

AI002.P11 Final Report,  
Artificial Intelligence Thinking PhD. Ngo Duc Thanh,  
Group 16,  
English vocabulary recommendation for learning

Nguyen Tran Nhat Trung  
23521684

Pham Hai Dang  
23520233

Hoang Hai Duong  
23520345

January 26, 2025

**Abstract**

This project introduces an approach to English vocabulary (*e.g. flashcard, question,...*) recommendation *for learning* by leveraging the BEIT3 model for text embeddings and integrating the Oxford Dictionary as a source of word-definition pairs. The BEIT3 model, a state-of-the-art text-to-text foundation model, is employed to calculate embeddings of word definitions, enabling semantic representation of vocabulary in a dense vector space. Our methodology enhances personalization by incorporating contextual data such as user hobbies, career aspirations, and web search history, which are embedded into the same vector space. This personalization ensures the recommendation of words that are not only semantically related but also contextually relevant to the user's interests and needs.

## Contents

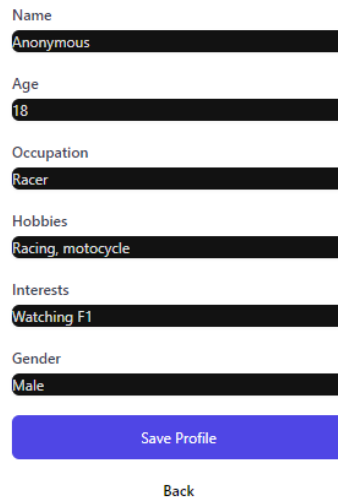
<b>1</b>	<b>Problem statement</b>	<b>3</b>
1.1	Input: . . . . .	3
1.2	Output[d10] . . . . .	5
1.3	Constraint: . . . . .	5
1.4	Requirement: . . . . .	6
<b>2</b>	<b>Decomposition</b>	<b>7</b>
2.1	Extract features from user's provided info . . . . .	7
2.2	Adapting to user's interests . . . . .	8
2.3	Measure frequency for each word. . . . .	10
2.4	Measure user's confidence. . . . .	11
2.5	Rescore and ranking. . . . .	11
<b>3</b>	<b>Evaluation</b>	<b>12</b>
3.1	Type-Token Ratio (TTR) . . . . .	12
3.2	Confidence score . . . . .	12
<b>4</b>	<b>Algorithm</b>	<b>14</b>
4.1	Front-end . . . . .	14
4.2	Back-end . . . . .	14
4.2.1	Feature Vector Calculation for Bio and Interest . . . . .	14
4.2.2	Word Similarity Computation . . . . .	15
4.2.3	Word Frequency Calculation . . . . .	15
4.2.4	Score Calculation and Question Generation . . . . .	16
4.2.5	Combining All Steps . . . . .	16

<b>5</b>	<b>Computational Thinking Application</b>	<b>17</b>
<b>6</b>	<b>Ethics Social</b>	<b>20</b>
6.1	Privacy Problem . . . . .	20
6.2	Security Problem . . . . .	21
6.3	Copyright Problem . . . . .	22

# 1 Problem statement

## 1.1 Input:

- **User provided information[d1]:** such as **age**, **occupation**, **hobbies**, **interests**, and **gender**, is processed through a detailed feature extraction pipeline to ensure structured and actionable insights. **Age** is represented numerically (e.g., 18), or can be describe as text as categories such as "young", "adult" or "senior". **Occupation** titles are standardized into broader categories or just a title of job (e.g., "Technology", "Engineer"). **Hobbies** are text describe user's hobbies (e.g., "Football", "Sports"). **Interests** are things that users frequently view and like, from which we can know their needs (e.g., "Political", "Gaming news"). **Gender** is a string so that can be used by people in specific community and culture (e.g., "Male", "Female"). For example:



A screenshot of a user profile form. It contains several input fields with labels on the left and values inside the fields. The fields are: Name (Anonymous), Age (18), Occupation (Racer), Hobbies (Racing, motorcycle), Interests (Watching F1), and Gender (Male). Below these fields is a blue button labeled 'Save Profile' and a smaller 'Back' link.

Field	Value
Name	Anonymous
Age	18
Occupation	Racer
Hobbies	Racing, motorcycle
Interests	Watching F1
Gender	Male

Save Profile

Back

Figure 1: User's provided infomation example

- **User's learning data[d2]:** User learning data comprises the user's **word\_id**, the **score** they obtained, and the **time** at which that score was achieved. This data is instrumental in calculating *penalties*, which in turn enhances the diversity and effectiveness of flashcard recommendations. By analyzing user performance over time, the system can identify areas where the user may need additional practice and tailor the flashcards accordingly. This personalized approach can significantly improve learning outcomes. This data will be filled automatically during the learning process. For example:

```
historyUrls : Array (10)
learningData : Array (19)
  0: Object
    word_id : 74056
    time : 1735513750
    point : 0.9
  1: Object
    word_id : 175534
    time : 1735513752
    point : 0.8
  2: Object
    word_id : 74056
    time : 1735513768
    point : 0.3
  3: Object
    word_id : 175534
    time : 1735513792
    point : 0.2
  4: Object
    word_id : 61929
    time : 1735513808
    point : 0.9
```

Figure 2: User's learning data example

- **User's browser history:** The URLs (*strings*) obtained from the user's browsing history constitute a valuable data for the application's recommendation system, enabling it to discern the user's interests through techniques such as URL parsing, topic modeling, and collaborative filtering, ultimately facilitating the provision of highly personalized and engaging content recommendations. This input will automatically track from the browser. For example:

```

{
  "genre": "stranger",
  "name": "hi",
  "historyUrls": Array (50)
    0: "https://translate.google.com/translate?hl=en&tl=vi&text=%20Error%20in%20invocat..."
    1: "https://stackoverflow.com/questions/26491360/chrome-extension-executes-"
    2: "https://developer.chrome.com/docs/extensions/reference/api/scripting"
    3: "https://www.facebook.com/"
    4: "https://github.com/whatwg/fetch/issues/551"
    5: "https://www.reddit.com/r/nestjs/comments/122zyyp/how_to_return_json_obj..."
    6: "https://translate.google.com/details?hl=en&tl=vi&text=%20Error%20in%20..."
    7: "https://www.google.com/"
    8: "https://stackoverflow.com/questions/1894792/how-to-determine-whether-a-"
    9: "https://www.google.com/search?q=typescript+check+object+has+property&..."
    10: "https://www.youtube.com/watch?v=ef4Vw-Senc"
    11: "https://www.canva.com/posters/templates/research/"
    12: "https://www.google.com/search?q=canva+scientific+poster&scas_esv=47dd34..."
    13: "https://www.facebook.com/profile.php?id=100016292749683"
    14: "https://www.youtube.com/watch?v=V2X5WwMFQ"
    15: "https://www.youtube.com/watch?v=VcbzYqM5_to"
    16: "https://www.youtube.com/watch?v=13by3is839I"
    17: "https://www.youtube.com/"
    18: "https://www.facebook.com/photo?fbid=122134075934548803&set=pcb.1221340..."
    19: "https://www.facebook.com/photo?fbid=122134074612548803&set=pcb.1221340..."
    20: "https://www.facebook.com/photo/?fbid=122134075934548803&set=pcb.122134..."
    21: "http://127.0.0.1:5173/#/login"
    22: "http://127.0.0.1:5173/#/card"
    23: "https://github.com/trungdangtapcode/Japanese-Learning-System"
    24: "https://github.com/trungdangtapcode/Japanese-Learning-System"
  }
}

```

Figure 3: User's browser history example

- **User's translated text:** is a literal "corpus" of English words that users have translated. We developed a *small browser extension* that facilitates seamless text translation while users browse the web or read online articles. This extension automatically captures and stores translated text into the *local storage*, creating a personalized dataset that highlights areas where users face linguistic challenges. The **automated** storage mechanism ensures the translation history is continuously updated, enabling the system to adaptively track and address the user's learning needs. For example:

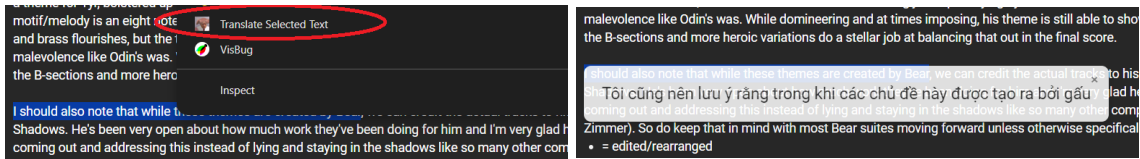


Figure 4: Translated text example (built-in extension)

Figure 5: Translated text example (after translating, save the English text)

```

> chrome.storage.local.get("savedEnglishTexts", async (result) => {
  console.log(result)
})
< undefined

{
  "savedEnglishTexts": Array(11)
    0: "Merge Classes in React with Tailwind CSS and Tailwind-merge"
    1: "Hi Sonny, You are welcome! Thanks, Stefan!"
    2: "I'm trying to return a JSON object from my Next endpoint called 'template', however, it's returning rendered html instead."
    3: "Scripting"
    4: "from async endpoint"
    5: "from async endpoint"
    6: "Imagine Dragons - Outbreak (Performance Video)"
    7: "Unreleased Soundtrack"
    8: "Music composed by Bear McCreary."
    9: "Injection"
    10: "method"
    11: "method"
    12: "method"
    13: "method"
    14: "I ran into this restriction while building HTTP functions for a library. I think this should be reconsidered because sometimes pos..."
    15: "I ran into this restriction while building HTTP functions for a library. I think this should be reconsidered because sometimes pos..."
    16: "method cannot have body"
  }
}

```

Figure 6: Translated text example (the translated text saved on local storage of chrome)

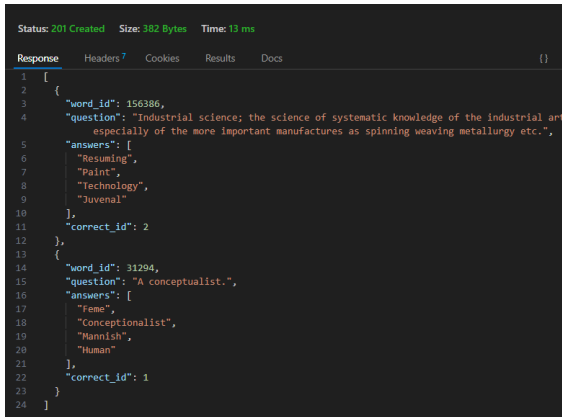
- **Dictionary data (optional):** it refers to a structured repository of terms, definitions. It functions as a corpus that supports retrieval operations, enabling the generation of relevant flashcards based on user queries or knowledge gaps. For example:

```
[8]: [(['Unresisted', 'Not resisted; unopposed.'),
      ('Duressor', 'One who subjects another to duress'),
      ('Here', 'Hair.'),
      ('Sunglass',
       'A convex lens of glass for producing heat by converging the sun\'s rays into a focus.'),
      ('Waterscape', 'A sea view; -- distinguished from landscape.'),
      ('Hamate', 'Hooked; bent at the end into a hook; hamous.'),
      ('Termer',
       'One who resorted to London during the law term only in order to practice tricks to carry on intrigues or the like.'),
      ('Intermission',
       'The temporary cessation or subsidence of a fever; the space of time between the paroxysms of a disease. Intermission is an entire cess.
f fever.'),
      ('Interlining',
       'Correction or alteration by writing between the lines; interlineation.'),
      ('Omniparity', 'Equality in every part; general equality.')]
```

Figure 7: Dictionary data example

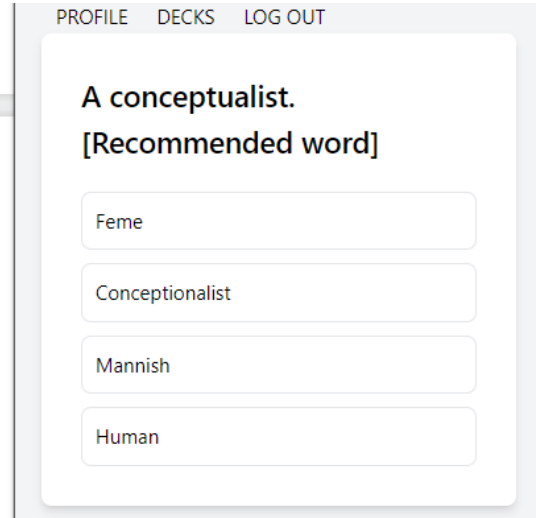
## 1.2 Output[d10]

A *single* object is the question with four possible answer choices, only one of which is correct (also the index for the correct one). To ensure variety and prevent memorization of specific card orders, the system employs a **randomized** output mechanism. This means that given the same user data and information retrieved from the database, the specific question presented on a flashcard may vary across different sessions or even within the same session. However, the system is designed to prioritize the presentation of flashcards that are most relevant and beneficial to the user’s learning progress. This prioritization is achieved through a probabilistic approach, where flashcards that address the user’s areas of weakness or align with their learning objectives are more likely to be presented. This dynamic and adaptive approach enhances the learning experience by continuously challenging the user and providing a personalized learning path.



```
Status: 201 Created Size: 382 Bytes Time: 13 ms
Response Headers Cookies Results Docs
1 {
2   {
3     "word_id": 156386,
4     "question": "Industrial science; the science of systematic knowledge of the industrial art,
5     especially of the more important manufactures as spinning weaving metallurgy etc.",
6     "answers": [
7       "Resuming",
8       "Paint",
9       "Technology",
10      "Juvenal"
11    ],
12    "correct_id": 2
13  },
14  {
15    "word_id": 31294,
16    "question": "A conceptualist.",
17    "answers": [
18      "Feme",
19      "Conceptionalist",
20      "Mannish",
21      "Human"
22    ],
23    "correct_id": 1
24  }
25 }
```

Figure 8: Output example (backend)



PROFILE DECKS LOG OUT

**A conceptualist.**  
**[Recommended word]**

Feme

Conceptionalist

Mannish

Human

Figure 9: Output example (frontend)

## 1.3 Constraint:

### 1. Information as English Text

- **Constraint:** All input information provided by the user must be in English text.
- **Reason:** Since the vocabulary recommendation system is built upon English text processing and embeddings generated by the BEIT3 model, using English ensures compatibility and accurate feature extraction. Non-English input could introduce errors or inconsistencies in the recommendation process.

- **Impact:** This constraint ensures that the system can effectively process the user's input, whether it's their hobbies, career details, or translations, and generate meaningful vocabulary suggestions tailored to their needs.

## 2. Information Needs to Be Honest

- **Constraint:** Users must provide honest information about their interests, career, hobbies, and other details.
- **Reason:** The accuracy and personalization of the recommendations rely heavily on the quality of the user-provided data. If the information is dishonest or inaccurate, the system may recommend irrelevant vocabulary, undermining its effectiveness and reducing the user's satisfaction.
- **Impact:** Honest data enables the system to better align vocabulary recommendations with the user's goals, making the learning process more effective and relevant to their context.

## 3. Chrome Extension to Access Browser History

- **Constraint:** The application must have access to the user's browser history through a Chrome extension.
- **Reason:** Browser history provides valuable contextual data about the user's online activities, such as websites visited and topics of interest. Extracting metadata like keywords from URLs enables the system to recommend vocabulary that aligns with the user's browsing habits.
- **Impact:** By leveraging browser history, the system can identify frequently visited topics and domains, ensuring the recommended vocabulary is not only personalized but also practical for the user's everyday use.

## 1.4 Requirement:

1. **Diverse Recommendations:** The vocabulary recommendations should exhibit a high level of diversity to ensure users are exposed to a broad range of words.
  - Diversity is measured using the **Type-Token Ratio (TTR)**[3.1].
  - **Requirement:** The TTR for the recommended vocabulary set must be greater than **0.7**.
  - **Reason:** A higher TTR ensures that the recommended words are unique and reduce redundancy, making the learning process more engaging and effective.
2. **Positive User Feedback:** The system must prioritize recommendations that align with the user's preferences and learning experience.
  - Users rate the recommended vocabulary on a scale of 1 to 5. Helps to improve the user's score during the learning process, which is calculated as in [3.2].
  - **Requirement:** A word is considered successfully recommended if the user gives it a rating of **3 or higher**.
  - **Reason:** This ensures that the system is actively refining its recommendations to match the user's needs and preferences based on feedback.
3. **Efficient Retrieval and Response Time:** The application must be optimized for fast processing and response.
  - **Requirement:** The total time for vocabulary retrieval and server response must not exceed **5 seconds**.
  - **Reason:** A quick response time ensures a seamless user experience and minimizes frustration caused by delays, encouraging consistent use of the application.

**Summary of Requirements:** These requirements are designed to balance **effectiveness**, **user satisfaction**, and **system performance**. By ensuring diversity through TTR, aligning recommendations with user feedback, and maintaining quick response times, the system can deliver a high-quality personalized learning experience.

## 2 Decomposition

In designing a flashcard recommendation system, the problem can be decomposed into two perspectives: aggregation and individual diversity. From the aggregation perspective, the goal is to ensure that the recommendation system covers a wide range of topics across all users, promoting comprehensive coverage of the flashcard database. This is particularly useful in collaborative learning environments where diverse content exposure is beneficial. On the other hand, the individual perspective focuses on tailoring recommendations to maximize the balance between quality and diversity for a specific user. Here, the system should optimize for relevance by prioritizing flashcards that address the user's current learning goals while introducing new or less familiar topics to enhance learning breadth. Achieving this trade-off requires personalized algorithms that adapt to a user's progress and preferences, ensuring effective and diverse learning experiences. To address these dual perspectives, the problem can be decomposed into the following steps:

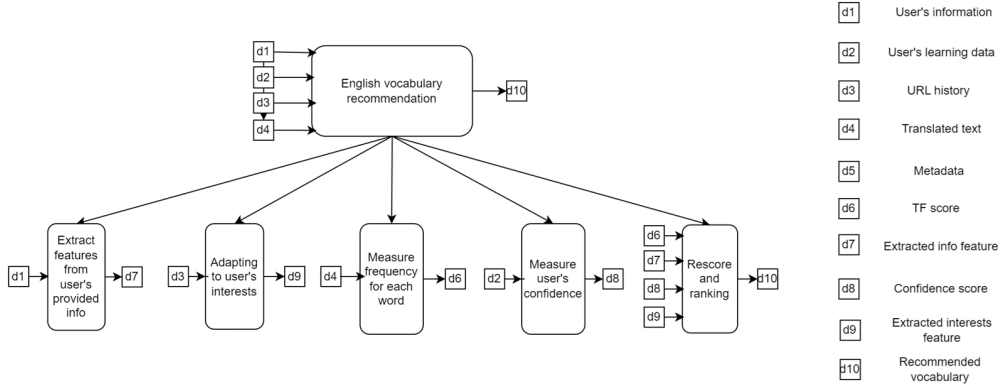


Figure 10: Decomposition Tree

### 2.1 Extract features from user's provided info

- **Input[d1]:** The user's provided data, which includes key demographic and behavioral attributes such as age, occupation, hobbies, interests, and gender. This raw input serves as the foundation for deriving actionable insights and generating personalized outputs.
- **Output[d7]:** The extracted feature that represent the input information in a machine-readable format. These features are tailored to encapsulate the user's profile in a way that can be leveraged by downstream algorithms or recommendation systems.

The core focus of this project lies in addressing the challenge of acquiring user-provided information, processing it into semantic embedding vectors, and subsequently leveraging these embeddings to generate tailored flashcard content. This solution aims to enhance user engagement and learning outcomes by providing content that is semantically aligned with user input. A detailed depiction of the semantic embedding process is illustrated in 11, which outlines the multi-step transformation of raw input into meaningful vector representations.

To achieve high-quality embeddings, we experimented with various deep learning models while maintaining consistent system hyperparameters, such as batch size, learning rate, optimizer configuration, and evaluation metrics. The performance outcomes of these models are summarized in 1. Through rigorous comparative analysis, the BEIT model emerged as the optimal choice. Its ability to generate robust contextualized representations made it a standout among competing architectures. The BEIT model's effectiveness stems from its transformer-based architecture, originally designed for image processing, which we adapted for text-based semantic embeddings. Its self-attention mechanism captures long-range dependencies, ensuring the creation of high-dimensional embeddings that encapsulate nuanced semantic information.

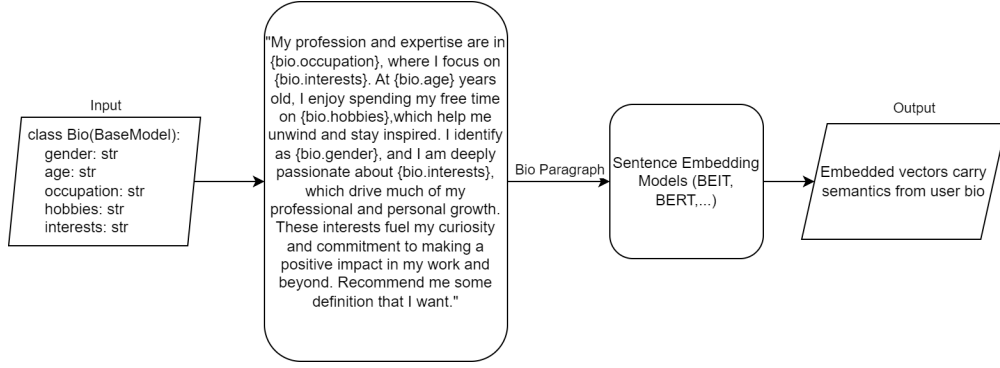


Figure 11: Extract features from user’s provided info

Model (r-5)	BEIT	BERT
Person		
Person A: {gender: "male", age: "20", occupation: "game developer", hobbies: "playing game", interests: "news about game, developing"}	Game Development, Gaming, News, Developer, Programming	Modeling, Tech, Gamer, Updates, Design
Person B: {gender: "female", age: "20", occupation: "Artist", hobbies: "Painting, sketching", interests: "Art trends, creative techniques, art exhibitions"}	Art, Creativity, Painting, Sketching, Artist	Trends, Expression, Design, Exhibitions, Note

Table 1: Compare BEIT and BERT on recommending word

## 2.2 Adapting to user’s interests

- **Input[d3]:** URLs from analyzing the websites user visit. We can understand their interests. This allows us to suggest relevant and engaging content, creating a more personalized experience. This browsing information will be automatically collected from the user’s browser
- **Output[d9]:** The extracted features represent the input information in a machine-readable format. These features are tailored to encapsulate the user’s history in a way that can be leveraged by downstream algorithms or recommendation systems.

The system must track the user’s browsing data in real-time, responding to every change in the user’s search history. This requires implementing a browser extension or script capable of detecting URL changes (through *onHistoryStateUpdated*). Each detected change needs to be communicated to the backend promptly, ensuring minimal latency and synchronization between the front-end tracker and the back-end processor. For example, when a user navigates from *example.com* to *news.com*, this transition must be recorded and transmitted efficiently.

The URLs must be analyzed and filtered to remove invalid or irrelevant entries, such as dynamically generated tracking links, advertisements, or non-content URLs (e.g., authentication pages or error pages). This step necessitates employing a robust URL validation mechanism and possibly integrating a domain categorization service to identify valid content-rich sources. For instance, retaining [en.wikipedia.org/wiki/Machine\\_learning](https://en.wikipedia.org/wiki/Machine_learning) would ensure relevant data is processed. And here some invalid URL:



Invalid URL	Reason
http://localhost:8080	These URLs are only accessible within a specific network and aren't accessible from outside the local environment.
https://example.com/app#fragment	If the metadata is generated dynamically via JavaScript, tools that don't execute JavaScript (e.g., simple crawlers or requests in Python) will not be able to retrieve it.
https://nonexistentdomain12345.com	If the URL points to a domain or resource that no longer exists, metadata retrieval will fail.
https://securewebsite.com/private-area	If the URL requires authentication (e.g., Facebook account) to access, metadata retrieval will fail without proper credentials.

Table 2: Example of Invalid URL

For each valid URL, the system must retrieve metadata such as the webpage title, description, and keywords. This involves parsing the website's HTML using web scraping or metadata extraction tools and ensuring compliance with web crawling guidelines

URL	Title	Description	Keywords
https://www.freecodecamp.org/news/ai-in-agriculture-book/	AI in Agriculture: How AI-Enhanced Farming Can Increase Crop Yields [Full Book]	Artificial intelligence is revolutionizing the agriculture industry, paving the way for a future of smarter, more efficient farming practices. Imagine a world where crops are grown with precision and care, maximizing yields like never before. With AI...	Artificial intelligence, future, machine learning
https://www.youtube.com/watch?v=66fXm8lfOLE	Ban tin UIT News thang 1/2025 — UIT RON RANG DON NAM MOI XUAN AT TY 2025 - YouTube	Thang 1, Truong Dai hoc Cong nghe Thong tin co nhung noi dung noi bat nhu sau:- KY HOP HOI DONG TRUONG DAI HOC CONG NGHE THONG TIN LAN THU 17 (KHOA I) - LE CO...	video, sharing, camera phone, video phone, free, upload

Table 3: Example of Metadata of URL

The extracted metadata must be converted into a numerical embedding vector that semantically represents the content. The embeddings from multiple URLs are then aggregated using an Exponentially Weighted Moving Average (EWMA) mechanism, emphasizing recent browsing behavior while retaining the influence of past activities. For example, if the user visits websites related to "Data Science" followed by "Machine Learning," the aggregated embedding should gradually shift towards representing their evolving interests. This pipeline not only requires seamless integration across multiple components but also demands computational

efficiency and scalability to handle diverse and dynamic browsing behaviors.

$$\mathbf{s}_t \leftarrow \alpha \mathbf{v}_t + (1 - \alpha) \mathbf{s}_{t-1}$$

$$\mathbf{s}_{corrected} \leftarrow \frac{s_t}{1 - \alpha^t}$$

Where:

- $\mathbf{s}_t, \mathbf{s}_{t-1}$  is the EWMA vector at time  $t, t - 1$ .
- $\mathbf{v}_t$  is the input vector at time  $t$  (feature extracted new URL).
- $\alpha$  (where  $0 < \alpha \leq 1$ ) is the smoothing factor that determines the weight given to recent versus past observations.

### 2.3 Measure frequency for each word.

- **Input[d4]:** is a "corpus", is a concatenated string derived from the translatedText field within a data comprising user-generated translations.
- **Output[d6]:** is a frequency distribution of words within the vocabulary. Specifically, for each unique word in the corpus, the output should provide its *Term Frequency (TF)* score. The overarching objective is to identify the words that exhibit the highest frequency within user-generated translations, thereby pinpointing the vocabulary items that users consistently encounter and potentially struggle with.

A challenge here is ensuring that the data is clean and uniform, as user-translated sentences often contain inconsistencies like extra spaces, punctuation errors, and variations in word case. These inconsistencies can skew the term frequency (TF) calculations. For example, consider users translate sentences such as:

- "The cat sat on the mat."
- "A CAT sat on the mat!"
- These sentences, when concatenated, produce the corpus: "The cat sat on the mat. A CAT sat on the mat!". Preprocessing should include normalization (e.g., converting to lowercase), tokenization (splitting into words), and removal of stopwords (e.g., "the", "on", "a"). After preprocessing, the corpus becomes: "cat sat mat cat sat mat".

To solve this, develop a preprocessing pipeline that normalizes the data, splits it into tokens, and removes irrelevant terms. Tools like *nlTK* or *spaCy* (in Python) can automate this process.

Once the corpus is preprocessed, the next task is to create a vocabulary—a set of unique words. This is crucial for calculating the TF score, as each word in the vocabulary needs to be assigned a score. A naive approach would be to extract every unique token from the corpus.

For example, if the processed corpus is "cat sat mat cat sat mat", the vocabulary is "cat", "sat", "mat". Tools like Python's `collections.Counter` can help generate this vocabulary efficiently. To refine the vocabulary further, consider excluding words with low document frequency across users, as they contribute little to the goal of identifying common translation difficulties.

Then, we compute the TF score - it quantifies how frequently each word appears in the corpus relative to the total number of words. The formula for TF is

$$TF(\omega) = \frac{\text{Number of occurrences of } \omega}{\text{Total number of words in the corpus}}$$

This metric provides insight into which words users often translate, indirectly revealing the words they encounter frequently. For example, in the corpus "cat sat mat cat sat mat mat", the total word count is 6. The TF scores are calculated as follows:

- For "cat":  $TF = \frac{2}{7} = 0.29$

- For "sat":  $TF = \frac{2}{7} = 0.29$
- For "mat":  $TF = \frac{3}{7} = 0.43$

The final step is to integrate the calculated TF scores into a recommendation system. Words with high TF scores are **prioritized**, as they are likely to indicate translation difficulties. You might combine the TF score with other metrics, such as user error rates or context-specific difficulty, to produce a comprehensive recommendation score.

## 2.4 Measure user's confidence.

- **Inputd2:** User learning data, including list of word\_ids, scores, and timestamps, informs penalty calculations to enhance flashcard recommendations. By analyzing performance trends, the system identifies areas needing improvement and adapts flashcards for effective learning.
- **Outputd8:** is structured as a dictionary, where each word\_id is associated with a confidence\_score that reflects the user's current proficiency level with the corresponding vocabulary flashcard. A higher confidence\_score indicates greater familiarity or mastery of the word, thereby reducing the likelihood of its recommendation in subsequent practice sessions. This scoring mechanism ensures an adaptive learning experience, prioritizing words that require more attention while gradually phasing out those already mastered. The calculation of the final scores from the input will be discussed based on the formula in the section 3.2.

## 2.5 Rescore and ranking.

- **Input:** The input is the aggregation of the outputs of the above parts including: Extracted info feature[d7], extracted interests feature[d9], TF Score[d6], confidence score[d8].
- **Output[d10]:** generated flashcard where each card presents a single question with four possible answer choices, only one of which is correct.

The recommendation task involves determining the most relevant word and its corresponding definition based on a set of outputs of subproblems. This problem is modeled as a weighted combination of vector embeddings, further refined by term frequency (TF) scores and penalized using confidence scores. Given two input vectors,  $v_{bio}$ [d7],  $s_{corrected}$ [d9], derived from distinct sources or contexts, we compute a weighted combination to form a single representative vector:

$$v_c = w_{bio}v_{bio} + w_{history}s_{corrected}$$

where  $w_{bio}, w_{history}$  are scalar weights that control the influence of each input vector. The weights can be tuned manually or learned through optimization techniques depending on the specific application. Then each word and its definition in *vocabulary* are embedded into the same vector space using same model. The cosine similarity between the combined vector  $v_c$  and the feature vector of a candidate word  $\omega$  is  $v_\omega$  (and its associated definition) is computed to measure semantic relevance.

To account for the importance of frequently occurring terms, a term frequency ( $TF$ ) score is added as a bonus to the cosine similarity score. Let  $TF_\omega$  denote the  $TF$  score for a candidate word  $\omega$ . The adjusted score will be added a *penalty* called confidence scores - it's represent the uncertainty in the input data. A higher confidence score indicates greater uncertainty and requires penalization. Previously we used Reciprocal Rank Fusion (RRF) but it is discretized by rank, losing the continuous meaning of similarity. The final score for each candidate is computed as:

$$FinalScore_\omega = [Similarity(v_c, v_\omega) + w_{TF}TF_\omega] \cdot e^{-\frac{confidence_\omega}{\tau_{conf}}}$$

After the experiment, we decided to choose the default hyperparameters of the system as follows:

Hyperparameter	Value
$\alpha$	0.1
$w_{bio}$	0.6
$w_{history}$	0.4
$w_{TF}$	2.0
$\tau_{conf}$	86400

Table 4: Default hyperparameter

### 3 Evaluation

The assessment of the system arises due to the fact that it is based on human interaction and is therefore dynamic, not static. Because the system runs in real-time, user-interactive mode, it cannot be evaluated like traditional systems using predefined, fixed datasets. For that reason, traditional evaluation methods which rely on static data cannot be applied here. Rather, the performance of this system has to be assessed using controlled experiments and a bunch of iterative tests that mimic actual usage situations.

These tests are meant to show how well the system works in real life, concentrating on the user and the flexibility of the suggestions. The measuring steps are also organized by a list of clear goals that look at different parts of the system, like how right the recommendations are, how much users get involved, and what learning results come from using the suggested flashcards. Apply all these test methods and metrics of performance to give an evaluation that will be close to reality for the system, in terms of vocabulary improvement. Below are the metrics to evaluate the effectiveness of the system:

#### 3.1 Type-Token Ratio (TTR)

Type-Token Ratio (TTR) is a measure of lexical diversity in a set of recommended vocabulary. It is calculated as the ratio of unique words (types) to the total number of words (tokens) in a dataset. A higher TTR indicates a more diverse and varied vocabulary, while a lower TTR suggests repetition and limited variety.

In our context, TTR is used to evaluate how well the system balances the diversity of recommended words while keeping the vocabulary relevant to the user’s interests, translations, and browsing history. This ensures that users are exposed to a wide range of vocabulary suited to their personal learning needs.

$$TTR = \frac{\text{Number of unique words (types)}}{\text{Total number of words (tokens)}}$$

- **Set 1:** ["career", "job", "interest", "skill", "profession"]
  - Unique words: 5
  - Total words: 5
  - **TTR:  $5/5 = 1.0$**  (Highly diverse vocabulary)
- **Set 2:** ["translate", "translation", "translation", "translate", "language"]
  - Unique words: 3 ("translate", "translation", "language")
  - Total words: 5
  - **TTR:  $3/5 = 0.6$**  (Moderate diversity with repetition)

#### 3.2 Confidence score

The confidence\_score metric comes from analyzing user-specific learning data, which includes a complete dataset of word identifiers, associated scores, and timestamps. These parameters represent important inputs into the calculation process, through which information is delivered to apply adaptive penalty mechanisms. Penalty mechanisms are made dynamic based on temporal and performance trends seen in the user’s learning history to enhance the recommendation engine, making flashcard suggestions maximum personalized to

optimize retention and engagement. Thus, the system by this methodology gets in line with established principles of spaced repetition and adaptive learning to become more effective in supporting vocabulary acquisition. Confidence\_score calculation representation formula:

$$\text{confidence\_score}_{\text{userData}, \text{timeNow}}(\text{word\_id}) = \sum_{(id, \text{point}, \text{time}) \in \text{userData}} 1_{\text{word\_id} == id} \cdot \frac{\text{point}}{\text{timeNow} - \text{time}}$$

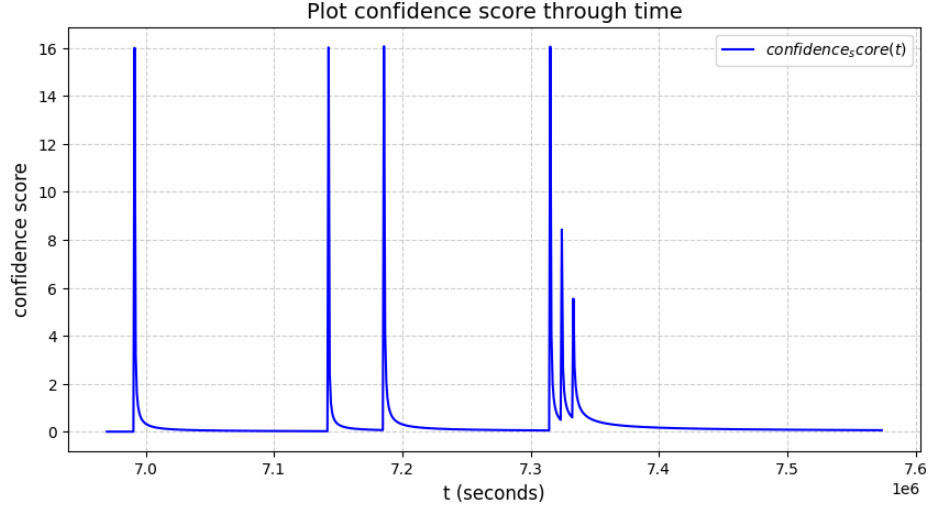


Figure 12: Confidence score visualization

Then apply the *negative exponential* to confidence\_score, we have coefficient to multiply to the recommendation score as **penalty** because it is in unit interval.

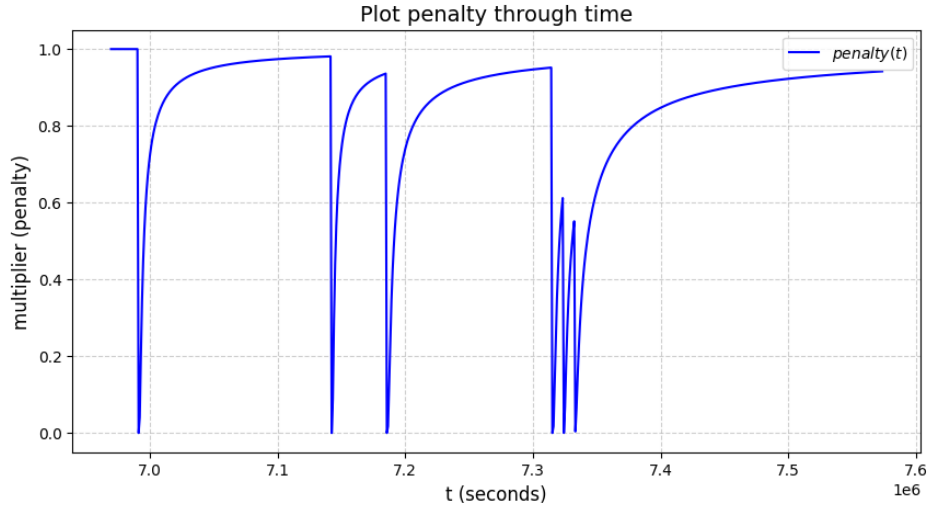


Figure 13: Penalty visualization

As depicted in the graph above, certain vertices introduce structural changes to the graph (the *peaks* of graph). These vertices correspond to instances immediately following the storage of *learningData* in the database. This pattern indicates that, after achieving a high performance score on a specific word, the likelihood of that word being suggested again in the near term is significantly reduced. This behavior aligns with the system's design to prioritize spaced repetition and mitigate redundant word suggestions.

## 4 Algorithm

### 4.1 Front-end

ViteJS overcomes the disadvantages of Webpack by coming up with a wholly different concept of development and build processes. ViteJS leverages native ES Modules within modern browsers to mean that there is no need for bundling during development. With this methodology, the time taken to fire up a development server and make changes is considerably reduced. While Webpack compiles the entire dependency graph, ViteJS serves files on-demand and transforms them when needed. This leads to a nearly-instantaneous server startup and HMR that is much faster than Webpack's implementation.

ViteJS integrates seamlessly with frameworks like React, Vue, and Svelte, commonly used in Chrome extension development. Its plugin-based architecture extends functionality and supports features like JSX and TypeScript out-of-the-box. That's the analysis for the front-end part of the application, here is the flowchart describing its algorithm:

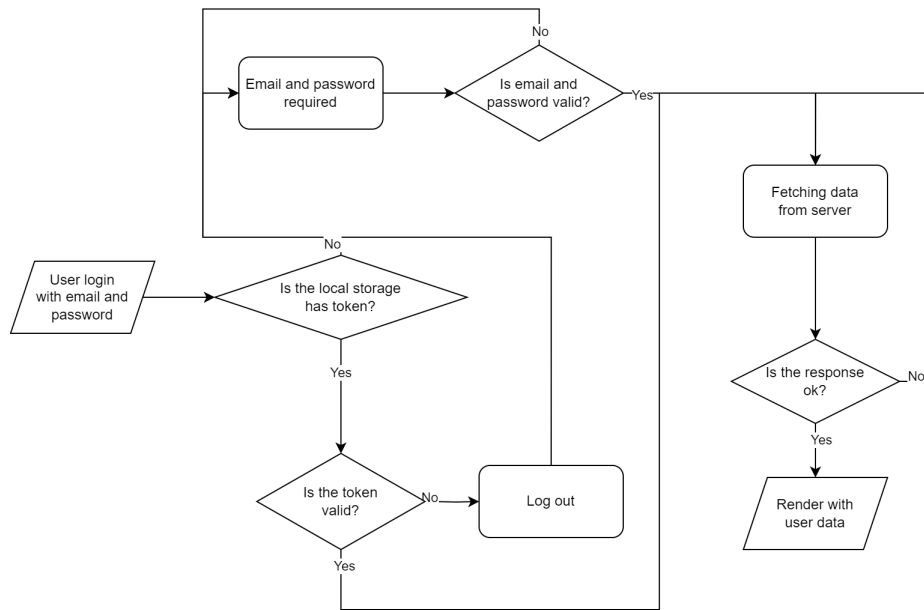


Figure 14: Front-end flowchart

### 4.2 Back-end

The backend architecture of the proposed English vocabulary flashcard recommendation system is designed to leverage the capabilities of FastAPI, known for its high performance and asynchronous processing, as the core backend framework. The front-end application communicates via RESTful APIs, sending requests to a NestJS middleware. NestJS acts as an intermediary layer, orchestrating the interaction between the client-side application and the backend.

Process API requests at FastAPI, thereby ensuring the proper routing of incoming requests and handling compute-intensive tasks related to data processing and execution of the recommendation model. This architecture helps in modularity and maintenance scalability with respect to the flow of data, and though it is not mandatory, it does facilitate future enhancements. RESTful API architecture was opted for easy integration, intercoupling, and modern web standard compliance. The core aim of this project is to provide personalized recommendations on flashcards to create a user-friendly learning process.

#### 4.2.1 Feature Vector Calculation for Bio and Interest

- Bio Feature Extraction

```

1 Input: bio (string containing personal and professional details)
2 Output: bio_emb (normalized feature vector)
3
4 1. Tokenize the bio using a tokenizer.
5 2. Compute the text embedding of the bio using a pre-trained model.
6 3. Squeeze the embedding to remove unnecessary dimensions.
7 4. Detach the embedding from the computation graph and move it to CPU.
8 5. Normalize the bio_emb by dividing it by its L2 norm.
9 6. Return the normalized bio_emb.

```

- Interest Feature Extraction (EMA-based)

```

1 Input: history (list of strings containing past internet searches),
2       alpha (float, EMA parameter), previous_emb (optional, previous
3       embedding for EMA)
4 Output: interest_emb (normalized feature vector for user interests)
5
6 1. Initialize interest_emb as a zero vector (1024 dimensions) if no
7    previous_emb is provided.
8 2. Loop through each entry in the history:
9    a. Concatenate relevant fields (e.g., title, description, keywords)
10   .
11   b. Skip if the concatenated string is too short (< 50 characters).
12   c. Update interest_emb using EMA:
13     interest_emb = (1 - alpha) * interest_emb + alpha *
14       text_embedding_of_entry.
15 3. Normalize the interest_emb by dividing it by (1 - alpha^(count+1)).
16 4. Return the normalized interest_emb.

```

#### 4.2.2 Word Similarity Computation

```

1 Input: WORD_EMBEDDINGS (matrix of word embeddings), WORD_MAPPING (mapping
2       of words to indices),
3       bio_emb, interest_emb
4 Output: sim_bio, sim_interest (similarity scores with each word)
5
6 1. Compute the dot product between WORD_EMBEDDINGS and bio_emb to get
7    sim_bio.
8 2. Compute the dot product between WORD_EMBEDDINGS and interest_emb to get
9    sim_interest.
10 3. Return sim_bio and sim_interest.

```

#### 4.2.3 Word Frequency Calculation

```

1 Input: corpus (text containing all words to analyze)
2 Output: word_count (dictionary of word frequencies), num_words (total
3       words in corpus)
4
5 1. Normalize the corpus by:
6   a. Converting to lowercase.
7   b. Replacing special characters with spaces.

```

```

7   c. Splitting into words.
8  2. Use a Counter object to calculate the frequency of each word.
9  3. Return the word_count dictionary and the total word count (length of
    the word list).

```

#### 4.2.4 Score Calculation and Question Generation

- Score Calculation

```

1  Input: sim_bio, sim_interest, word_count, num_words, weights (w_bio,
    w_freq, w_history)
2  Output: final_score (list of scores for each word)
3
4  1. Initialize an empty list for final scores.
5  2. Loop through each word in WORD_EMBEDDINGS:
6      a. Compute freq_bonus as (word frequency / total words) if the word
          exists in word_count; otherwise, set freq_bonus = 0.
7      b. Compute the total score for the word:
8          total_score = w_bio * sim_bio[idx] + w_history * sim_interest[
          idx] + w_freq * freq_bonus.
9      c. Append the total score to the final_score list.
10 3. Return final_score.

```

- Question Generation

```

1  Input: final_score, WORD_MAPPING, k (number of questions to generate)
2  Output: res (list of questions with words and their answers)
3
4  1. Sort the indices of final_score in descending order of scores.
5  2. Select the top k indices as idx1 and the next 3k indices as idx2 (
    shuffle idx2 for wrong answers).
6  3. Initialize empty lists for word, score, and definition using idx1.
7  4. Loop through each word in idx1:
8      a. Create a question dictionary with:
9          - word_id: the word index.
10         - question: the definition of the word.
11         - answers: 3 wrong words (from idx2) and the correct word (
            shuffled).
12         - correct_id: the index of the correct word in answers.
13      b. Append the question dictionary to res.
14 5. Return the res list.

```

#### 4.2.5 Combining All Steps

```

1  Input: history, bio, corpus, WORD_EMBEDDINGS, WORD_MAPPING, k (number of
    questions)
2  Output: res (list of questions)
3
4  1. Compute interest_emb using the history and compute_interest_emb().
5  2. Compute bio_emb using the bio and calc_text_embedding().
6  3. Compute sim_bio and sim_interest using the embeddings and similarity
    computation.

```



```

7 4. Compute word_count and num_words using the corpus and word frequency
   count.
8 5. Compute final_score using the similarity scores, word frequencies, and
   weights.
9 6. Generate k questions using the final_score and question generation
   logic.
10 7. Return the res list of questions.

```

## 5 Computational Thinking Application

The problem of recommending flashcard questions tailored to user preferences and behaviors involves significant computational thinking to ensure efficiency, accuracy, and personalization. It requires processing diverse data inputs, such as user-provided information, passive data (e.g., browser history), and interaction metrics, to generate relevant and diverse recommendations. Computational methods are essential for managing large datasets, identifying patterns, and implementing algorithms that balance constraints like diversity (high TTR) and performance. By leveraging *computational thinking*, the solution integrates data processing, pattern recognition, and optimization to deliver personalized, high-quality flashcard suggestions effectively.

<b>Problem Identification</b>	We need an English learning application that is more personalized.
<b>Pattern Recognition</b>	Users' personal information, such as career, hobbies, browsing history, and learning history, provides valuable insights for vocabulary recommendations.
<b>Abstraction</b>	This system can effectively work by extracting features from user-provided inputs, including translations, personal information, and browser history, to generate personalized vocabulary recommendations.

Table 5: Iteration 1

<b>Problem Identification</b>	What are the key requirements for the application?
<b>Pattern Recognition</b>	<ul style="list-style-type: none"> <li>• <i>Honesty</i>: Users must provide truthful information about their careers, interests, and learning preferences.</li> <li>• <i>Access</i>: Users need to grant permission to access their browser history.</li> <li>• <i>Usage</i>: The application should become the primary means of translation for users to collect relevant data.</li> </ul>
<b>Abstraction</b>	The key goal of this application is to deliver personalized vocabulary recommendations. Achieving this requires accurate and comprehensive data to best reflect the user's preferences and needs.

Table 6: Iteration 2

<b>Problem Identification</b>	How can we measure the effectiveness of the system?
<b>Pattern Recognition</b>	<ul style="list-style-type: none"> <li>• <i>Type-Token Ratio (TTR)</i>: Measures the diversity of vocabulary by evaluating the proportion of unique words in a given text or user history.</li> <li>• <i>Competence Formula</i>: A metric to quantify how well a user handles the recommended words, combining factors like learning success rate and familiarity with the words.</li> </ul>
<b>Abstraction</b>	The system needs to measure both the uniqueness of vocabulary (e.g., through TTR) and user competence (e.g., via formulas that track performance and learning progress). Recommendations should adapt to the user's evolving ability and familiarity with the language.

Table 7: Iteration 3

<b>Problem Identification</b>	How can we process user-provided information, translations, and browsing history for vocabulary recommendations?
<b>Decomposition</b>	<ul style="list-style-type: none"> <li>• Extract user-provided information such as career, hobbies, and interests.</li> <li>• Extract insights from user translations</li> <li>• Adapt to User’s interests.</li> </ul>
<b>Pattern Recognition</b>	Advancements in language models like BEIT3 enable the efficient extraction of meaningful embeddings from textual data.
<b>Abstraction</b>	The input can be framed as sentences describing the user’s career, hobbies, learning goals, and browsing topics. These inputs can be used to identify and recommend vocabulary that matches the user’s context and needs.

Table 8: Iteration 4

<b>Problem Identification</b>	How can we effectively extract user-provided information?
<b>Pattern Recognition</b>	<ul style="list-style-type: none"> <li>• User-provided information is typically in sentence form, such as "I am an engineer interested in robotics."</li> <li>• BEIT3 embeddings can capture semantic meaning effectively, making it a suitable model for analyzing this input.</li> </ul>
<b>Abstraction</b>	User-provided information can be directly transformed into feature-rich embeddings using BEIT3. These embeddings will guide the vocabulary recommendation system.

Table 9: Iteration 5

<b>Problem Identification</b>	How can we extract information from user translations?
<b>Pattern Recognition</b>	<ul style="list-style-type: none"> <li>• <i>Frequency of Words</i>: Words that appear more frequently in the user’s translations are likely more relevant to their learning process and context.</li> <li>• <i>TF (Term Frequency) Scores</i>: These scores can quantify the importance of specific words within translations.</li> </ul>
<b>Abstraction</b>	By analyzing the frequency and term frequency of words in user translations, the system can prioritize recommending words that are more relevant and meaningful to the user.

Table 10: Iteration 6

<b>Problem Identification</b>	How can we adapt to the user’s interests?
<b>Pattern Recognition</b>	<ul style="list-style-type: none"> <li>• We can utilize the Browsing history</li> <li>• Raw URLs alone do not provide enough meaningful context.</li> <li>• Extracting metadata (such as keywords, page titles, or content summaries) from URLs can provide valuable information about the user’s interests and browsing habits.</li> </ul>

Table 11: Iteration 7

## 6 Ethics Social

### 6.1 Privacy Problem

The collection and processing of user information such as age, gender, occupation, interests, browsing history, and translation history pose significant challenges in ensuring data privacy and maintaining user trust. Therefore we have measures for our application for user information as follows:

- **Data Minimization**: We *ONLY* collect the information necessary for the functionality of the application. For example, instead of storing detailed browsing histories, consider retaining only aggregated insights that meet the application’s requirements.
- **Explicit Consent**: We ensure users are informed about the data collected, its purpose, and how it will be used. Implement opt-in mechanisms that allow users to consent to specific data processing activities.
- **Consent**: Explicit user consent will be required before collecting or processing any data. Users will have the option to opt out of certain features without losing access to basic functionalities.

## 6.2 Security Problem

The backend of our application is built using NestJS, a framework renowned for its robust security features. NestJS provides a strong foundation for our application by offering built-in mechanisms for authentication, authorization, and data protection. This choice aligns with our commitment to ensuring the confidentiality and integrity of user data.

- Authentication is a cornerstone of application security, and NestJS provides seamless integration with Passport, a widely used authentication middleware for Node.js. Passport simplifies the implementation of various authentication strategies, such as local authentication, OAuth, and third-party providers.

By using Passport in NestJS, developers can implement both session-based and token-based authentication with minimal overhead, ensuring the authentication process is secure and maintainable.

- JSON Web Tokens (JWT) are an industry standard for securing RESTful applications. NestJS natively supports JWT via the `@nestjs/jwt` package, which works seamlessly with Passport. The JWT mechanism involves *Token Issuance*, *Token Validation*

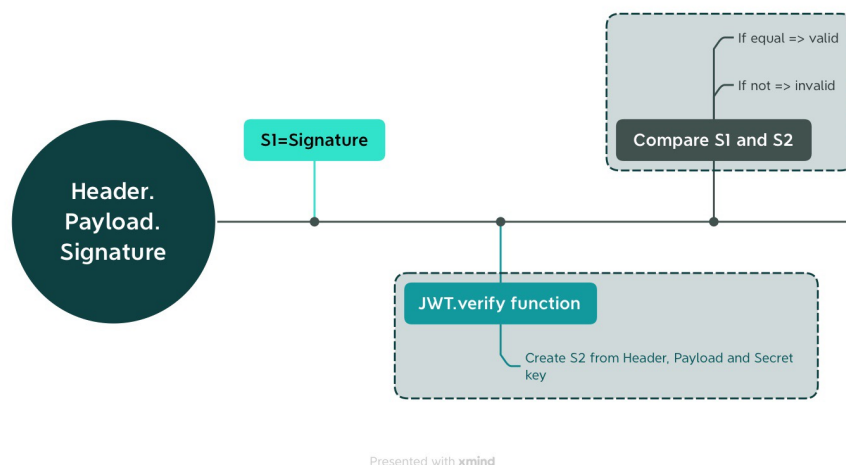


Figure 15: JWT verify token

- NestJS applications commonly utilize robust hashing algorithms like **bcrypt** to hash passwords before storing them in the database. These algorithms are designed specifically for password security and incorporate features such as salting and computational difficulty.

When comparing passwords, we only compare two hashed strings. Hashing is a one-way function, so if the server is attacked, the attacker will not know the user's password.

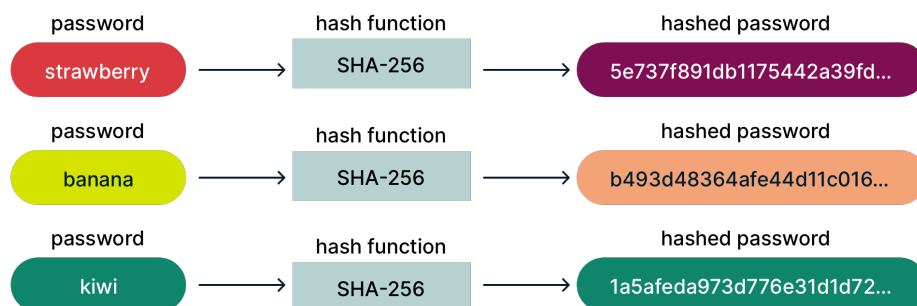


Figure 16: Hashing password

### 6.3 Copyright Problem

The application utilizes various resources, including dictionaries, embeddings, and translations, which may involve intellectual property and copyright concerns. To mitigate these issues:

- **Licensed Resources:** Ensure that all external resources (e.g., Oxford Dictionary, Google Translate) are used in compliance with their respective licenses. Proper attribution will be provided where required.
- **Fair Use Compliance:** For educational and non-commercial purposes, leverage resources under fair use guidelines. Only minimal necessary data will be utilized to achieve application objectives.
- **User-Generated Content:** Clearly define user rights for content they create or provide within the application, ensuring they retain ownership of their data.
- **Open-Source Models:** For models like BEIT3, adhere to the licensing terms provided by the creators to ensure no misuse of the technology.

Github repo:  <https://github.com/trungdangtapcode/Flashcard-Recommendation-Extension>