the given query (red text).

Table 5. Top-5 similar words for each model.

Method	Word query	Top-5 similar words	Category
FastText Original	Rendang (*Food name*)	- rendang (*lowercase of Rendang*)	Synonym
		- Kalio (*half-cooked Rendang*)	Related
		- Rendan (*mistype of Rendang*)	Synonym
		- Dendeng (*food from the same origin city of Rendang*)	Related
		- rendangnya (his/her/their rendang)	Derived
Compress Model 1	Rendang (*Food name*)	- Padang (*Padang City; Rendang origin city*)	Related
		- Ini (*this*)	Uncategorized
		- Dari (*from*)	Uncategorized
		- Tersebut (*the*)	Uncategorized
		- Dan (*and*)	Uncategorized
Compress Model 2	Rendang (*Food name*)	- rendang (*lowercase of Rendang*)	Synonym
		- Dendeng (*food from the same origin city of Rendang*)	Related
		- Balado (*food from the same origin city of Rendang*)	Related
		- Pindang (*food from the same origin city of Rendang*)	Related
		- Gulai (*food from the same origin city of Rendang*)	Related
Compress Model 3	Rendang (*Food name*)	- rendang (*lowercase of Rendang*)	Synonym
		- Dendeng (*food from the same origin city of Rendang*)	Related
		- Balado (*food from the same origin city of Rendang*)	Related
		- Pindang (*food from the same origin city of Rendang*)	Related
		- Gulai (*food from the same origin city of Rendang*)	Related

This also applies to several other words. We select 10 words that appear quite frequently in the Indonesian translation of Quran, then we calculate the accuracy of each model in returning the most similar words of the given word. The ten selected words and their accuracy results for each model are shown in Table 6 and Table 7, respectively.

Table 6. Selected frequent words from Indonesian translation of Quran for intrinsic evaluation.

ALLAH	BUMI	AZAB	KAUM	KITAB
IMAN	LANGIT	KAFIR	ZALIM	HATI

Table 7. FastText model’s accuracy returning most similar words.

Method	Accuracy (%)
FastText Original Model [23]	86.00
Compress Model 1	82.00
Compress Model 2	**88.00**
Compress Model 3	**88.00**

4.2 Extrinsic Evaluation for Compressed Model

For extrinsic evaluation, we evaluated the compressed models in a publicly available NLP task for the Indonesian language. Currently, there is no publicly available task specifically using Indonesian Quran translation. Therefore we chose Ukara Automatic Essay Scoring task [24] to evaluate compressed models. We used the same dataset, data split and experiment scheme as previous research [19]. We use Compress FastText models as a feature extractor for essay sentences before being inputted into the Neural Network with a single hidden layer. The hyper-parameter search process uses Optuna [25] and we also include the regularization technique proposed in the study, increasing batch [26].

Ukara dataset consists of two sets of problems. In this study, we only use data A. The evaluation results of Ukara's data A are shown in Table 8.

Table 8. Model performance on automatic essay scoring task.

Method	F1-Score
SBERT [14]	89.44
FastText Original	88.82
Compress Model 1	88.63
Compress Model 2	88.75
Compress Model 3	88.89

In Table 9, it appears that, in general, the FastText model is still not as good as SBERT for the case of Automatic Essay Scoring using a simple model. However, from the table, it appears that Compress Model 3 has the best results compared to other FastText models, although the differences are not so significant. This also indicates that the FastText model compression technique can maintain a good performance.

Based on the evaluations that have been conducted, in the next stage, we only use Compress Model 3 as the embedding model to change the representation of words in the query.

5. RESULTS AND DISCUSSION

5.1 Retrieval-system Evaluation

This evaluation focuses on evaluating the success rate of verse retrieval of the auto-completion system. The system is evaluated using two sets of data tests. Each data test contains pairs of a query and an intended verse that need to be retrieved. The first data test is synthetic data generated using the English translation of the Quran, as mentioned in sub-section 3.5. The second data test is related to data that contains Out-of-Vocabulary (OOV) words that we built by ourselves.

As mentioned, we try different stop-word threshold values in these experiments. We use retrieval evaluation on English Quran translation data to evaluate the stop-word threshold. We got 1000 occurrences as the best stop-word threshold based on model-retrieval performance on top-5 accuracy and top-10 accuracy. Results in the later part use 1000 occurrences as the stop-word threshold.

5.1.1 Retrieval Evaluation on English Quran Translation Data

For the first data test, we build it by randomly choosing 150 verses from the English translation of the Quran. Then, we translate them into Indonesian using the Google Translate API. From those 150 verses, we will randomly select 5-8 words that are not included in the stop-word list as a query. The system will return the ten verses closest to the given query. The illustration of this dataset processing and samples can be seen in Table 9.

We calculate the system's accuracy by checking whether the intended or correct verse is in the ten retrieved verses (top-10 accuracy), as given in the query. If the correct verse is one of the ten verses, the system is considered successful in retrieving the verse. In addition to top-10 accuracy, we also conduct the same evaluation to find top-5 accuracy. Top-5 accuracy is calculated when the correct verse is one of the five retrieved verses. The results of this evaluation are shown in Table 10.

Table 9. Sample queries of data test obtained from the English translation. Red-colored words are new words that do not appear in the original Indonesian translation.

Original Indonesian Translation	Original English Translation	English Translation Translated into Indonesian	Randomly Picking 5-8 Words
Kami menjadikan Al-Qur'an dalam bahasa Arab agar kamu mengerti.	Indeed, We have made it an Arabic Qur'an that you might understand.	Sesungguhnya Kami telah menjadikannya Al-Qur'an berbahasa Arab agar kamu mengerti.	Sesungguhnya Kami Al-Qur'an berbahasa Arab mengerti.
dengan membawa gelas, cerek dan sloki (piala) berisi minuman yang diambil dari air yang mengalir	With vessels, pitchers and a cup [of wine] from a flowing spring	Dengan bejana, kendi dan secangkir [anggur] dari mata air yang mengalir	bejana, secangkir [anggur] air mengalir
Wahai manusia! Kamulah yang memerlukan Allah; dan Allah Dialah Yang Mahakaya (tidak memerlukan sesuatu), Maha Terpuji.	O mankind, you are those in need of Allah, while Allah is the Free of need, the Praiseworthy	Wahai manusia, kamu adalah orang-orang yang membutuhkan Allah, sedangkan Allah adalah Yang Maha Bebas lagi Maha Terpuji	Wahai orang-orang membutuhkan Allah Maha Bebas Maha Terpuji.

Table 10. Top-5 accuracy and Top-10 accuracy for each method on test set built from English translation.

Method	Top-5	Top-10
Sentence + word vector	**55.33%**	**64.67%**
Sentence vector	38.67%	51.33%
Word matching	54.67%	62.67%

From Table 10, it appears that the use of word vectors to select the most relevant words prior to calculating the sentence vector affects the quality of the model's performance. Sentence method + word vector retrieval outperforms other methods.

The retrieval method with word matching achieved a fairly high score in Table 10 compared to the sentence vector method, which is almost similar to that in our proposed method. However, the evaluation process with the English translation scheme does not exploit the advantages of vector embedding in dealing with out-of-vocabulary words (OOV). Therefore, we tested these three methods in dealing with OOV cases by building a special test data that also includes OOV words, either synonyms, informal abbreviations or typos.

5.1.2 Retrieval Evaluation on OOV Data

This special data test that includes OOV words is used as the second part of system evaluation. This data test was created from two main sources. The first source is written lecture notes. The lecture notes are collected from several respondents who had taken notes during the Islamic lectures. We copied the part of the lecture notes that included a verse quote along with the letter number and verse number. The second source is obtained from the recall of several respondents who memorized the Quran. We asked them to remember some verses of the Indonesian translation and write down the verses. Verse paraphrasing, word abbreviations and writing errors usually occur during recall. In the case of the Quran note-taking, the process of OOV emergence is unavoidable. Some examples of data tests are shown in Table 11. The test data is publicly available and can be downloaded at https://bit.ly/iqrtest (4^{th} revision).

We use the test set for the three models and the results are shown in Table 12. In this table, it appears that the sentence + word vector embedding method is far outperforming word-matching retrieval. The sentence vector method produces the lowest performance. This can happen, because some quite long verses cause the average word vector for the entire verse not to represent the verse.

Table 11. Example of test data. The part highlighted in red is an example of OOV, because it is different or not in the original verse. The number next to it corresponds with the word in the original verse closest to it.

Original verse	Query
Dan (ingatlah) ketika **Lukman[1] berkata[2]** kepada anaknya, ketika dia memberi pelajaran kepadanya, "Wahai anakku! Janganlah engkau mempersekutukan Allah, sesungguhnya mempersekutukan (Allah) adalah benar-benar kezaliman yang besar."	Luqman[1] menasehati[2] anaknya
Wahai manusia! Sungguh, Kami telah menciptakan kamu dari seorang laki-laki dan seorang perempuan, kemudian Kami jadikan kamu **berbangsa-bangsa**[3] dan bersuku-suku agar kamu saling mengenal. Sesungguhnya yang paling mulia di antara kamu di sisi Allah ialah orang yang paling bertakwa. Sungguh, Allah Maha Mengetahui, Mahateliti.	manusia diciptakan berbeda-beda[3] untuk saling mengenal
Wahai orang-orang yang beriman! Bertakwalah **kepada**[4] Allah sebenar-benar takwa kepada-Nya dan janganlah kamu mati kecuali **dalam keadaan**[5] Muslim.	bertakwa kpd[4] Allah dan jangan mati selain mjd[5] muslim
Katakanlah, "Wahai hamba-hamba-Ku yang melampaui batas terhadap diri mereka sendiri! **Janganlah**[6] kamu berputus asa **dari rahmat**[7] Allah. Sesungguhnya Allah mengampuni dosa-dosa semuanya. Sungguh, Dialah Yang Maha Pengampun, Maha Penyayang.	jgn[6] putus asa utk meminta pd[7] Allah

Table 12. Top-5 accuracy and Top-10 accuracy for each method on test-set that included OOV.

Method	Top-5	Top-10
Sentence + word vector	**70.59%**	**76.47%**
Sentence vector	35.29%	47.06%
Word matching	55.89%	58.82%

5.2 Note-taking Application Prototype

The interface of the note-taking application prototype looks like in Figure 5. In Figure 5-left, users write queries starting with a tilde and choose a suggested verse from the top of the space. In Figure 5-right, after choosing a verse, it will automatically replace the query in the body of the note. Using a server with 8 virtual cores, the average retrieval takes about 2.2 seconds.

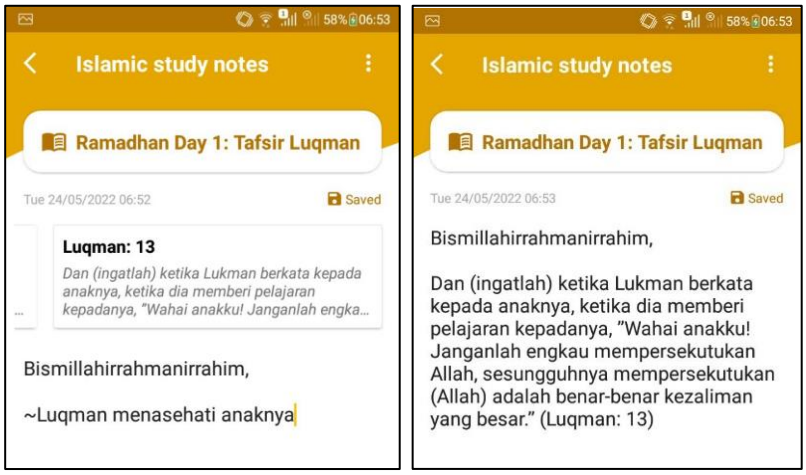

Figure 5. The interface when the user writes a query and gets recommendation (left). The interface when the recommendation auto-completes the query (right).

We tested this prototype on 32 respondents, consisting of Islamic lecturers and participants. The result showed that 96.9% of participants believe that smart Quran auto-completion in the note-taking

application can help the user in the note-taking process.

6. CONCLUSIONS

In this study, we propose an auto-completion system using semantic vector representation. To obtain vector embedding, we use Compress FastText, which is more efficient in storage and memory usage, so it is suitable to be used at the production stage. In the retrieval process, we propose a combination of word and sentence-vector embedding and cosine distance when searching for verses. This method proved to be more effective in the two types of evaluation performed compared to other methods.

In the future, much can still be developed, especially in terms of reliability at the production stage. The auto-completion system demands a model that can process queries in real time. For this reason, it is necessary to efficiently use resources and optimize algorithms to get the expected system's performance. The lack of available data might also affect the current system's retrieval result. We can use the current prototype or other ways to gather more data that fits in the Quran-retrieval domain.

ACKNOWLEDGEMENTS

This research was supported by the Directorate of Research and Community Service at Universitas Islam Indonesia through beginner research grant number 001/Dir/DPPM/70/Pen.Pemula/III/2022

REFERENCES

[1] B. J. Lee, "Smartphone Tapping *vs.* Handwriting: A Comparison of Writing Medium," The EuroCALL Review, vol. 28, no. 1, p. 15, DOI: 10.4995/eurocall.2020.12036, 2020.

[2] Z. Abu Bakar and N. Abdul Rahman, "Evaluating the Effectiveness of Thesaurus and Stemming Methods in Retrieving Malay Translated Al-Quran Documents," Lecture Notes in Computer Science, vol. 2911. pp. 653–662, 2003, DOI: 10.1007/978-3-540-24594-0_67.

[3] A. Aulia, D. Khairani and N. Hakiem, "Development of a Retrieval System for Al Hadith in Bahasa (Case Study: Hadith Bukhari)," Proc. of the 5th Int. Conf. on Cyber and IT Service Management (CITSM 2017), DOI: 10.1109/CITSM.2017.8089323, 2017.

[4] I. Humaini, T. Yusnitasari, L. Wulandari, D. Ikasari and H. Dutt, "Informatian Retrieval of Indonesian Translated Version of Al Quran and Hadith Bukhori &Muslim," Proc. of the 2018 Int. Conf. on Sustainable Energy, Electronics and Computing System (SEEMS 2018), DOI: 10.1109/SEEMS.2018.8687330, 2019.

[5] M. Adriani, J. Asian, B. Nazief, S. M. M. Tahaghoghi and H. E. Williams, "Stemming Indonesian: A Confix-stripping Approach," ACM Transactions on Asian Language Information Processing, vol. 6, no. 4, pp. 1–33, DOI: 10.1145/1316457.1316459, 2007.

[6] A. Z. Arifin, I. P. A. D. Mahendra and H. T. Ciptaningtyas, "Enhanced Confix Stripping Stemmer and Ants Algorithm for Classfying News Document in Indonesian Language," Proc. of the 5th Int. Conf. on Information & Communi. Technology and Systems, no. April 2014, pp. 149–158, 2009.

[7] D. K. Po, "Similarity Based Information Retrieval Using Levenshtein Distance Algorithm," Int. J. of Advances in Scientific Research and Engineering, vol. 06, no. 04, pp. 06–10, 2020.

[8] S. Wang and R. Koopman, "Semantic Embedding for Information Retrieval," Proc. of CEUR Workshop, vol. 1823, pp. 122–132, 2017.

[9] Y. Yuan, Improving Information Retrieval by Semantic Embedding, B.Sc. Essay, Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, Netherlands, [Online], Available: http://essay.utwente.nl/82070/, 2020.

[10] T. Mikolov, K. Chen, G. Corrado and J. Dean, "Efficient Estimation of Word Representations in Vector Space," Proc. of the 1st Int. Conf. on Learning Representations (ICLR 2013), arXiv: 1301.3781, DOI: 10.48550/arXiv.1301.3781, 2013.

[11] P. Bojanowski et al., "Enriching Word Vectors with Subword Information," Trans. of the Association for Computational Linguistics, vol. 5, pp. 135–146, DOI: 10.1162/tacl_a_00051, 2017.

[12] J. Devlin, M. W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT 2019), vol. 1, pp. 4171–4186, 2019.

[13] M. Zidny Naf'an, Y. Sari and Y. Suyanto, "Word Embeddings Evaluation on Indonesian Translation of AI-Quran and Hadiths," IOPscience, vol. 1077, no. 1, p. 012025, DOI: 10.1088/1757-899X/1077/1/012025, 2021.

[14] A. Aziz Altowayan and A. Elnagar, "Improving Arabic Sentiment Analysis with Sentiment-specific Embeddings," Proc. of the 2017 IEEE Int. Conf. on Big Data, vol. 2018-Jan., pp. 4314–4320, DOI:

10.1109/BigData.2017.8258460, 2017.
[15] F. Alam, M. Afzal and K. M. Malik, "Comparative Analysis of Semantic Similarity Techniques for Medical Text," Proc. of the Int. Conf. on Information Networking, vol. 2020-Jan., pp. 106–109, DOI: 10.1109/ICOIN48656.2020.9016574, 2020.
[16] M. S. Saputri, R. Mahendra and M. Adriani, "Emotion Classification on Indonesian Twitter Dataset," Proc. of the 2018 Int. Conf. on Asian Language Processing (IALP 2018), pp. 90–95, DOI: 10.1109/IALP.2018.8629262, 2019.
[17] R. A. Rajagede and R. P. Hastuti, "Stacking Neural Network Models for Automatic Short Answer Scoring," Proc. of 5th Int. Conf. on Information Technology and Digital Applications (ICITDA 2020), vol. 1077, pp. 0–6, Yogyakarta, Indonesia, 2020.
[18] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings Using Siamese BERT-networks," Proc. of 2019 Conf. on Empirical Methods in Natural Language Processing and 9th Int. Joint Conf. on Natural Language Processing (EMNLP-IJCNLP 2019), DOI: 10.18653/v1/d19-1410, 2020.
[19] R. A. Rajagede, "Improving Automatic Essay Scoring for Indonesian Language Using Simpler Model and Richer Feature," Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics and Control, vol. 6, no. 1, pp. 11–18, DOI: 10.22219/kinetik.v6i1.1196, 2021.
[20] D. Dale, "Compress-FastText," Github Repository, [Online], Available: https://github.com/avidale/compress-fasttext, Accessed: Jun. 01, 2022.
[21] G. Nasution, "Quran - API," Github Repository, [Online], Available: https://github.com/gadingnst/quran-api, 2022.
[22] H. A. Robbani, "Sastrawi Python," Github Repository, [Online], Available: https://github.com/har07/Py Sastrawi, 2018.
[23] E. Grave et al., "Learning Word Vectors for 157 Languages," Proc. of the 11th Int. Conf. on Language Resources and Evaluation (LREC 2018), pp. 3483–3487, Miyazaki, Japan, 2019.
[24] G. B. Herwanto et al., "UKARA: A Fast and Simple Automatic Short Answer Scoring System for Bahasa Indonesia," Proc. of the Int. Conf. on Educat. Assessment and Policy, DOI: 10.26499/iceap.v2i1.95, 2018.
[25] T. Akiba, S. Sano, T. Yanase, T. Ohta and M. Koyama, "Optuna: A Next-generation Hyper-parameter Optimization Framework," Proc. of the ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, DOI: 10.1145/3292500.3330701, 2019.
[26] S. L. Smith, P. J. Kindermans, C. Ying and Q. V. Le, "Don't Decay the Learning Rate, Increase the Batch Size," Proc. of the 6th Int. Conf. on Learning Representations (ICLR 2018), [Online], Available: https://openreview.net/forum?id=B1Yy1BxCZ, 2018.
[27] R. Sennrich, B. Haddow and A. Birch, "Improving Neural Machine Translation Models with Monolingual Data," Proc. of the 54th Annual Meeting of the Association for Computational Linguistics, pp. 86-96, Berlin, Germany, 2016.
[28] D. R. Beddiar, M. S. Jahan and M. Oussalah, "Data Expansion Using Back Translation and Paraphrasing for Hate Speech Detection," Online Social Networks and Media, vol. 24, Article no. 100153, 2021.

ملخص البحث:

في هذا البحث، يجري تصميم وتجربة نظام للاسترجاع الاوتوماتيكي للألفاظ والجُمل في الآيات القرآنية، بناءً على التّرجمة الإندونيسية للقرآن الكريم، من شأنه ان يساعد المشاركين في حضور المحاضرات لاكتساب المعرفة الاسلامية في تدوين ملاحظاتهم أثناء تلك المحاضرات. حيث يقوم المستخدم بكتابة كلماتٍ معينة أو أجزاء من الآيات الكريمة التي يستمع إليها، وبناءً على ذلك يعمل النّظام على استرجاع الآيات القرآنية الأقرب الى تلك التي قام المستخدم بادخالها الى النّظام. في هذه الدراسة، استخدمنا نموذج (FastText) وقمْنا بحساب مسافة جيب التّمام بين الطّلب المقدّم من المستخدم والآيات القرآنية من أجل عملية الاسترجاع. كذلك قمْنا بخطواتٍ لجعل النّظام مثالياً، وأجرينا تقييماً للنّظام من خلال فحْص مدى قُرب النّتائج من الواقع الفعلي. وبلغت الدّقّة التي حصلنا عليها (70.59%) في حالة أوّل (5) آيات مُسترجَعة، و (76.47%) في حالة أوّل (10) آيات مُسترجَعة.