Extractive-Abstractive Hybrid Approach to Snippet Generation

Mehran Ali Banka
Graduate Student, Computer Science
New York University
6 MetroTech Center, Brooklyn, NY 11201
mab10251@nyu.edu

Abstract—**This paper explores the combination of Extractive and Abstractive Summarization Algorithms for use in search engines. In Information Retrieval, this problem is more formally referred to as Query-based Summarization. Traditional work in this area was focused on extractive summarization, which picks out the top-K most relevant sentences from the original document, which may or may not be based on the context of the query. Various state-of-the-art algorithms have been implemented for this purpose, and some of them will be discussed in this paper. Extractive summarization ultimately suffers from many drawbacks as it is dependent on the document completely and picking K sentences often creates a disjointed summary, or a summary that lacks context. The recent surge of transformer-based LLM models solve the problem of Abstractive summarization, and these models are becoming increasingly good at generating new content and creating cohesive summaries. In this paper, I try to use a graph-based extractive query-based summarization algorithm to generate a candidate paragraph, which then gets fed into various off-the-shelf LLM's for snippet generation.**

I. INTRODUCTION

Human computer interaction has taken off in the last 15 years and has significantly affected the field of Natural Language Processing (NLP).The proliferation of information on the internet has led to palpable challenges in text summarization for various real-world applications. The rapid dissemination of information has made it increasingly difficult for users to manually sift through large volumes of text to identify pertinent details. Text summarization can be quite effective in solving this problem.

There are quite a few ways to divide the problem of automatic text summarization. Extractive and Abstractive summarization are two major categories. In Extractive summarization, the top-K most pertinent sentences from a document are chosen and retrieved for the user. Thus, the output that the user sees is precisely present in the original document. In contrast, transformer based models are used for Abstractive summarization. These models are much more complex, as they need the system to do various additional tasks like sentence compression, paraphrasing, and reformulation to make the summary look more human-like. A good abstractive summary is not only expected to cover the main ideas of the document but also be linguistically fluent.

Summarization can also be categorized as single document based and multi-document based. Snippets produced by search engines are generally single document based. Some answers that search engines provide directly on their main page, such as a direct answer to "When was Lincoln born?" might be multi-document based.

Lastly, summarization can either be generic or topic-guided. In this paper, by topic-guided, I mean query based or query focused (QFDS) .Generic summarization gives an overview of the main ideas of the input document to the user. Query focused has the context of the query associated with it, and is generally a more complex problem. Traditionally, extractive generic summarization techniques received relatively more attention that query focused ones. But there are various state-of-the art extractive algorithms for QFDS. These can be broadly divided into three areas - Machine Learning based, Graph based and Clustering based.

While Extractive summarization can do a pretty good job at picking the top-K most relevant sentences from a given document, the summaries produced often look disjointed and are gramatically incorrect. This is a problem that abstractive summarizers handle very well. If the document is long enough, we could use extractive summarization to generate a "candidate paragraph", which can be then fed to some off-the-shelf transformer based models for abstrative summarization. The length of the output can also be controlled, and for the case of snippet generation, it can be kept to a couple of sentences. This technique could potentially combine the best of both worlds and is also much more computationally efficient than training a dedicated model for the problem of query based snippet generation. This is because it makes the job of an abstractive summarizer very easy, as it doesn't have to focus on the query at all, and also has to summarize just a paragraph into a few lines, rather than the whole document.

II. LITERATURE SURVEY

A. Background

Many different approaches have been studied for extractive QFDS, such as graph-based, machine-learning based, clustering based etc [1]. In this section, I will briefly describe these approaches, and some of the work done in those areas.

Concept-Based Methods: Concept-based methods in text summarization involve extracting concepts from a given text using external knowledge bases such as WordNet, HowNet, and Wikipedia. Unlike traditional word-based approaches, these methods assess sentence importance based on retrieved concepts. The process includes retrieving concepts from an external knowledge base, constructing a conceptual vector or graph model to illustrate the relationship between concepts and sentences, and employing a ranking algorithm to score the sentences (Moratanch Chitrakala, 2017). [2]

Topic-Based Methods: Topic-based methods in text summarization focus on identifying the main subject or topic of a document. Common approaches for topic representation include Term Frequency, Term Frequency-Inverse Document Frequency (TF-IDF), lexical chains, and topic word approaches. The process of a topic-based extractive summarizer involves converting the input text into an intermediate representation that captures the discussed topics and assigning importance scores to sentences based on this representation (Nenkova McKeown, 2012).[3]

Clustering-Based Methods: In multi-document extractive summarization, the focus is on identifying central sentences within a cluster to cover vital information on the cluster's main subject. Sentence centrality is determined by the centrality of words, often measured through the centroid of the document cluster in a vector space, consisting of words with TF-IDF scores above a set threshold (Erkan Radev, 2004) [4]. Centroid-based summarizers score sentences by creating a representative centroid through TF-IDF representations and considering sentences as central if they contain more words from the cluster centroid, reflecting their proximity (Erkan Radev, 2004; Mehta Majumder, 2018) [6]. Clustering-based summarization, addressing relevance and redundancy, involves clustering input sentences using algorithms. Clusters are ranked based on the prevalence of important words, and representative sentences are selected from these clusters for the summary (Nazari Mahdavi, 2019) [5]. This approach combines both importance and contextual relevance, as demonstrated in the works of Erkan Radev (2004), Mehta Majumder (2018), and Nazari Mahdavi (2019).

Machine Learning-Based Methods: Ouyang et al.(2011) [7] proposed a method for document summarization based on user queries, utilizing regression models. The method employs seven features for scoring, including three query-dependent features (word matching, semantic matching, named entity matching) and four query-independent features (word TF-IDF, stop-word penalty, sentence position). The scoring process involves assessing the overlap between the user's query and document sentences, expanding the query semantically, considering common named entities, and using TF-IDF to get word importance. A composite function based on the Support Vector Regression (SVR) approach is then applied to rank sentences and calculate their scores (Basak et al., 2007 [8]). SVR was one of the approaches I considered but could not implement due to unavailability of a large pertinent training set. DUC datasets would have been appropriate here, but they are hard to get. I will describe this approach in a bit more detail later.

Graph-Based Methods: Graph-Based Methods in text summarization utilize sentence-based graphs to represent a document or cluster of documents. Prominent examples include LexRank (Erkan Radev, 2004 [4]) and TextRank (Mihalcea Tarau) [9]. Erkan and Radev explore how random walks on such graphs aid in summarization. LexRank's scoring process involves representing document sentences with an undirected graph, where each node is a sentence and edges' weights reflect semantic similarity using cosine similarity. A ranking algorithm is then employed to determine sentence importance, with the sentences ranked based on LexRank scores, akin to the PageRank algorithm but with an undirected graph (Moratanch Chitrakala, 2017; Brin Page, 1998) [10], [2]. Just like PageRank introduces random jumps to converge, LexRank also uses random jumps and the resulting Markov chain formulation is almost exactly like the one in the PageRank paper.

For the purpose of Extractive summarization, I selected a graph based method "QSLK" by Abdi & Idris, 2015 [11] which improves upon LexRank in many ways. This method considers not only sentence to sentence similarity (S2S), but also sentence to query similarity(S2Q). It uses sentence expansion using a knowledge base WordNet, and computes not just the semantic similarity but also syntactic similarity. These weights for both can be adjusted, as can be the weights assigned to S2S and S2Q matrices.

III. Extractive Model Built

In this section, I will describe the the QSLK (Abdi & Idris, 2015 [11]) model I built. I initially shortlisted the SVR model (Basak et al., 2007 [8]), and the Graph based QSLK (Abdi & Idris, 2015 [11]) model. In addition to these, many other algorithms were explored such as LexRank, Topical-N, CTMSUM, etc [4], [12], [13]. SVR needs a lot of training data, and computational power. DUC 2005 and 2006 was an appropriate data set for this model, and that's what was used in the original paper. These datasets are not easy to get, so I ended up not implementing this method. QSLK does not have a dependency on a lot of training data, and in theory can perform better than SVR due to it's many enhancements over LexRank. For this paper, I implemented the QSLK algorithm in Python and used that to generate candidate paragraphs that can be fed to Abstractive Summarizers. I will describe QSLK implementation in detail.

A. Query-Based Summarization using Linguistic Knowledge (QSLK)

From a graph perspective, the problem of finding a relevant summary or snippet from a document, given a query is not that different from the basic search problem itself i.e., finding relevant documents given a query. Brin & Page, 1998 [10], introduced PageRank for this, which models this problem into finding a stationary distribution of a markov chain. The formulation is:

$$p(u) = \frac{d}{N} + (1-d) \sum_{v \in \mathrm{adj}[u]} \frac{p(v)}{\deg(v)} \tag{1}$$

This can be written in matrix notation as:

$$p = [dU + (1-d)B^T]p \tag{2}$$

Where d is the damping factor, typically chosen between 0.1 and 0.2, and U is a square matrix with all elements equal to 1/N. By taking some transition probability away from the actual nodes and giving them random jump probabilities in return, we ensure that the markov chain is irreducible and aperiodic. The resulting vector "p" is the stationary distribution of the markov chain.

Erkan & Radev, 2004 [4], introduced LexRank which follows a very similar approach to PageRank, for the purpose of document summarization. In this model, a connectivity matrix based on intra-sentence cosine similarity is used as the adjacency matrix of the graph representation of sentences. They discuss several methods to compute centrality using the similarity graph. Their results show that degree-based methods (including LexRank) outperform both centroid-based methods and other systems participating in DUC in most of the cases.

LexRank computes the edge weights based on cosine similarity of the tf-idf vectors. Most classic extractive algorithms focus on semantic similarity between the sentences. Semantic similarity between sentences is often measured by comparing the meaning or context of two sentences. One common approach is to use vector representations of sentences in a high-dimensional space. The problem with relying on just semantic similarity is that it suffers from context sensitivity and word sense disambiguation. Words can have different meanings in different contexts. For example, "bank" can refer to a river bank as well as a financial institution. Furthermore, "John called Adam" and "Adam called John" are semantically similar but have different meanings.

Syntactic similarity pertains to the assessment of structural resemblance or similarity in grammatical constructions between two linguistic expressions, such as sentences or phrases. It involves analyzing the arrangement and relationships of words within these expressions, typically through the use of syntactic structures like parse trees. A common approach to measuring syntactic similarity involves comparing the shared substructures or nodes within the parse trees of two expressions, providing insights into their grammatical likeness.

Using both semantic and syntactic similarity between sentences can result in a much more robust model. Also, the original LexRank solves just the document summarization problem, and is not query based. In QSLK, both the sentence-to-sentence (S2S) similarity and sentence-to-query (S2Q) similarity are considered when assigning edge weights and solving the final markov chain. The weight given to syntactic and semantic similarity (α) and the weight given to S2S and S2Q similarity (β) are set as parameters in the code and can be controlled.

IV. QSLK Implementation

The following section outlines the QSLK algorithm and my implementation details. The process can be broadly classified into three steps:

- Preprocessing the document and the query.
- Creating the graph containing the S2S and S2Q weights.
- Formulating the Markov chain and solving it.

A. Pre-processing

This step involves preparing the document and the query for graph computations. Each sentence from the main document and the query is segmented and tokenized, followed by stop word removal. The sentences are then stored in a dictionary with keys being their IDs. It is important to have an ID associated with each sentence as they represent the nodes on the graph later on. For the purpose of the computations, the query is treated like just another sentence, but its ID is noted so that it can be separated at the time of solving the Markov chain. I have used Pythons NLTK library to do various NLP tasks. This code can be seen in *NLPWorker.py* file.

B. S2S and S2Q Graph

This is the core data structure of the process. The QSLK algorithm depends on the concept of voting, just like PageRank. The final score of a sentence is the votes it receives from other sentences and the votes it receives from the query. These scores and then weighted using parameters and summed up. To describe the algorithm, I will consider building an edge between and two sentences S1 and S2. For this purpose, the query is also treated like any other sentence.

Before the two sentences are evaluated, it is important to outline some data structures and algorithms used frequently by the process. These are:

- **WordSet:** For any two sentences S1 and S2, this is a set containing the root words of all their unique words. The root words are derived using WordNet which is available in Python's NLTK library. WordNet is a lexical database of the English language that relates words to one another in terms of synonyms, hypernyms, hyponyms, and more. It is a large lexical database of nouns, verbs, adjectives, and adverbs, where words are organized into sets of synonyms, each expressing a distinct concept. The following simple algorithm describes how to create the WordSet.

Algorithm 1: WordSet Creation for two sentences S1 and S2

```
Function create_wordset(S1, S2) is
    Set U1 = {}, Set U2 = {}
    forall words w in S1 do
        rw = WordNet.getRootWord(w)
        U1.add(rw)
    forall words w in S2 do
        rw = WordNet.getRootWord(w)
        U2.add(rw)
    return U1 $\cup U2$
```

- **Information Content:** In WordNet, the "information content" of a word refers to the specificity or uniqueness of the concept represented by that word in the WordNet hierarchy. It is often associated with the depth of the concept in the hierarchy. WordNet organizes words into a hierarchical structure, where more general terms (hypernyms) are located at higher levels and more specific terms (hyponyms) are located at lower levels. The information content of a word is influenced by its position in this hierarchy. In practical terms, information content is often used in information retrieval, natural language processing, and related fields. It can help measure the specificity of a term in a given context. For example, in the context of WordNet, the information content of a word might be calculated based on the depth of the node representing that word in the hierarchy. In the code, Information content is used in the sentence expansion process while calculating the syntactic and semantic score of two sentences. The formula used is:

$$IC(w) = 1 - \frac{\log(\text{synset}(w) + 1)}{\log((max_w))} \quad (3)$$

where synset(w) is the number of synonyms of the word w, obtained from WordNet, and maxw is the number of words in WordNet. In *NLPWorker.py*, I have set it to a static value of 155000.

- **Least Common Subsume (LCS):** The concept of the Least Common Subsumer (LCS) in the context of two concepts A and B refers to the most specific concept that is an ancestor of both A and B in a concept tree defined by the is-a relation. This tree is typically organized hierarchically, where the is-a relation represents a more general to more specific relationship. For example, consider a concept tree where "Vehicle" is more general than "Car" and "Bicycle." If A represents "Car" and B represents "Bicycle," the LCS in this case would be "Vehicle" because it is the most specific concept that is an ancestor (parent) of both "Car" and "Bicycle". This concept is commonly used in semantic similarity and ontology-related tasks, providing a measure of shared specificity between concepts. This implementation depends on identifying the LCS to quantify the semantic relatedness between terms in a hierarchical structure. Algorithm 2 is used to identify LCS , and is implemented in *NLPWorker.py.*
- **Word Similarity Score:** Aytar et al. 2008 [14], define the following formulation to determine the similarity score of two words w1 and w2:

$$\text{Sim}(w_1, w_2) = \begin{cases} \frac{2\times\text{IC}(\text{LCS}(w_1,w_2))}{\text{IC}(w_1)+\text{IC}(w_2)}, & \text{if } w_1 \neq w_2 \\ 1, & \text{if } w_1 = w_2 \end{cases} \quad (4)$$

where IC(LCS(w1,w2)) represents the Information Content of the Least Common Subsumer of w1 and w2. IC(w1) and IC(w2) are the Information Content of words w1 and w2 respectively.
- **Word Expansion Algorithm:** Content expansion in Natural Language Processing (NLP) involves increasing the amount of textual data available for a task. This is crucial for addressing challenges such as data scarcity, improving model generalization, and enhancing overall performance. By augmenting datasets synthetically or including diverse examples, content expansion helps models generalize better, handle out-of-distribution data, prevent overfitting, and adapt to changing scenarios. Techniques like data augmentation and exposure to a wider range of linguistic variations contribute to model robustness and effectiveness in various real-world applications.
In this project, I have used Algorithm 3 to get the expansion score between two words w1 and w2. This score predicts the likelihood of w1 and w2 to be similar. It is based on the concept of Word Similarity Score and LCS algorithm defined above.

Algorithm 2: Least Common Subsumer for two words w1 and w2

```
Function find_LCS(w1, w2) is
    List L1 = synsets(w1)
    List L2 = synsets(w2)
    common_hypernyms H = {}
    forall words wa in L1 do
        forall words wb in L2 do
            H.add(Lowest_Common_Hypernym(wa,wb))
    max_depth_hypernym ← 0
    max_depth ← 0
    forall hypernyms h in H do
        if h.depth > max_depth then
            max_depth_hypernym ← h
            max_depth ← h.depth
    return max_depth_hypernym
```

- **Semantic and Syntactic Vector of Sentences:** To calculate the semantic and syntactic score, we need to express the sentences S1 and S2 in vector form. For this, we create a semantic vector and a syntactic vector for each of the two sentences. The semantic vector represents the conceptual alignment of a sentence, whereas the syntactic vector represents the grammatical alignment Algorithm 4 below (Alguliev et al. 2011;Li et al. 2006b) [14] describes how to get

Algorithm 3: Content word expansion score (CWE) for w1 and w2

```
Function cwe(w1, w2) is
    root_w1 = WordNet.lemma(w1)
    root_w2 = WordNet.lemma(w1)
    if root_w1 == root_w2 then
        return 1
    ic_w1 ← $IC(root\_w2)$
    ic_w2 ← $IC(root\_w2)$
    lcs = LCS(root_w1,root_w2)
    return 2*IC(lcs)/(ic_w1 + ic_w2)
```

these vectors. Note the algorithm only shows how to calculate for S1, the same logic can be used for S2. This code has been implemented in *NLPWorker.py*

- **Semantic Score:** Given two sentences S1 and S2, their semantic score is defined as the dot product of their semantic vectors:

$$\text{sem\_score}(S1,S2)=\frac{\sum_{j=1}^{m}(w1_j\times w2_j)}{\sqrt{\sum_{j=1}^{m}w1_j^2}\times\sqrt{\sum_{j=1}^{m}w2_j^2}} \tag{5}$$

where w1 and w2 are the semantic vectors of S1 and S2 respectively.

- **Syntactic Score:** Given two sentences S1 and S2, the following formula is used to calculate their syntactic score. The formula calculates the similarity between the syntactic vector O1 and O2 using the Euclidean distance and normalization. The resulting similarity score ranges from 0 (completely dissimilar) to 1 (completely similar).

$$\text{syn\_score}(S1,S2)=1-\frac{||O1-O2||}{||O1+O2||} \tag{6}$$

- **Graph Edge Weight:** Given two sentences S1 and S2, a graph edge between them has a weight that is a combination of their semantic and syntactic score. This is controlled with a parameterα, which can be changed in *Constants.py*. An α value of 0.5 means equal weight is given to both semantic and syntactic similarity.

$$\begin{aligned}\text{Sim}(S1,S2)=&\ \alpha\cdot\text{sem\_score}(S1,S2)\\&+(1-\alpha)\cdot\text{syn\_score}(S1,S2)\end{aligned} \tag{7}$$

Algorithm 4: Getting the semantic and syntatic vectors of two sentences

```
Function vectorize(S1, S2) is
    word_set = create_wordset(S1, S2)
    sem1 = [0] * len(word_set)
    sem2 = [0] * len(word_set)
    syn1 = [0] * len(word_set)
    syn2 = [0] * len(word_set)
    forall words i,w in enumerate(word_set) do
        if w in S1 then
            sem1[i] ← 1
            syn1[i] ← $S1.index(w)$
        else
            max_score ← 0
            max_idx ← 0
            forall words j,wj in enumerate(S2) do
                score ← $cwe(wj,w)$
                if score > max_score then
                    max_score ← $score$
                    max_idx ← $j$
            sem1[i] ← $max\_score$
            syn1[i] ← $max\_idx$
```

- **Markov Chain:** After creating the graph which represents all S2S and S2Q weights, we can formulate the following markov chain to get the probability vector:

$$p=[\beta U+(1-\beta)W]p \tag{8}$$

Here W is a square matrix representing the similarity score of a sentence with every other sentence in the document. Its rows are normalized to add to 1. U is a square matrix representing the similarity score of each sentence with the query. The columns are broadcasted to match the dimension of W. β determines the weight given to S2Q similarity versus S2S similarity. A value of 0.5 means they are equally important. This parameter can also be controlled in *Constants.py.* The maximum number of iterations to run to solve the markov chain has also been given as a parameter in the code. The markov chain solution is implemented in *QSLK.py.* After we get p, we pick the top-K scores from it and use the corresponding sentences to create an extractive paragraph. K can also be controlled in *Constants.py*. If we want to stop here for the snippet, and not feed the Abstractive summarizers, then we set K to 2 and use the top 2 sentences for snippets.

- **Extra Visualizations in the code:** Debugging the similarity graph can be quite challenging. I have used the *networkx* package in the code to create the graph, and the graph and the weight matrix can be visualized for debugging purposes by setting the plot_viz variable in *Constants.py*. The diagrams below show a couple of these visualizations for a short document.

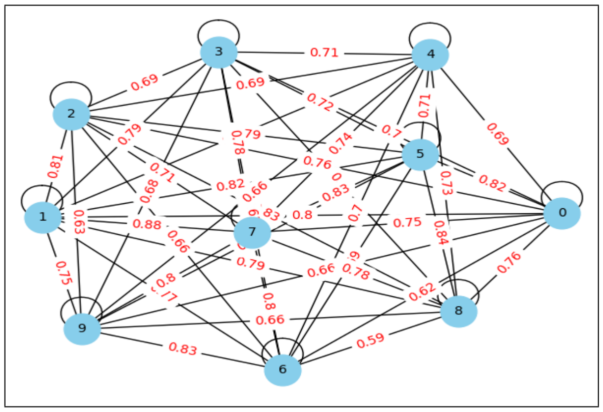

Fig. 1. S2S and S2Q Graph using networkx

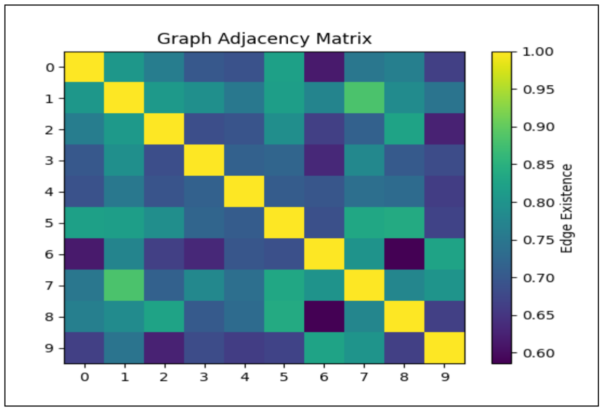

Fig. 2. S2S and S2Q Weight Matrix

V. Abstractive Summarization

The candidate paragraph that extractive summarization generates can now be passed to various abstractive models for further smoothing and cohesion.The history of transformer-based models in natural language processing (NLP) traces back to the introduction of the transformer architecture in the seminal paper "Attention is All You Need" (2017) by Vaswani et al [15]. The innovative self-attention mechanisms of transformers revolutionized sequence-to-sequence modeling. Building on this, BERT [16] (Bidirectional Encoder Representations from Transformers) emerged in 2018, showcasing significant advancements by pre-training on extensive corpora. OpenAI's GPT-2 (Generative Pre-trained Transformer 2) in 2019 demonstrated state-of-the-art language generation. The year 2020 brought T5 (Text-To-Text Transfer Transformer) by Raffel et al. [17], introducing a unified text-to-text paradigm for NLP tasks. GPT-3 (Generative Pre-trained Transformer 3) in 2020 set new benchmarks with an impressive 175 billion parameters, showcasing remarkable few-shot learning capabilities across diverse language tasks. For the purpose of this paper, I used the following pre-trained models to generate snippets from the candidate paragraphs:

- **T5-base:**
  Introduced by Raffel et al. in "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer" (2020) [17], represents a pivotal variant of the Text-To-Text Transfer Transformer (T5) model. T5-base is an encoder-decoder architecture pre-trained on a diverse combination of unsupervised and supervised workloads, with tasks unified under a text-to-text framework. This design allows T5-base to handle various natural language processing (NLP) tasks effectively. The model utilizes relative scalar embeddings, and encoder input padding can be performed on both the left and right sides.
- **Facebook/BART-large-CNN:**
  Introduced by Lewis et al. in "Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation and Comprehension" (2019) [18], is a significant model in the realm of transformers. BART (Bidirectional and Auto-Regressive Transformers) combines a bidirectional encoder, resembling BERT, with an autoregressive decoder similar to GPT. The pre-training process involves denoising the sequence of original phrases through random rearrangement and in-filling. Facebook/BART-large-CNN excels in various text production

tasks, particularly in summarization and comprehension, demonstrating its effectiveness with a dual-encoder-decoder architecture.

- **Google/PEGASUS-xsum:**
  Introduced by Zhang et al. in "PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization" (2019) [19], is a model designed specifically for abstractive summarization. PEGASUS adopts a unique pre-training task closely aligned with summarization objectives, where key sentences are strategically removed or masked, and the remaining phrases are combined into a cohesive output sequence resembling an extractive summary. PEGASUS achieves robust summarization performance across various downstream datasets, evaluated using ROUGE and BLEU metrics.
- **BERT:**
  Introduced by Devlin et al. in "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" (2018) [16], BERT utilizes a bidirectional transformer architecture to pre-train on large corpora, learning contextualized word representations. The pre-training task involves predicting masked words in a given context, enabling BERT to grasp intricate dependencies and nuances in language. BERT's strength lies in its contextual embeddings, which capture the meaning of words based on surrounding context. Its bidirectional approach allows the model to consider both preceding and following words, enhancing its understanding of context. BERT is primarily designed for tasks like language understanding rather than abstractive summarization. However, it can be adapted for extractive summarization, which involves selecting and assembling existing sentences from the input document to form a summary.

I have used BERT for extractive summarization to compare it's performance with the QSLK model that was developed earlier. Having an off-the-shelf extractive summarize like BERT gives us a good threshold to judge QSLK as a standalone model, rather than a feed into more complex transformer based models. Using QSLK, BERT, and the three abstractive models described above, I ran tests with 7 different combinations and will discuss the results in the next section. All these off-the-shelf models have been implemented in *Abstractive_Summarizer.py*

VI. Data setup

There are plenty of summarization datasets available on the internet. But most of these are not suitable from a web search engine point of view. The purpose of this project was to implement snippet generation, and there is no gold-standard snippet generation dataset available. Creating such datasets often needs human evaluation, and one of the potential candidates was the DUC 2005 and 2006 dataset. However, these datasets are not available until certain legal procedures are completed. Even if they were available, they are useful for seeing gold-standard summaries, not snippets. The difference is that a snippet is on average 1.5 sentence long, whereas a summary is much longer. There is also the challenge of finding a generic query based summarization dataset. There are a few available that focus on a particular topic but they would not be a fair test for this project. Some datasets are also eliminated as they contain huge documents whose processing needs a lot of computational power. Ultimately, lack of a generic dataset and more importantly, the lack of gold-standard snippets was a major bottleneck in this project.

Fortunately, QSLK is not a machine learning based algorithm and all the other models have been pre-trained, so we don't need massive amounts of data to do testing. For the purpose of this project, I created a dataset of 50 test samples manually. I created this dataset by searching random queries on google, and using google's own snippets as the gold-standard. There are other problems with this as will be described in testing, but it does a good job for the most part. The structure of each test sample is a text file that contains the query in the <q> tags, the document in the <d> tags and the google snippet in the <summ> tags. Code has been added to read this format and adding a new test sample is trivial. The dataset will be shared along with the project.

VII. Testing

As discussed above, it was decided that Google's own snippets would be used as the gold standard. This is not completely ideal. Firstly, gold standard in summarization is traditionally human evaluated. To compare a snippet which is generated by an algorithm against a snippet which is also generated by an algorithm is not ideal. In many cases, both snippets are appropriate but different. There is no way to break the tie without human evaluation. Secondly, snippets are very short texts, and google's snippets are often truncated and incomplete. This poses a problem in comparison too.

BLEU (Bilingual Evaluation Understudy) is a metric commonly used to evaluate the quality of machine-generated text, such as machine translation or text summarization. BLEU measures the similarity between the generated text and one or more reference texts, which are typically human-generated. The BLEU score ranges from 0 to 1, where higher values indicate better agreement between the generated and reference texts. BLEU is sensitive to the length of the generated text compared to the reference text. If the summaries are consistently short, BLEU might not provide a fair evaluation, as shorter summaries can lead to higher BLEU scores due to the brevity penalty. The brevity penalty is described as:

$$\text{Brevity Penalty} = \min\left(1, \frac{\text{output length}}{\text{reference length}}\right) \quad (9)$$

Thus, the problem with using BLEU, atleast in an absolute manner in this case, is that for snippets, which are generally 2 sentences at max, the match could be very good hit or a very bad miss. For summaries, BLEU works very well as there is a lot more content to work with. This was my observation with the testing as well. All the models suffer from this problem. So, rather than considering BLEU to be absolute in this case, BLEU values have to interpreted on a relative basis. As all snippets are benchmarked against a very short google snippet, their relative score is much more useful. Thus a relative application of BLEU has been implemented. For each test case, against each kind of model, a winner is selected. The winner is the one with the best BLEU score. At the end of the test cases, the mean, median, max, min, 25 percentile and 75 percentile of each model is also printed out. A separate python class *run_all_tests.py* has been implemented for this and we just need to set the appropriate test and output directories

Finally, I also got the snippets tested by a couple of my colleagues at NYU. They are Syed Ahazam Tariq (sat10045@nyu.edu) and Junaid Girkar (jg7649@nyu.edu). They were given the document, google snippet, query, and each model's snippet for evaluation and ranked each model's performance out of 10. This score has been reported too. This again provides a more balanced evaluation as we have both machine evaluation and human evaluation.

For each test run, 30 samples were selected from the test set, and the following 7 models were used to generate snippets:

1. QSLK - Only: This uses only the QSLK algorithm implemented in this paper. This is completely extractive and gives the baseline performance of QSLK.

2. BERT (Query-guided): This uses only the query guided BERT model. This is completely extractive and gives a good comparison for QSLK - Only

3. T5 (Query-guided): This uses only the query guided T5 model. This is completely abstractive.

4. BART-Large-CNN (Query-guided): This uses only the query guided BART-Large-CNN model. This is completely abstractive.

5. QSLK(Query-guided) + T5 (No-Query): This uses the hybrid approach which is the focus of this paper. It uses QSLK along with the query to generate a candidate paragraph and then uses T5 to generate a snippet from it without the query context

6. QSLK(Query-guided) + BART-Large-CNN(No-Query): Same as model no. 5 but uses BART-Large-CNN model.

7. QSLK(Query-guided) + pegasus-xsum(No-Query): Same as model no. 5 but uses pegasus-xsum model.

VIII. Results Analysis

The results are quite interesting. In general, it was observed that BART-Large-CNN and T5-

TABLE I

Performance of the models ($\alpha = 0.5, \beta = 0.5$)

Model Name	Statistics				
	Mean	**Max**	**Min**	**Win Count**	**Win Share**
QSLK - Only	0.11	0.57	0.03	1	3.40%
BERT (Query-guided)	0.18	0.62	0.04	6	20.60%
T5 (Query-guided)	0.16	0.58	0.06	7	23.80%
BART-Large-CNN (Query-guided)	0.17	0.5	0.04	7	23.80%
QSLK(Query-guided) + T5 (No-Query)	0.12	0.47	0.03	5	17.00%
QSLK(Query-guided) + BART-Large-CNN(No-Query)	0.13	0.47	0.03	1	3.40%
QSLK(Query-guided) + pegasus-xsum(No-Query)	0.05	0.31	0.004	2	6.80%

TABLE II

Performance of the models ($\alpha = 0.8, \beta = 0.3$)

Model Name	Statistics				
	Mean	**Max**	**Min**	**Win Count**	**Win Share**
QSLK - Only	0.10	0.44	0.02	1	3.40%
BERT (Query-guided)	0.18	0.62	0.03	4	13.60%
T5 (Query-guided)	0.16	0.58	0.05	7	23.80%
BART-Large-CNN (Query-guided)	0.17	0.5	0.04	7	23.80%
QSLK(Query-guided) + T5 (No-Query)	0.12	0.47	0.03	5	17.00%
QSLK(Query-guided) + BART-Large-CNN(No-Query)	0.14	0.47	0.04	2	6.80%
QSLK(Query-guided) + pegasus-xsum(No-Query)	0.10	0.93	0.01	3	10.2%

TABLE III

Performance of the models ($\alpha = 0.2, \beta = 0.7$)

Model Name	Statistics				
	Mean	**Max**	**Min**	**Win Count**	**Win Share**
QSLK - Only	0.09	0.39	0.01	0	0%
BERT (Query-guided)	0.18	0.62	0.10	9	30.6%
T5 (Query-guided)	0.16	0.58	0.06	7	23.80%
BART-Large-CNN (Query-guided)	0.17	0.50	0.04	7	23.80%
QSLK(Query-guided) + T5 (No-Query)	0.10	0.46	0.04	1	3.40%
QSLK(Query-guided) + BART-Large-CNN(No-Query)	0.13	0.53	0.02	3	10.2%
QSLK(Query-guided) + pegasus-xsum(No-Query)	0.07	0.20	0.01	2	6.80%

TABLE IV

Human Evaluation of the models ($\alpha = 0.5, \beta = 0.5$)

Model Name	**Person A mean score**	**Person B mean score**
QSLK - Only	4.3/10	5.4/10
BERT (Query-guided)	5.3/10	6.7/10
T5 (Query-guided)	4.6/10	5.6/10
BART-Large-CNN (Query-guided)	5.3/10	6.1/10
QSLK(Query-guided) + T5 (No-Query)	5.5/10	6.6/10
QSLK(Query-guided) + BART-Large-CNN(No-Query)	7.0/10	8.3/10
QSLK(Query-guided) + pegasus-xsum(No-Query)	4.8/10	5.1/10

base were the most consistent out of all the abstraction models, whereas BERT easily beat QSLK based on extractive summarization alone. But as predicted, the combination of extractive candidate generation using QSLK followed by abstractive summarization won the most. These are the last three models in the tables above. For example, when $\alpha = 0.5$ and $\beta = 0.5$, the hybrid models won 27.2% of the test cases. In the second table we see that increasing α to 0.8 and giving more weight to semantic similarity at the time of candidate generation did even better. The hybrid models won 34% of the time, with an especially significant improvement in the QSLK(Query-guided) + pegasus-xsum(No-Query) model. This model seems to be the most sensitive to the parameter tuning of QSLK, out of all the 3 hybrid models. In table 3, I decreased α to 0.2, which means most weight was given to syntactic similarity. This is obviously not a good choice, and as expected it significantly degraded the QSLK algorithm (its win share became 0% and had very low statistics), and all the downstream hybrid models also struggled due to bad candidate generation. This is a strong evidence of how good candidate generation can affect the performance of abstractive models. Their win percentage dropped from 34% to around 21%.

The last table gives the mean scores of the documents evaluated by my two colleagues. Their analysis was that QSLK-pegasus hybrid is often a hit or a miss. It was the only model that scored a perfect ten. But its scores were quite low in general. QSLK(Query-guided) + BART-Large-CNN(No-Query) was mutually agreed to be the best-performing model, as shown by the score given by both judges. Although this model won a few test cases in the BLEU score-based contest, it is not at all apparent from the BLEU score and its win count that this model performs so much better than others. This goes back to the point discussed before about the limitation of using BLEU for evaluating snippets. For example, they also reported seeing high BERT scores even though the snippet wasn't that good. This could be because BERT can be directly controlled to produce very few sentences (2 at max, in this case), and BLEU score rewards shorter content. However, the general takeaway is still that human evaluation is more trustworthy in this case, and the BLEU scores should be interpreted relatively.

IX. CONCLUSION

From the results above, it could be seen that the hybrid models performed better and won more test cases than either extractive or abstractive models alone. Since extractive summarization can be done beforehand on documents, this hybrid approach can be quite useful to especially summarize very long documents. In this project, QSLK(Query-guided) + BART-Large-CNN(No-Query) was selected as the best hybrid summarize. I would like to work more on this project in the future, if possible. It would be great if these models can be run on DUC data sets. I posit a good improvement in individual BLEU and ROGUE scores. Reliance on google's snippets as gold standard was not a good assumption. More work is needed on finding query based summarization datasets from a web search engine point of view.

X. ACKNOWLEDGMENT

I would like to thank Dr. Torsten Suel for giving me an opportunity to work on this project, and the feedback about my work. I would also like to thank my colleagues Syed Ahazam Tariq and Junaid Girkar for finding the time to do a human evaluation of snippets.

REFERENCES

[1] "Recent automatic text summarization techniques: a survey,". Artif. Intell. Rev., vol. 47, no. 1, pp. 1–66, Jan. 2017, doi: 10.1007/s10462-016-9475-9.https://www.researchgate.net/publication/299499824_Recent_automatic_text_summarization_techniques_a_survey.

[2] A survey on extractive text summarization. N. Moratanch; S. Chitrakala Available: https://ieeexplore.ieee.org/document/7944061

[3] Automatic Summarization By Ani Nenkova and Kathleen McKeown. Available: https://www.cis.upenn.edu/~nenkova/1500000015-Nenkova.pdf.

[4] Erkan, G. and Radev, D.R. (2004) Lexrank: Graph-Based Lexical Centrality as Salience in Text Summarization. Journal of Artificial Intelligence Research, 22, 457-479. https://arxiv.org/abs/1109.2128.

[5] A survey on Automatic Text Summarization.N. Nazari M. A. Mahdavi https://doi.org/10.22044/jadm.2018.6139.1726.

[6] Content Based Weighted Consensus Summarization. Parth Mehta, Prasenjit Majumder .

[7] Ouyang Y, Li W, Li S, Lu Q (2011) Applying regression models to query-focused multi-document summarization. Inf Process Manag 47:227–237 [Online]. https://www.researchgate.net/publication/220229610_Applying_regression_models_to_query-focused_multi-document_summarization.

[8] Basak D, Pal S, Patranabis DC (2007) Support vector regression. Neural Inf Process Lett Rev 11:203–224

[9] TextRank: Bringing Order into Text (Mihalcea Tarau, EMNLP 2004) https://aclanthology.org/W04-3252.pdf

[10] Brin, S. and Page, L. (1998) The Anatomy of a Large-Scale Hypertextual Web Search Engine. Computer Networks and ISDN Systems, 30, 107-117. http://dx.doi.org/10.1016/S0169-7552(98)00110-X

[11] Query-based multi-documents summarization using linguistic knowledge and content word expansion. DOI: 10.1007/s00500-015-1881-4. Asad AbdiA Norisma Idris Ramiz Aliguliyev. https://www.academia.edu/19401264/Query_based_multi_documents_summarization_using_linguistic_knowledge_and_content_word_expansion

[12] CTSUM: extracting more certain summaries for news articles. Xiaojun Wan and Jianmin Zhang Pages 787–796 https://doi.org/10.1145/2600428.2609559

[13] G. Yang, Kinshuk, D. Wen and E. Sutinen, "A Contextual Query Expansion Based Multi-document Summarizer for Smart Learning," 2013 International Conference on Signal-Image Technology Internet-Based Systems, Kyoto, Japan, 2013, pp. 1010-1016, doi: 10.1109/SITIS.2013.163.

[14] Abdi A, Idris N, Alguliev RM, Aliguliyev RM (2015) Automatic summarization assessment through a combination of semantic and syntactic information for intelligent educational systems. Inf Process Manag 51:340–358

[15] Attention Is All You Need. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. https://arxiv.org/abs/1706.03762

[16] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova https://arxiv.org/abs/1810.04805

[17] Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J. Liu https://arxiv.org/abs/1910.10683

[18] BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, Luke Zettlemoyer https://arxiv.org/abs/1910.13461

[19] PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization. Jingqing Zhang, Yao Zhao, Mohammad Saleh, Peter J. Liu https://arxiv.org/abs/1912.08777