**Final Project**

**Snippet generation algorithms**

By Ziming Song

[zs2815@nyu.edu](mailto:zs2815@nyu.edu)

## **Abstract**

Millions of people access web search engines such as Google and Bing to find information relevant to their interests. During search, users visit a web search engine and use an interface to specify a query that best describes their information need. Upon query issuing, the engine's retrieval modules identify potentially relevant pages and present them to users in a ranked list based on their relevance to the query keywords. Currently, all major search engines display search results as a ranked list of URLs, accompanied by the returned pages’ titles and snippets (small text fragments that summarize the context of search keywords). These snippets provide a preview of the pages' content, aiding users in evaluating search results and making informed decisions about which pages to explore.

Thus far, the extraction of text snippets from the returned pages’ contents relies on statistical methods in order to determine which text fragments contain most of the query keywords due to their efficiency and simplicity. In this project, I implement four different snippets generation algorithms based on previous research and proposed a snippet generation algorithm. I also utilize the OpenAI API for extracting web content snippets. The project aims to enhance user decision-making by providing efficient insights into search results before clicking on a page. Towards this goal, I replicated the previous published methods and incorporate the latest OpenAI API. Then, I propose my own statistical method. Finally, I evaluate and compare query-related text fragments for usefulness and expressiveness. Overall, the project selects the most relevant and representative text nugget from each retrieved page, employing various algorithms to align with the query intention.

## **Introduction**

The advent of the internet has significantly enhanced access to information, bringing individuals into closer proximity to knowledge than ever before. Web search engines have emerged as the preeminent tools for acquiring pertinent information on a given subject. The appeal of search engines lies in their user-friendly interface, allowing individuals to articulate their queries in natural language and receive in response a list of URLs that point to pages, which relate to the information sought.

Retrieved results are systematically organized to reflect the importance or relevance of web pages to a given query. Despite the usability and approachability of these engines, individuals often find themselves overwhelmed by the sheer volume of information presented, as the results often consist of extensive lists of URLs. To help users locate the desired information, search engines augment retrieved URLs with text snippets, which are extracted either from the description meta-tag or from specific tags within the text.

A snippet is usually a set of contiguous text, typically in the size of a paragraph, which provides a preview of the content extracted from a retrieved webpage in order to assist users in determining whether the page suits their information interest or not. Depending on their decisions, users might access the pages’ contents simply by clicking on initially retrieved URLs or ignore them and explore the next bunch of results. Contemporary approaches to snippet selection often involve the extraction of text passages with keyword similarity to the query, employing statistical methods. For example, Google's snippet extraction algorithm[1] utilizes a sliding window of 15 terms (or 100 characters) over the retrieved document to generate text fragments containing the queried keywords. The first two passages within the text are merged together to produce the final snippet. Generally, these statistical methods demonstrate commendable efficiency.

Evidently, if we could equip search engines with a powerful mechanism that generates self-descriptive and document expressive text snippets, a lot of time would be saved for individuals seeking information online. That is, if we provide users with that piece of text from a page that is the most relevant to their search intention and which is also the most representative extract of the page, we may help them decide the page is of interest to them before they actually click on it.

In this project, I utilize five different statistical and query-dependent snippet selection algorithms and propose a snippet selection algorithm called *weight selection*. I also use OpenAI API to get NLP based snippets. BM25 Selection, Vector-Space Model Selection, Weight Selection are score-based. The seven different methods are as follows:

1. Linear Match
2. Prefix Match
3. Keyword Match
4. BM25 Selection
5. Vector-Space Model Selection
6. Weight Selection
7. OpenAI API

Then I applying these seven snippet selection techniques to a number of searches and compare them on different dimension. In brief, the contributions of this project are as follows:

* I replicate and implement five different statistical snippet selection methods. Additionally, I utilize the OpenAI API, a prominent NLP tool, to extract snippets from provided content.
* I propose a statistical measure for quantifying the snippet’s closeness to the query intention (usefulness). In my work, a useful snippet is the text fragment in a retrieved page that exhibits the greatest terminological overlap to the query keywords.
* I use metrics that estimates the importance and the representation ratio of a snippet with respect to the contents of the entire pages from which this was extracted, aiming at identifying the query focus in the search results.
* I compare seven snippet selection techniques using metrics based on users’ feedback.

The remainder of the article is organized as follows. Discussion begins with a detailed description of seven different approach in snippets’ selection. Then in Section 3, I experimentally evaluate the effectiveness of different snippet generation algorithm in focusing retrieval on the query and I discuss obtained results. In Section, I try to explain the reasons behind the result and discuss future work.

## **Algorithm**

It is widely acknowledged that web users decide on which results to click based on limited information. Typically, information seekers rely on the retrieved page’s title, URL and text snippet that contains their search keywords to infer whether the page is of interest to their search pursuit or not.

Despite the fact that web page titles are manually specified and generally indicative of the page's contents, the automatically extracted text snippets are not consistently descriptive of the page's thematic content. Given that snippet extraction is solely dependent on the distribution of query keywords within these elements, it becomes evident that the selected snippets may convey incomplete or even misleading information about a page's content, potentially leading users to click on pages only marginally relevant to their search pursuits.

Obviously, decisions based on little information are susceptible to be bad decisions. A bad decision is encountered when the user clicks on a result misguided by a text snippet, which is of little relevance to the linked page’s contents. Similarly, a bad decision might be when the user decides not to click on a good result, simply because a poor or seemingly unrelated text snippet.

In this section, I present seven different approaches towards the automatic extraction of query-relevant and document-expressive snippets. The aim is to assist web information seekers make informed decisions, about whether to click on a retrieved result or not.

All the following algorithms for snippets generation operates exclusively on a query-dependent paradigm, which implies that the generation of snippets is contingent upon the inherent characteristics of the given query. Suppose the list of individual words in a given query *q* as *word\_list*, the web content of each result is *wc*.

### **Linear Match**

Linear match algorithm based on [1],is the basic snippet selection algorithm. As its name implies, it finds the snippets according to terms in *word\_list*. The algorithm can be described by the following principal steps.

(L1) For a given *word\_list*, find position of first two terms *t1*, *t2* which exactly match a word in *word\_list*.

(L2) Utilize a sliding window of 15 terms (about 100 characters) over *wc* to generate text fragments containing the *t1*, *t2*.

(L3) Merge the first two passages within the text together to produce the final snippet.

### **Prefix Match**

In English and many other languages, verbs have different possessive forms, such as present continuous tense, past tense. For instance, the verbs "run", "runs", and "running" share the common prefix "run". Likewise, nouns and verbs conveying identical meanings may share identical prefixes; e.g., the terms "decompose" and "decomposition" both incorporate the prefix "decompos".

Prefix matching involves the identification and incorporation of common prefixes in user queries, allowing for a more contextually aligned snippet response. Information seekers tend to type verb base form, noun singular in a query (usually the same as prefix of different forms). By recognizing and leveraging the shared prefixes in queries, the snippet generation process becomes more adept at capturing the user's intent and delivering concise, targeted information. Prefix Match algorithm can be described by the following principal steps.

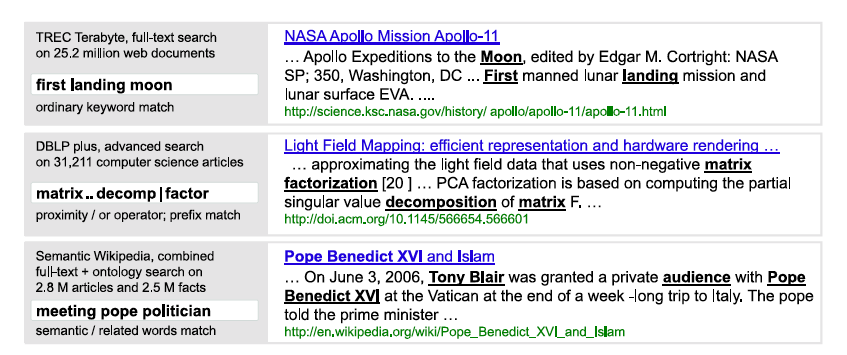


Figure 1 Two examples of query-dependent snippets. The first example using Linear Match, shows a snippet (with two parts) for an ordinary keyword query; this can easily be produced by the document-based approach. The second example using Prefix Match shows a snippet for a combined proximity/OR query.[2]

(P1) For a given *word\_list*, find position of first two words *w1*, *w2*, which a word in *word\_list* is the prefix of *w1*, *w2*.

(P2) Utilize a sliding window of 15 terms (about 100 characters) over *wc* to generate text fragments containing the *w1*, *w2*.

(P3) Merge the first two passages within the text together to produce the final snippet.

### **Keyword Match**

Now we focus on query itself. Not all words in *word\_list* have the same significant. We call words demonstrating greater relevance to the query as keywords. It is evident that the text surrounding a keyword typically holds greater importance. Consequently, the selection of these contextual segments as snippets generally proves to be more relate to the information sought. Keyword Match algorithm can be described by the following principal steps.

(K0) For a given *word\_list* whichcontain *n* words, calculate the frequency of each word *ti* in document *dj*.

(K1) For a given *word\_list* whichcontain *n* words, weight each word. Formally, the weight associated with the term *ti* in document *dj* is given by

where is the frequency of *ti* in *dj*, *N* is the total number of documents considered (in my approach N = 20) and *dfi* is the total number of documents that contain *ti*.

Having weighted every word in the top N retrieved pages, sort them using heap and I retain *m* most highly weighted terms, as "keyword". Formally, *m* associated with the minimum keyword number (= 2) and proportion of keywords among all words (= 25%), is given by

The keywords are stored in *keyword\_list*.

(K2) For a given *keyword\_list*, find position of first two terms *k1*, *k2* which exactly match a word in key*word\_list*.

(K3) Utilize a sliding window of 15 terms (about 100 characters) over *wc* to generate text fragments containing the *k1*, *k2*.

(K4) Merge the first two passages within the text together to produce the final snippet.

### **BM25 Selection**

The above methods introduced in section 2.1-2.3 all select the first two passages with the desired words. Despite their efficiency, these methods overlook the fact that the best snippets not always occur at the beginning at a web page. Consequently, an algorithm has been introduced to address this limitation, wherein a score-based snippet generation approach is employed. This algorithm selects snippet based-on scores of passages after a comprehensive traversal of the entire web content, thereby enhancing the usefulness and expressiveness of snippet identification.

BM25, an acronym for "Best Matching 25," is a ranking function commonly used in information retrieval and text mining. BM25 operates on the probabilistic information retrieval model and excels in handling various document lengths and query structures. This ranking function evaluates the relevance of a document to a given query by considering term frequencies, document lengths, and term-specific parameters. Unlike traditional term frequency-inverse document frequency (TF-IDF) approaches, BM25 introduces a saturation term to prevent term frequencies from disproportionately influencing the ranking. This feature makes BM25 particularly effective in scenarios where the length of documents varies significantly. The BM25 score-based algorithm can be described by the following principal steps.

(B0) For a given *word\_list* whichcontain *n* words, calculate the frequency of each word *ti* in document *dj*; calculate the position list *p\_l* of of each word *ti* in document *dj.* *p\_l* is a hash table, the key is term in *word\_list*, the value is a list of this term’s position in document in ascending order.

(B1) For a given *word\_list* whichcontain *n* words, calculate BM25 of each word. Formally, the part of BM25 associated with the term *ti* in document *dj* is given by

where is number of occurrences of term *ti* in document *dj*, is the length of document *dj,*  is the average length of documents in the collection*,* is the total number of documents, and is total number of occurrences of t in the collection.

(B2) Group the position in *p\_l* in order. The group result *g\_r* contains several lists; each list contains position of different term in *p\_l* in ascending order. The difference between each position in a list is less than snippets range *sr* (*sr* = 50) characters.

(B3) Calculate each list’s score in *g\_r*. Formally, the score of each list is given by

where r is the length of a list in *g\_r*. Then select the top k (k = 2) lists using heap with highest score.

(B4) Extract passage around the position in top k lists, merge the passages and within the text together to produce the final snippet, making sure the snippets in reasonable length.

### **Vector-Space Model Selection**

This algorithm is also score-based and almost the same as BM25 Selection. The only difference between the two methods is score function.

The Vector-Space Model (VSM) is a mathematical representation used in information retrieval and natural language processing to analyze and represent textual data. Introduced as a framework for document retrieval, VSM represents documents as vectors in a high-dimensional space, where each dimension corresponds to a unique term or word. In the Vector-Space Model, the similarity between documents is quantified by measuring the cosine of the angle between their respective vectors. This enables the model to identify semantic relationships and similarities among documents based on the distribution of terms within them.Its flexibility and simplicity make it a foundational concept in the field of information retrieval and contribute significantly to the snippet generation.

The Vector-Space Model Selection algorithm can be described by the following principal steps.

(V0) Same as B0

(V1) For a given *word\_list* whichcontain *n* words, calculate cosine score of each word. Formally, the part of cosine score associated with the term *ti* in document *dj* is given by

where is number of occurrences of term *ti* in document *dj*, is the length of document *dj,* is the total number of documents, and is total number of occurrences of t in the collection.

(V2) (V3) (V4) Same as B2, B3, B4.

### **Weight Selection**

I propose this algorithm combining Keyword Match and score-based snippet generation method. Just like Keyword Match, Weight Selection selects *m* keywords, and then calculate the weight of each key words. As for scoring process, *p\_l* only contains position of keywords rather than all words in word list. This algorithm combines the advantages of Keyword Match algorithm and score-based snippet generation method, making snippets contain more relevant query words in the whole web content.

The Weight Selection algorithm can be described by the following principal steps.

(W0) For a given *word\_list* whichcontain *n* words, calculate the frequency of each word *ti* in document *dj*; calculate the position list *p\_l* of of each word *ti* in document *dj.* *p\_l* is a hash table, the key is term in *word\_list*, the value is a list of this term’s position in document in ascending order.

(W1) For a given *word\_list* whichcontain *n* words, weight each word. Formally, the weight associated with the term *ti* in document *dj* is given by

where is the frequency of *ti* in *dj*, *N* is the total number of documents considered (in my approach N = 20) and *dfi* is the total number of documents that contain *ti*.

Having weighted every word in the top N retrieved pages, sort them using heap and I retain *m* most highly weighted terms, as "keyword". Formally, *m* associated with the minimum keyword number (= 2) and proportion of keywords among all words (= 25%), is given by

The keywords and weight are stored in *keyword\_hashtable*. The key of the hash table is keyword, the value is its corresponding weight.

(W2) Group the position in *p\_l* in order. The group result *g\_r* contains several lists; each list contains position of different keywords in *p\_l* in ascending order. The difference between each position in a list is less than snippets range *sr* (*sr* = 50) characters.

(W3) Calculate each list’s score in *g\_r*. Formally, the score of each list is given by

where r is the length of a list in *g\_r*. Then select the top k (k = 2) lists using heap with highest score.

(W4) Extract passage around the position in top k lists, merge the passages and within the text together to produce the final snippet, making sure the snippets in reasonable length.

### **OpenAI API**

The above algorithms introduced in section 2.1-2.6 are all statistical. Even though statistical methods perform well on efficiency, statistically generated snippets are rough indicators of the query terms co-occurring context but, they lack coherence and do not communicate anything about the semantics of the text from which these are extracted.

The OpenAI API is a powerful tool that provides programmatic access to various language models developed by OpenAI, including GPT models such as GPT-3. This API enables developers to integrate cutting-edge natural language processing capabilities into their applications, products, or services. By using the OpenAI API, developers can send requests to the model and receive generated text as a response. This allows for a wide range of applications, from natural language understanding and question-answering systems to content generation and creative writing assistance.

I use OpenAI API to generate query-dependent snippet. First, I generate an API key, which is used for authentication. Then I use client to get message of snippet generation request using model ‘gpt-3.5-turbo-1106’. The OpenAI API offers a versatile and efficient way to leverage advanced language models for snippet generation tasks[[1]](#footnote-1).

## **Experimental**

In the previous sections, several algorithms are proposed to select snippets from the contents of the query matching pages. In this section, I experimentally evaluate and compare the effectiveness of these algorithms. In Section 3.1, I introduce the programming details about my algorithms’ implementation. In Section 3.2, I describe the dataset that I used for evaluation. In Section 3.3, I discuss how I evaluated the effectiveness of our model. In Section 3.4, I represent obtained results and compared results of different algorithm.

### **Programming details**

The final project code based on Assignment #2 and #3, which helps to extract top K results of a specific query and find document content. Based on previous work, the majority of algorithms are implemented using C++ with the exception of the OpenAI API algorithm. It is because API using C++ are not provided by OpenAI company now. So, OpenAI API algorithm is implemented using Python.

For specific codes implemented in C++, see file *Snippets.cpp*, *Snippets.h*. For OpenAI API algorithm, see file *chatgpt.py*[[2]](#footnote-2).

For calculating recall and precision, I use Python, see file *genExpResult.py*.

### **Experimental dataset**

In this project, I use *msmarco-docs.trec.gz*[[3]](#footnote-3) as dataset, which is the same as dataset I used in Assignment #2 and #3. This dataset contains 8,501,800,960 bytes.

In picking experimental queries, I randomly select 20 different queries which contain 1-5 words, and then use disjunctive query to get top K (K=20) results.

The top K results of each query are saved a *.txt* file in *result* folder to save time spending on lexical structure and document table loading and searching result. Program can read these .txt file and generate snippet directly from the document without finding the top K result again and again.

### **Evaluation measures**

For evaluation, I use these seven different snippet generation algorithms on 20 different queries that each query has top 20 results. In another word, each of the algorithm is applied on 400 documents to generate snippet of a specific query. Sometimes OpenAI API may not have result because its limited token. These cases are not considered in the evaluation.

Then, I evaluated the snippets that each algorithm delivered by comparing them to the snippets that participants manually determined for the same set of pages and queries. I invited 20 participants to select snippets from documents manually. Each document has no less than 5 participants. Note that the manually selected snippets considered in evaluation are those selected by the majority (i.e., 3 out of 5) of the participants who examined them. I started evaluation by estimating the recall and the precision of the snippets generated by different algorithms.

The recall[3] relies on the overlapping elements between the set of manually selected snippets for each of the query matching pages examined and the set of system-selected snippets for the same set of pages and queries. Note that both sets are of size s, where s is the total number of snippets of an algorithm examined. Recall is defined as the fraction of the manually selected snippets that are also extracted by the algorithm, formally given by

where the nominator is the size of the intersection of the two sets of snippets and s is the total number of snippets considered.

To estimate recall for the different snippet selection metrics, participants are asked to manually select snippets from each document according to the passage’s usefulness, coherence and expressiveness. OpenAPI usually extract result not directly from the text, it is more like an answer of the query based on document or conclusion of the document. So, I do not calculate its recall.

I also evaluated the precision of the snippets selected of each algorithm for each of the examined pages and queries, using the same methodology that we used for estimating recall. In particular, we wanted to examine whether algorithm generated snippets can help our subjects infer the correlation that their source pages have to the query intention.

More specifically, to evaluate precision, participants are asked to examine the snippets extracted using each of the algorithm and assess the following: (i) whether the snippet is helpful in deciding whether to click on the respective document, (ii) whether the correlation between the query intention and the retrieved document can be inferred by the displayed snippet. In our evaluation, we consider a snippet to be precise if both criteria are met. Therefore, precision is defined as the ratio of precise snippets over the total number of automatically selected snippets (s), given by

As in the case of recall measurement, each snippet was examined by five different subjects and a snippet is considered to be precise if at least three of the participants marked the snippet as being precise.

### **Experimental results**

#### **Recall**

Table 1 lists the performance of six different algorithms with respect to the measurement of recall. In particular, the table reports the recall values that each of our snippet selection metrics demonstrates as well the number of snippets used in recall’s calculation.

In overall, results indicate that score-based algorithms perform better on recall, especially that using BM25 as score function.

|  |  |
| --- | --- |
| Algorithm | recall |
| Linear Match | 0.54 |
| Prefix Match | 0.525 |
| Keyword Match | 0.539 |
| BM25 Selection | 0.785 |
| Vector-Space Model Selection | 0.775 |
| Weight Selection | 0.606 |

Table 1 recall of algorithms

#### **Precision**

However, recall is not a sufficient indicator of each algorithm’s accuracy in selecting precise snippets, i.e. snippets that are informative of their correlation to the query and which assist users make informed clicking decisions. To capture each algorithm’s accuracy in extracting precise snippets, subjects are used to assess the snippets selected by our model’s metric from the pages retrieved for experimental queries. Results of precision are shown in Table 2.

In overall, results indicate that score-based algorithms perform better on precision, especially that using BM25 as score function. The chatgpt 3.5 model does not perform well on the task of snippet generation as expected.

|  |  |
| --- | --- |
| Algorithm | precision |
| Linear Match | 0.635 |
| Prefix Match | 0.625 |
| Keyword Match | 0.632 |
| BM25 Selection | 0.810 |
| Vector-Space Model Selection | 0.798 |
| Weight Selection | 0.716 |
| OpenAI API | 0.752 |

Table 2 precision of algorithms

#### **Examples**

Here lists some queries and its corresponding snippet using different algorithm.

Query: top 10 bar in USA

Snippet:

Linear Match: ... be performed with any grip, in recent years some have used the term to ...with the chin brought over top of a bar, was used in the ...

Prefix Match: ... search Not to be confused with bars.pull-up techniques A pull-up is ...upper-body compound pulling exercise. Although it can be performed with ...

Keyword Match: ... up with the chin brought over top of a bar, was used in the ...with or passes over the top of the bar. A chest-up"...

BM25 Selection: ...bar, face-up, and grasps the bar with extended arms. The exercise is performed by pulling the chest up...bar or go over the bar, with a supinated palms-facing grip. Variations of pull ups, beyond being named...

Vector-Space Model Selection: ...in 6 hours". The most pull ups in 12 hours is 3,515 by Andrew Shapiro (USA), in Great Falls, Virginia, USA, on 14 May 2016.^ "Most pull ups in 12 hours". The most pull ups in...in 2016. [34]Most in minute with 40 pounds: 31 by Ron Cooper (USA) in Marblehead, Massachusetts, USA, on...

Weight Selection: ...10. Archived from the original on 2016-03-12. 8. PULL-UP. Often called upright rowing motion. Stand close...10 May 2002. Archived from the original (PDF) on 22 May 2011. a command will not mandate that Marines...

OpenAI API: As of January 2015 the most repetitions within a given time period: [38]3 minutes: 100 by Ngo Xuan Chuyen (VIE) in 1988 during "Strongest Soldier in Vietnam" contestalternativerecords.co.uk [ edit]

Query: hell heaven god Buddha

Snippet:

Linear Match: ... resolutely denied in Buddhism. Gautama Buddha did not endorse belief in a creator deity, ...wiser than us. The Buddha is often portrayed as a teacher of the ...

Prefix Match: ... in Greek mythology, the goddess Hera often became enraged when her husband, ...denied in Buddhism. Gautama Buddha did not endorse belief in a creator deity,...

Keyword Match: ... the belief in a creator god issara-nimmana-vada) is frequently mentioned ... grim, and painful hell realms to the most sublime, refined, ...

BM25 Selection: ...Buddha as "teacher of devas," and shows all beings how to work for Nibbana.^ Bhikku, Thanissaro (1997...heaven or hell. Beings are born into a particular realm according to both their past kamma and their...

Vector-Space Model Selection: ...Buddha for some time. The sutta records a long audience he had with the Blessed One which culminated...Buddha as "teacher of devas," and shows all beings how to work for Nibbana.^ Bhikku, Thanissaro (1997...

Weight Selection: ...god (issara-nimmana-vada) is frequently mentioned and rejected, along with other causes wrongly adduced...god showed his admiration and reverence for the Exalted One. ", "A discourse called Sakka's Questions...

OpenAI API: The concept of divine retribution is resolutely denied in Buddhism. Gautama Buddha did not endorse belief in a creator deity, and stated that questions on the origin of the world are worthless.

## **Discussion**

The results obtained from both recall and precision metrics unequivocally indicate that the BM25 Selection algorithm stands out as the most effective snippet generation algorithm among the seven algorithms employed in this study, based on the specific dataset utilized. . Generally speaking, it is discerned that score-based algorithms exhibit superior performance, ostensibly attributed to their capacity to select snippets from entire documents. Besides, in some case those algorithms who generate snippet from the beginning (Linear Match, Prefix Match, Keyword Match) of the document perform better. This phenomenon is posited to be a consequence of documents typically give pivotal or concluding information at the beginning, a characteristic perceived as valuable by our study participants.

An additional aspect warranting attention is the comparatively suboptimal performance exhibited by the OpenAI API in snippet generation. Notably, the API is constrained in generating snippets for only 290 out of 400 documents, ostensibly attributed to text limitations. Even when a snippet is generated, it frequently diverges from the document's context and appears more inclined toward addressing the query rather than faithfully representing the document's content. Moreover, instances are noted where the API extracts and consolidates information from the document, departing from using the original passage. These factors collectively contribute to the subpar performance observed with the OpenAI API in the context of snippet generation.

## **Reference**

1. Google Patent 2003, Detecting query-specific duplicate documents, US Patent No. 6615209.
2. Hannah Bast and Marjan Celikik. 2014. Efficient Index-Based Snippet Generation. ACM Trans. Inf. Syst. 32, 2, Article 6 (April 2014), 24 pages. <https://doi.org/10.1145/2590972>
3. Varlamis, Iraklis & Stamou, Sofia. (2009). Semantically driven snippet selection for supporting focused Web searches. Data & Knowledge Engineering. 68. 261-277. 10.1016/j.datak.2008.10.002.

1. In experiment, the snippet generated by openAI API is more like a conclusion/answer of a query rather than snippet from a text. [↑](#footnote-ref-1)
2. Due to the limitation OpenAI provided, the content of a message’s token should less than 16,385, the number of requests in 1 minute should less than 3, the number of request in 1 day should less than 200. [↑](#footnote-ref-2)
3. https://microsoft.github.io/msmarco/TREC-Deep-Learning-2020 [↑](#footnote-ref-3)