Fast, Semantic Command-F for Intelligently Searching Unstructured Content

Prathik Naidu, Anirudh Jain, Dian Huang

Problem

Searching for information across a document or webpage is a critical and necessary use-case that has wide applicability across many domains. However, current solutions do not fully serve this need. Notably, Command-F is a classic, well-known keyboard shortcut used to quickly find a word or phrase in a web page or document. This form of "search" across content is rudimentary because it doesn't allow a user to ask questions and find relevant portions of a document beyond an exact match. Instead, traditional Command-F relies on keyword based similarity, which is limiting because you can't ask full questions and rely on semantic intent to find answers on a page; users have to guess what keywords will lead them to the information they seek. In particular, these issues become increasingly problematic for long documents where information can be dense and difficult to parse. Thus, allowing users to directly query the web page and returning the relevant answers is a much more effective and intuitive approach.

"Finding information on long and complex web pages is hard and I have to guess what to search with ctrl-F "

Stanford Student

"I'm frustrated because I have to keep guessing keywords or just skim the document"

Stanford Student

Recent advancements in language models, namely GPT-3, are an impressive showcase of advanced Q&A use cases – these methods are able to parse the semantic meaning of long documents or webpages (i.e. semantic search), drawing meaningful conclusions across ideas described throughout the writing. There are two key limitations for this existing work: 1) there does not exist an easy-to-use interface or application that enables a user to ingest content and answer semantic queries over that text and 2) continuously pinging a model for each query results in high latency and creates negative user experiences at scale. Traditional command-F shines because it's an incredibly lightweight experience for quickly finding word on a page and does so nearly instantaneously.

Our Solution

In light of both the pros and cons of regular command-F, we had two important requirements when building out a solution for intelligently searching web pages: first, the application must have a **friendly, easy-to-use interface** and second, the application must be **optimized for low latency** to improve the user experience. Thus, after careful consideration, our final solution was a chrome extension for semantic search of text where users can directly ask questions and have the relevant answers highlighted on the webpage.

In summary, there were three key features in the application:

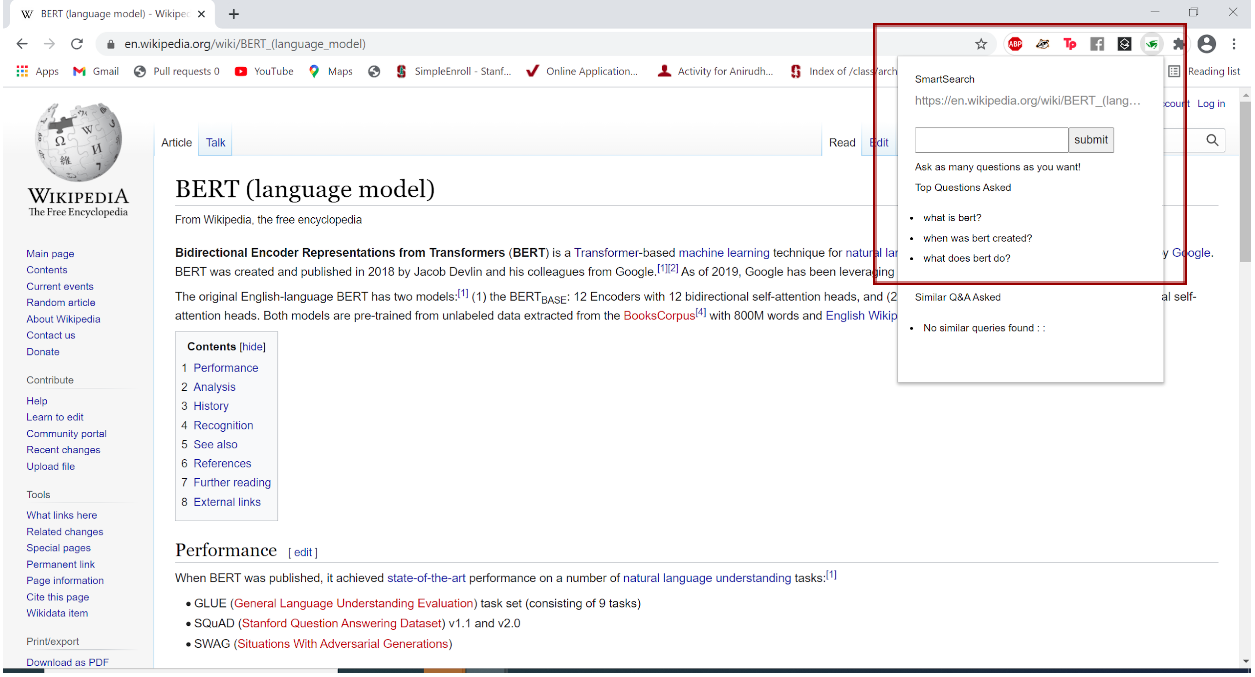
1. Low latency semantic querying of webpages (Q&A)
2. Top asked questions per web page with cached answers
3. Semantically similar questions and associated answers

Application Design

To satisfy the requirement of easy-to-use, we built a chrome extension as our final UI for enabling users to provide semantic queries (questions) and find answers directly on the page. We initially explored building a separate web application (i.e. website) where users could link a webpage and type questions. However, we quickly realized that this would be a suboptimal user experience since people don't want to open up a new tab in order to find content on a different page. Instead, they just want to find their answer with whatever page is already open. A chrome extension was natural here because it's not invasive to a users workflow (just requires clicking a button on the extensions bar) and enables us to write javascript for directly modifying the webpage (we took advantage of this by highlighting the relevant answers in the text based on the model results). Our design choice here resulted in an application that was similar in UI to regular command-F, but was far more powerful as it enabled complex semantic queries. As a result, the system complexity was completely hidden from the user.

To further the goal of making this as easy-to-use for the end-user, we decided to add two additional features beyond semantic question and answering. The first was to display the top asked questions per webpage (across all users) in the chrome extension and enable users to click on these questions and see the corresponding answers immediately highlighted on the page. This allows users to see what the most popular queries on the information are, immediately get the results from themselves, and can be a dramatic speed-up if the user's intended query was one of the top queries. The second was to have the extension show the most semantically similar questions (across all web pages and users), their corresponding answers, and the webpage they came from. Our goal here was to not only provide an interface that was simple and easy to use for searching a single page, but enable the user to leverage a knowledge base that spanned the entire internet, and further enhance their semantic querying. Similar Q&A's provides additional powerful insights for the user into the domain they are exploring beyond the webpage they are on. Even more importantly, as we scale and users keep searching across pages, we develop a larger corpus of related questions and answers and top Q&A's to show on the application--the experience and search capabilities of our tool continuously get better with more usage (self-feeding cycle)!

As an example, we'll walk through a workflow for how a user might interact with our application. First, starting the app is as simple as loading a webpage and hitting the button for our chrome extension on the toolbar. From there, the search window pops up, showing a textbox to enter your query.



SmartSearch

https://en.wikipedia.org/wiki/BERT_(lang…

Ask as many questions as you want!

Top Questions Asked

- what is bert?
- when was bert created?
- what does bert do?

Similar Q&A Asked

- No similar queries found : :

BERT (language model)

From Wikipedia, the free encyclopedia

Bidirectional Encoder Representations from Transformers (**BERT**) is a Transformer-based machine learning technique for natural lar … y Google. BERT was created and published in 2018 by Jacob Devlin and his colleagues from Google.[1][2] As of 2019, Google has been leveraging

The original English-language BERT has two models:[1] (1) the $\text{BERT}_{\text{BASE}}$: 12 Encoders with 12 bidirectional self-attention heads, and (2 … al self-attention heads. Both models are pre-trained from unlabeled data extracted from the BooksCorpus[4] with 800M words and English Wikip

Contents [hide]

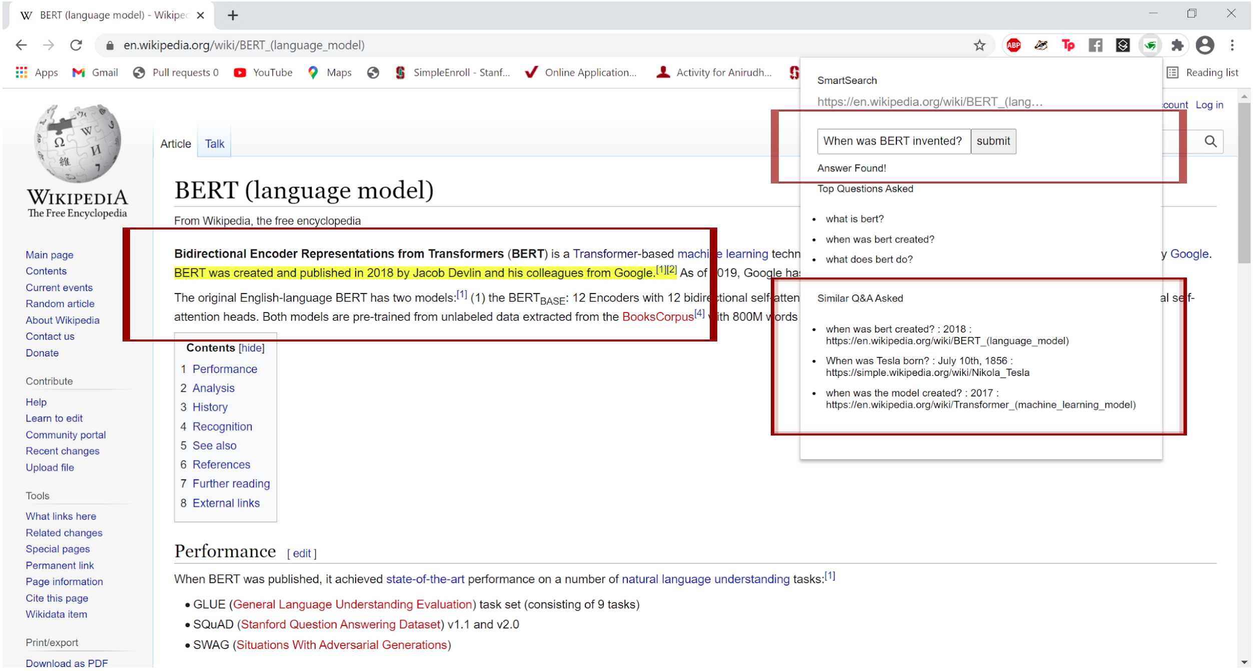
Performance [edit]

When BERT was published, it achieved state-of-the-art performance on a number of natural language understanding tasks:[1]

- GLUE (General Language Understanding Evaluation) task set (consisting of 9 tasks)
- SQuAD (Stanford Question Answering Dataset) v1.1 and v2.0
- SWAG (Situations With Adversarial Generations)

We also show the top questions ("what is bert?") that were previously asked on that page (across all users) to get context on what might be useful to search for. If a user clicks on one of the top asked questions, the answer will instantaneously be highlighted on the page (as if the user had queried the question themselves) as we cache the associated answers to the top questions in the front-end.

The user can then enter a query, for example "When was BERT invented?" and the application will highlight all the relevant text on the page that answers the user's question, making it easy to answer complex questions about a long piece of text.



BERT (language model)

From Wikipedia, the free encyclopedia

Bidirectional Encoder Representations from Transformers (**BERT**) is a Transformer-based machi e learning techn y Google. BERT was created and published in 2018 by Jacob Devlin and his colleagues from Google.[1][2] As of 019, Google ha

The original English-language BERT has two models:[1] (1) the $\text{BERT}_{\text{BASE}}$: 12 Encoders with 12 bidire tional self atten al se f-attention heads. Both models are pre-trained from unlabeled data extracted from the BooksCorpus[4] ith 800M ords

Contents [hide]

1 Performance
2 Analysis
3 History
4 Recognition
5 See also
6 References
7 Further reading
8 External links

Performance [edit]

When BERT was published, it achieved state-of-the-art performance on a number of natural language understanding tasks:[1]

- GLUE (General Language Understanding Evaluation) task set (consisting of 9 tasks)
- SQuAD (Stanford Question Answering Dataset) v1.1 and v2.0
- SWAG (Situations With Adversarial Generations)

SmartSearch

https://en.wikipedia.org/wiki/BERT_(lang…

When was BERT invented? submit

Answer Found!

Top Questions Asked

- what is bert?
- when was bert created?
- what does bert do?

Similar Q&A Asked

- when was bert created? : 2018 : https://en.wikipedia.org/wiki/BERT_(language_model)
- When was Tesla born? : July 10th, 1856 : https://simple.wikipedia.org/wiki/Nikola_Tesla
- when was the model created? : 2017 : https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)

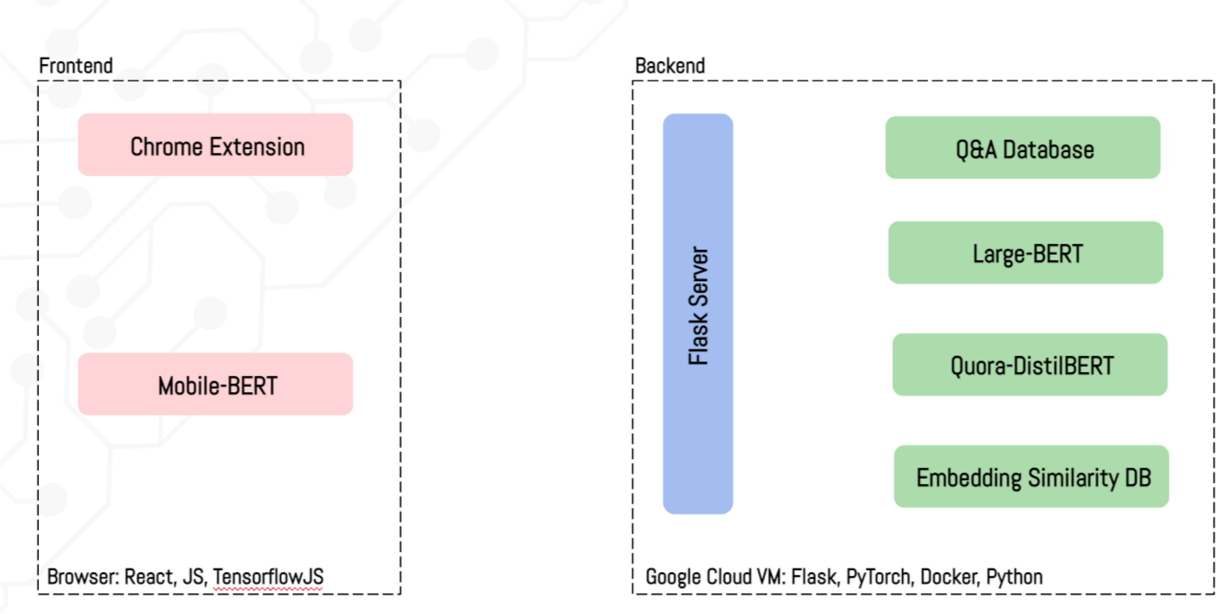
Moreover, in addition to showing the answer to the user's query, the application also shows semantically similar questions that have been previously queried (across all users), the corresponding answers, and the webpage where the information was found. For example, we can see that it lists "when was the model created?", shows that the answer is "2017", and links to the Transformer wikipedia page. This allows the user to explore similar information (i.e. Transformers) across the entire web.

A video demo can be seen at this link: https://drive.google.com/file/d/1LevO2rHonksGCEIUeID2skasknOdnsL-/view?usp=sharing

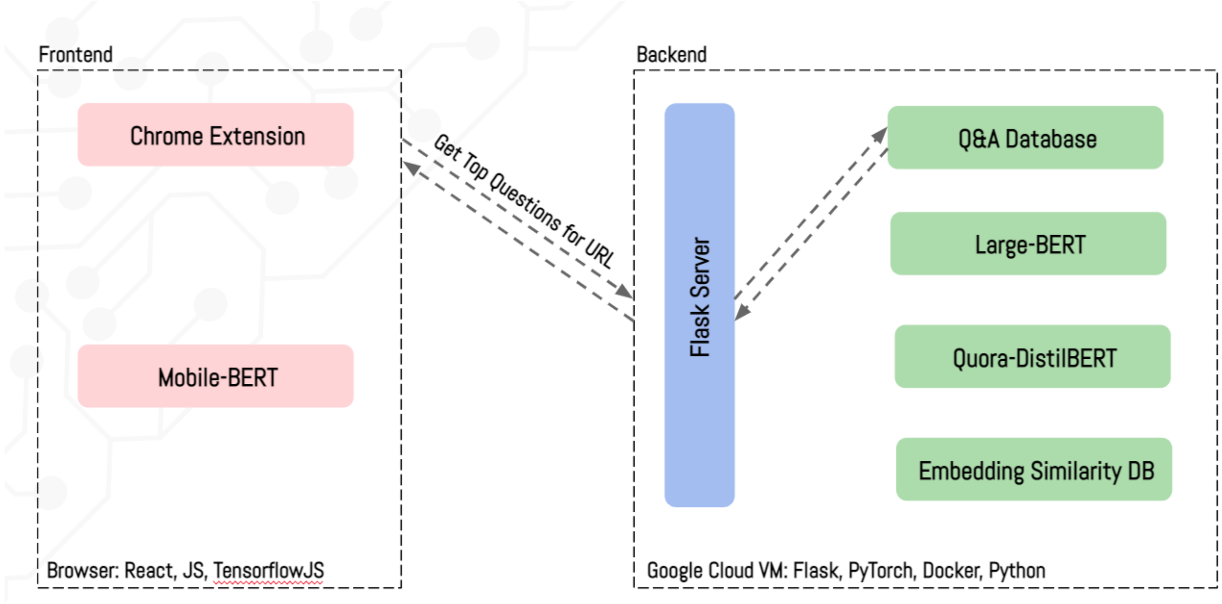
Machine Learning and System Design

To satisfy our second requirement of low latency, we had to carefully iterate and engineer our system so that we could keep our powerful feature set as previously described while still maintaining fast performance. We will walk through our overall system and explain the system optimizations concurrently.

As shown, our frontend runs entirely on the browser through a chrome browser extension. This is built through React (UI), Javascript (frontend logic such as extracting text from webpage), and we have TensorflowJS running as well for our edge compute model MobileBERT [1]. The backend is running on a GPU-backed Google VM server (currently just a K80 Tesla) with a Python Flask server as an orchestration engine for REST API calls from the front end, a hosted database for storing all questions and answers across user (Q&A database), a docker container running a Milvus database to store question embeddings (Embedding Similarity DB), and Pytorch-backed models for question and answers (Large-BERT [3]) and sentence embedding (Quora-DistilBERT [3]).

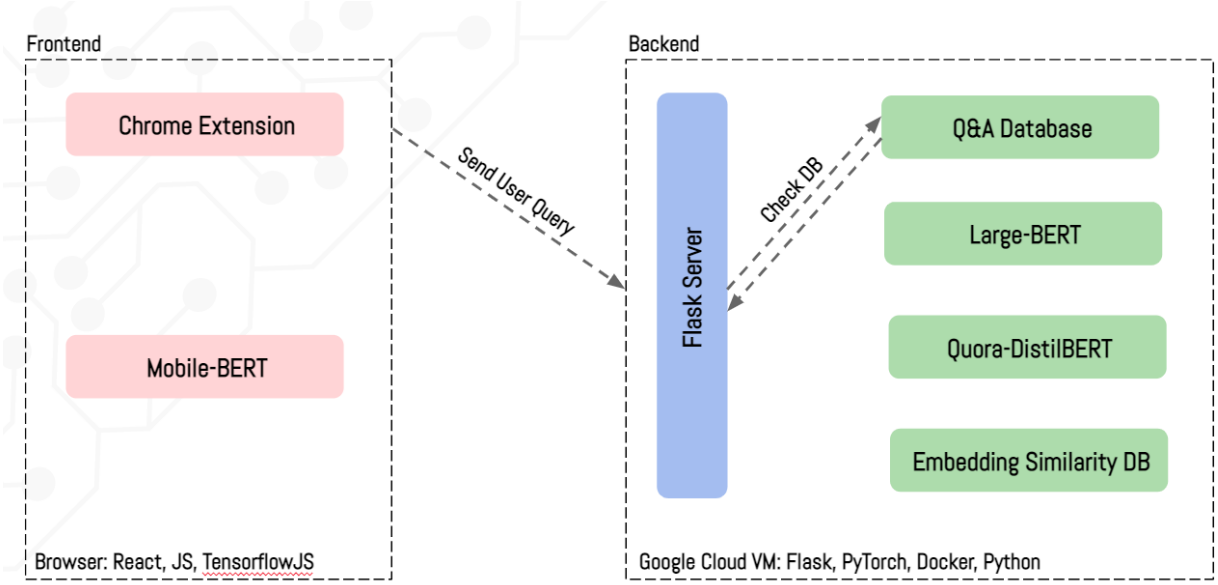


We will now walk through how the system components work together to implement each of our three features (listed above).

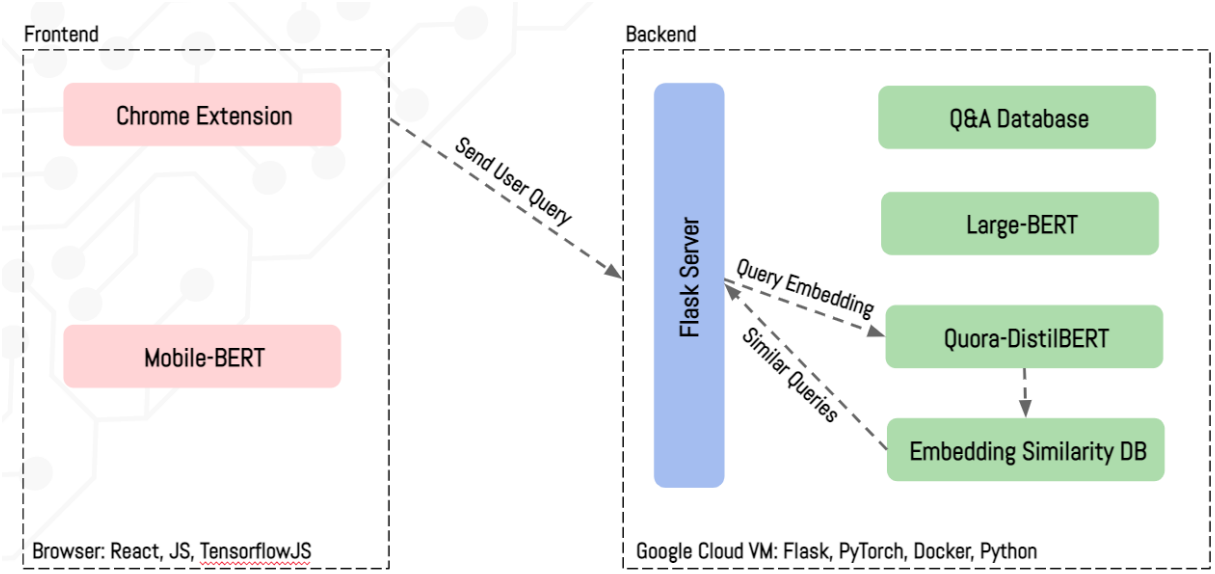


When a user first visits a web page and opens the extension, the chrome extension immediately sends an API call to the Flask server to retrieve the top asked questions for that URL. The Flask server then queries the Q&A database and grabs the top questions and answers for the URL and returns them to the frontend where they are displayed and the answers are cached. In addition to being a useful feature, this also saves a lot of computation resources as oftentimes users have the exact same queries so immediately retrieving the most popular Q&A's prevents the front-end and back-end from searching for the answer again and reduces latency.

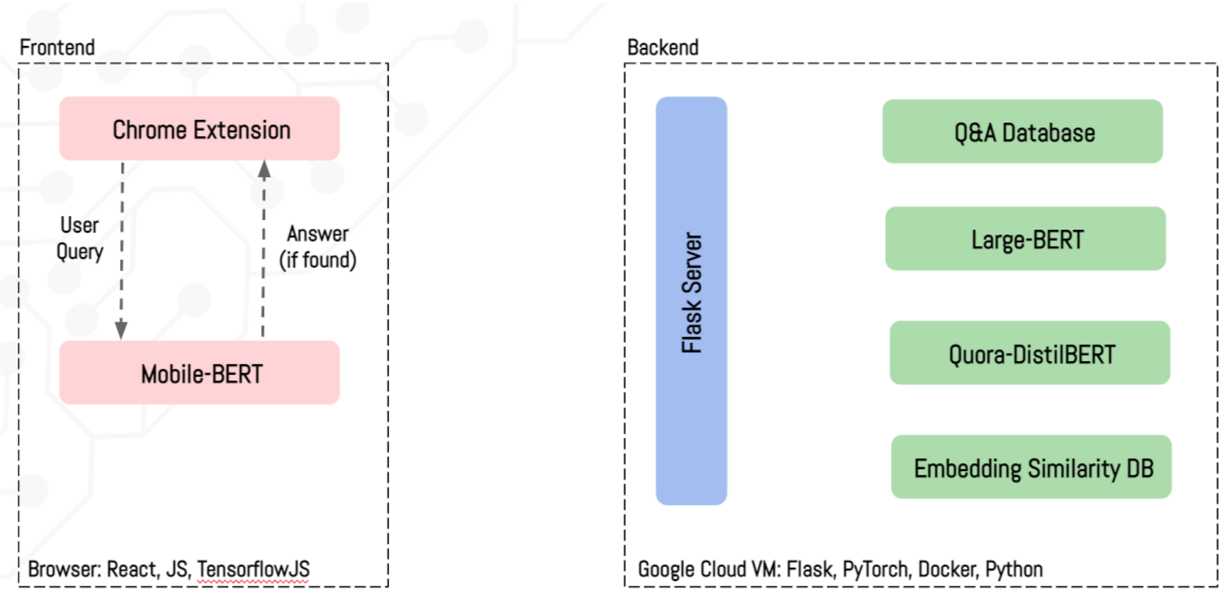
When a user enters in a query, the browser immediately sends an API call to the Flask server to check the Q&A database to see if the query has been previously asked and retrieves the stored answer and returns it to the frontend to display. We built our own relational-like database to store questions, answers, and URLs and allow for fast indexing. Having this database prevents repeated computation on duplicate queries and allows us to get the answer near instantaneously. Having this data also allows for the development of features like the most popular Q&As and similar Q&As (any potentially many others!) and is an invaluable resource. In fact, as more users are using our product and it scales, our database will store more and more previously asked questions, so it will be more likely to hit on a previous question and the application performance should only improve with greater use.



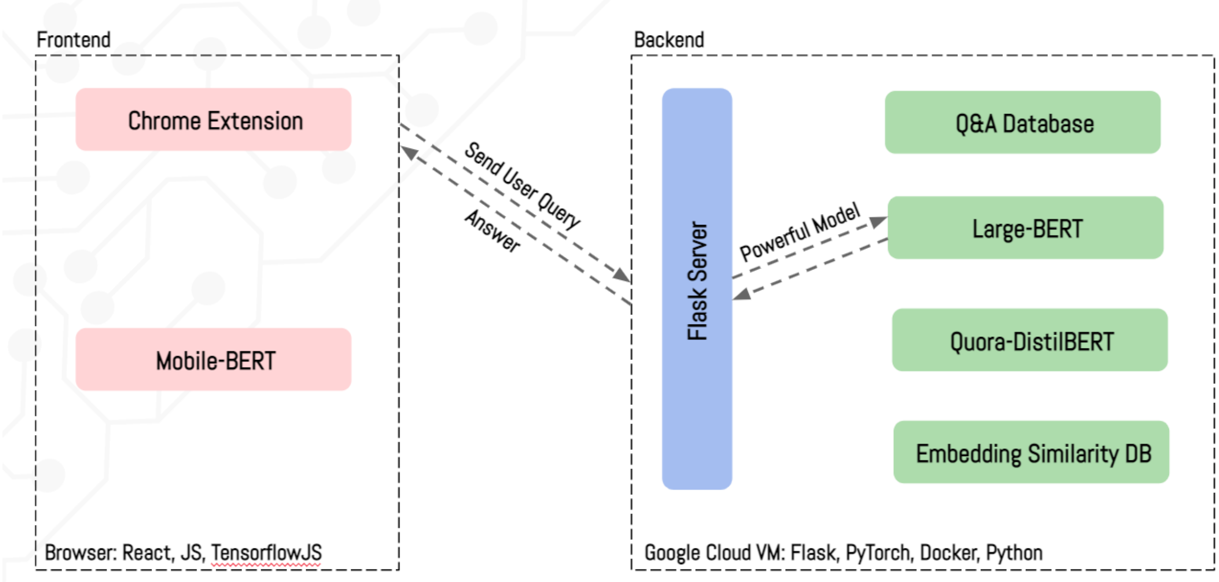
While looking for a stored answer to the user query, the extension simultaneously sends an API call to find similar questions and answers. First, the query is run through a sentence embedding model (Quora-DistilBERT model [3]), the embedding is generated and added to the embedding database, and then the closest query embeddings are found through a nearest neighbor search (inner product distance) on the embedding database. The closest query embeddings within a threshold are then translated to their respective questions, answers, and URLs via the Q&A database and returned to the frontend to display. The embedding similarity database is a special database optimized for nearest neighbor searches built on top of optimized Approximate Nearest Neighbor Search (ANNS) indexing libraries (https://milvus.io/).



If the query and answer is not found in the Q&A database, then the query is run through an edge compute model (MobileBERT [1]) which is running on TensorflowJS in the browser. Since we're running a model to identify relevant parts of an article given a user's question, we're inherently bottlenecked by the backend model inference speed and round trip communication.. Experimentation showed this can take surprisingly long, especially for long bodies of text. In order to minimize latency as much as possible, we decided to run a MobileBERT directly in the browser to answer the user query from the text. However, as this runs on the browser, it takes a significantly longer time to process long texts. To solve this, we batch processed where we run the MobileBERT on each sequence with a maximum of 2500 characters batch by batch and highlight the answer on the webpage for each batch. The advantage is that the user does not have to wait for a complete processing of all text content to get the first answer.



Lastly, we have confidence bounds on our model predictions and if the MobileBERT frontend model is not able to find a satisfactory answer in the text (as it is a less powerful model), the extension sends an API call to run the query and text through the more powerful backend model (Large-BERT [2]) and get an answer. We reserve this as the last-step in the Q&A process because it is the most computationally expensive as it involves querying the backend and running a much larger and more powerful model that has slower inference time. Only the most complicated, difficult, and unique queries should be routed here (by our logic) as the Q&A database and frontend model should be able to handle most queries.



Overall, we applied several system-level optimizations for latency (see evaluation section for more detail) including a Q&A database, embedding database, caching top asked questions, and an edge computing model that greatly lower latency greatly and will lend to even further performance boosts as the application scales. Furthermore, doing these optimizations led to much more powerful and robust features such as top Q&A's and semantically similar Q&As.

Machine Learning Models

The models and how they are used are largely described in the previous section. We will briefly describe the models and what they were trained or fine-tuned on.

Mobile BERT [1]: The model is a pre-trained MobileBERT model fine-tuned on SQuAD 2.0 dataset running on TensorflowJS. This model was chosen because it is a compressed version of BERT that runs 4x faster and has 4x smaller model size so is well-optimized for running on the edge. This has been fine-tuned on SQuAD which is a dataset consisting of articles from Wikipedia and a set of question-answer pairs for each article. This makes this model optimized for fast Q&A. The model takes a tokenized passage and a question as input, then returns a segment of the passage that most likely answers the question. We originally tried running a larger model (normal BERT fine-tuned on SQuAD) on the frontend but found that it was far too slow and thus settled upon MobileBERT. Model performance was assessed on SQuaD 2.0.

BERT-Large [2]: Similar to above, this is a pre-trained BERT-large model fine-tuned on SQuaD 2.0 to make it optimized for question and answering. We settled upon BERT-large because while the backend model had to be powerful, it still needed to fit within our computational constraints (namely, a Tesla K80 GPU) and maintain fast inference speed. We tried other models (also fine-tuned on SQuaD) such as XLNet but found that BERT had the best performance while still meeting the inference threshold. Model performance was assessed on SQuaD 2.0 and takes in the same input/output as the MobileBERT above.

Quora-DistilBERT [3]: This is a universal sentence transformer which generates any length sentence into an embedding of size 768. The model is a Sentence BERT model (SBERT: Siamese BERT) which was first trained on NLI+STSb data then was fine-tuned on the Quora Duplicate Questions dataset. This model was chosen because it generates relatively small embeddings with size 768 which makes storing in the embedding database easier and also allows for faster nearest neighbor search. Also, this model had previously been shown to get optimal performance on the Quora dataset which directly deals with finding similar semantic questions. Thus, both computationally and performance-wise it met the requirements.

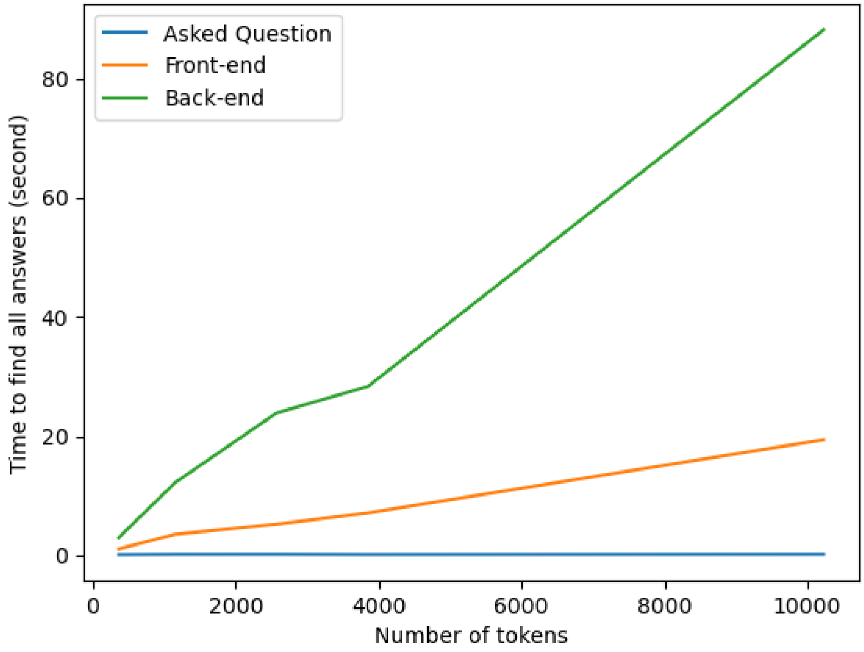
System Evaluation

System-level evaluation

Latency: For the previously asked question, the system only needs to send the question to the database and get the answer. Its latency mainly depends on the time it takes to highlight all the answers on the webpage, which depends on the structure of the webpage and the number of answers. But typically, it takes less than 200ms to finish all the highlighting.

For the front-end, we measure how much time it takes to find all the answers. As we are using batch processing mentioned above, which highlights the answer batch by batch, this measurement is pessimistic. By taking a total of five measurements on various token lengths, its average speed is about 2.34ms/token.

For the back-end, we use the same metric. To reach the backend, the search needs to be completed at the front-end first. Similar to the above setting, the average speed for the backend is about 8.79ms/token. This means that for a webpage with 10K tokens, the user has to wait 88 seconds to get the answer if the search needs to be done at the back-end. The figure below shows the time to get all answers vs tokens.



Overall Performance on Different Questions: For a simple question like "What is BERT", this system finds the location of the answer accurately. Based on our rough evaluation, the system achieved ~75% satisfactory answers (21/28 queries). However, for very vague questions, such as "What does BERT do?", many irrelevant answers were given, such as "BERT won the Best Long Paper Award" is also one of the answers. We also tried 20 questions across websites with different specialized domain knowledge such as Yahoo news, Wikipedia, blog posts, etc., and found that for only 7 of them the system does not give satisfactory answers (30% error rate on difficult domains). For very difficult queries, the accuracy is low as the frontend model has high confidence rate on wrong answers while the larger backend model is able to give better answers but doesn't get routed to due to the front-end model answering. For example, from Chip's blog "7 reasons not to join a startup and 1 reason to", a question such as "When do I join a startup?" will get an answer as "early-stage", which is not correct.

Instance-level evaluation

To evaluate the individual front-end model's sensitivity to input, we evaluate typos with 18 different questions on different webpages which the MobileBERT has the corresponding answers without typos. The typo includes adding, replacing, and removing a character. The MobileBERT fails to find the answer for eight of them, gives the same answer as before for five of them, and achieves an F1 score of 36.1. (The F1 score here measures the number of words that are the same, which is the same as SQuAD 2.0 metric.) The low score on F1 indicates that the system is vulnerable to a typo. Autocomplete or typo correction would be needed in the future. Here are some examples of typos we test:

Question 1: What is BERT?

Answer: *Bidirectional Encoder Representations from Transformers (BERT) is a Transformer-based machine learning technique for natural language processing (NLP) pre-training developed by Google*

Question 1 with Typo: Wat is BERT?

Answer: *BERT is a deeply bidirectional, unsupervised language representation, pre-trained using only a plain text corpus.*

The first answer is more accurate. However, for most of the failed cases, the problem is that the system cannot find the answer. When finding similar question queries, typos also affect the performance:

Question 2: What is transformer?

Similar Question: *what are the uses of a transformer? : natural language processing*

Typo Question: What is tranformer?

Similar question: *how are demographics collected? : A census*

This type of error highly depends on the keyword. For the above example, the key word is *transformer*, so once there is a typo, it cannot find questions related to *transformer*. This is an area of improvement with easy wins that we can easily fix with a typo corrector running in the frontend.

Broader Impacts & Limitations

Lightweight, semantic, command-F has the potential to dramatically increase efficiency in manually reading long, complex documents. Rather than spending the time to go through every sentence in detail, which can take time, our application leverages state-of-the-art language models combined with a simple UI and backend infrastructure optimized for low latency to provide an experience to quickly identify answers to questions on a page. Why rely on generic keyword matching when you can have a system that synthesizes and answers your questions right away?

While our application actually is broadly applicable, useful, and usable by anyone who searches the internet and uses Command-F, we believe that there is a sect of users who have the most to leverage from this tool. Our target users are people who spend time working with complicated, long and dense documents. For example, academics working on research papers would benefit from the ability to quickly synthesize primary and secondary sources in order to identify relevant aspects of literature that might inform their work. Their current workflows entail going through large amounts of information, but with semantic cmd-F, they can quickly ask the questions they need on a document and get an answer right away.

Another example would be lawyers, who constantly work with a large corpus of information across legal cases, due diligence, and other processes. Like researchers, lawyers have to go through these documents manually, oftentimes relying on paralegals and scribes to sift through and find relevant information. What we've heard from a few conversations is that this creates a disconnect between what the lawyer is doing and the documents they need to go through. With our application directly built into the browser, lawyers can easily ask legal questions related to their work and get delivered the right answers.

Of course, our application does not come without limitations. First, while there are a number of use cases for academics, lawyers and many others that would benefit from this tool, we're leveraging off-the-shelf language models that have been fine-tuned for general purpose question and answering, but haven't been fine-tuned to a particular domain. As a result, many of the answers to user queries may not be directly relevant for specific industries in our current version. However, one area for further exploration is focusing on one specific user group and building a tailored model with data from that domain (for example, building our semantic cmd-F application just for lawyers). To keep our application more generalizable, we could also leverage active learning by observing where a user scrolls to on the page if an answer was not relevant, and use the information from that page to continuously fine-tune the models.

Second, our application is currently built on a centralized database of queries, their corresponding embeddings, and the answers to those queries. One of the tradeoffs we face is how we leverage the search data from different users in order to show similar queries, which ultimately improves the experience for others. Storing and making use of that data across all of our users may create a privacy concern, but at the same time, our query similarity feature wouldn't be as useful if there isn't enough data. While we didn't account for this issue directly in our MVP of semantic cmd-F, one way we would solve this is only storing queries for topics that the user is most interested in, so we can deliver a better user experience for those areas without needing to collect all queries.

Reflection

Overall, we loved working together on this project and excited to continue tweaking the interface for more people to try it out. We split up implementation by feature, and each contributed to the ideation, development, and iteration of the final product equally. We came in with an idea to build a better version of cmd-F that enabled users to go beyond keyword matching and actually query for answers. In the process of building, we faced the challenges of optimizing for low latency while also building out a robust backend ML system that works seamlessly with a frontend, but we're proud of what we came up with!

In terms of our application and system design, making the decision of using a chrome extension was important because it grounded us in what the user experience would be like, which ultimately led to features like showing the top questions and related queries directly on the front end. In terms of technology, the MobileBERT significantly reduces the latency and computation power. It not only enhances the user experience, but also allows us to keep this chrome extension running at a much lower cost.

If we had more time, we want to invest more time on the front end and make it more appealing to the users, such as improving the design, adding colors, and making the interface more intuitive through keyboard shortcuts (rather than clicking on the extension) . Moreover, based on our evaluation result, the system is still prone to typos. So we hope to build an autocomplete system at the front-end to reduce the number of typos. We would also like to train our model with more data and make the result more accurate, as our model currently gives many irrelevant answers.

In addition, we hope to further lower the latency for complex queries on long and dense documents. We want to use heuristics to quickly identify and process only the sections that potentially have the answers. Doing so would dramatically improve the latency and get as close as possible to the near-instant feeling of cmd-F that we are trying to replicate. We also want to incorporate typo correction as our evaluation highlighted that as an easy win where we could remove several errors in our system and increase robustness. Lastly, we want to add a feature where users can indicate an answer is wrong and then can highlight the correct answer in the webpage. We can then gather these Q&A pairs and fine-tune our models in real-time to constantly improve based on user feedback. In the future, we hope to publish this chrome extension on the web store for public use!

References

[1] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, & Denny Zhou. (2020). MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, & Kristina Toutanova. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

[3] Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

[4] https://pymilvus.readthedocs.io/en/latest/tutorial.html

[5] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," 2019

[6] https://developer.chrome.com/docs/extensions/mv3/getstarted/

[7] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. arXiv preprint arXiv:1806.03822, 2018.

[8] https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs