Document-Based Question And Answering With Hierarchical Search Using Fine-Tuned AlBERT Models

Catalog:

- **Project Scope**
- **Application Demonstration**
- **Project Architecture**
- **Models: Albert-Based Models**
    - Twin-Albert for long-text document searching
    - AlBert for sentence searching
    - AlBert for answer searching
- **Limitations and Future works**

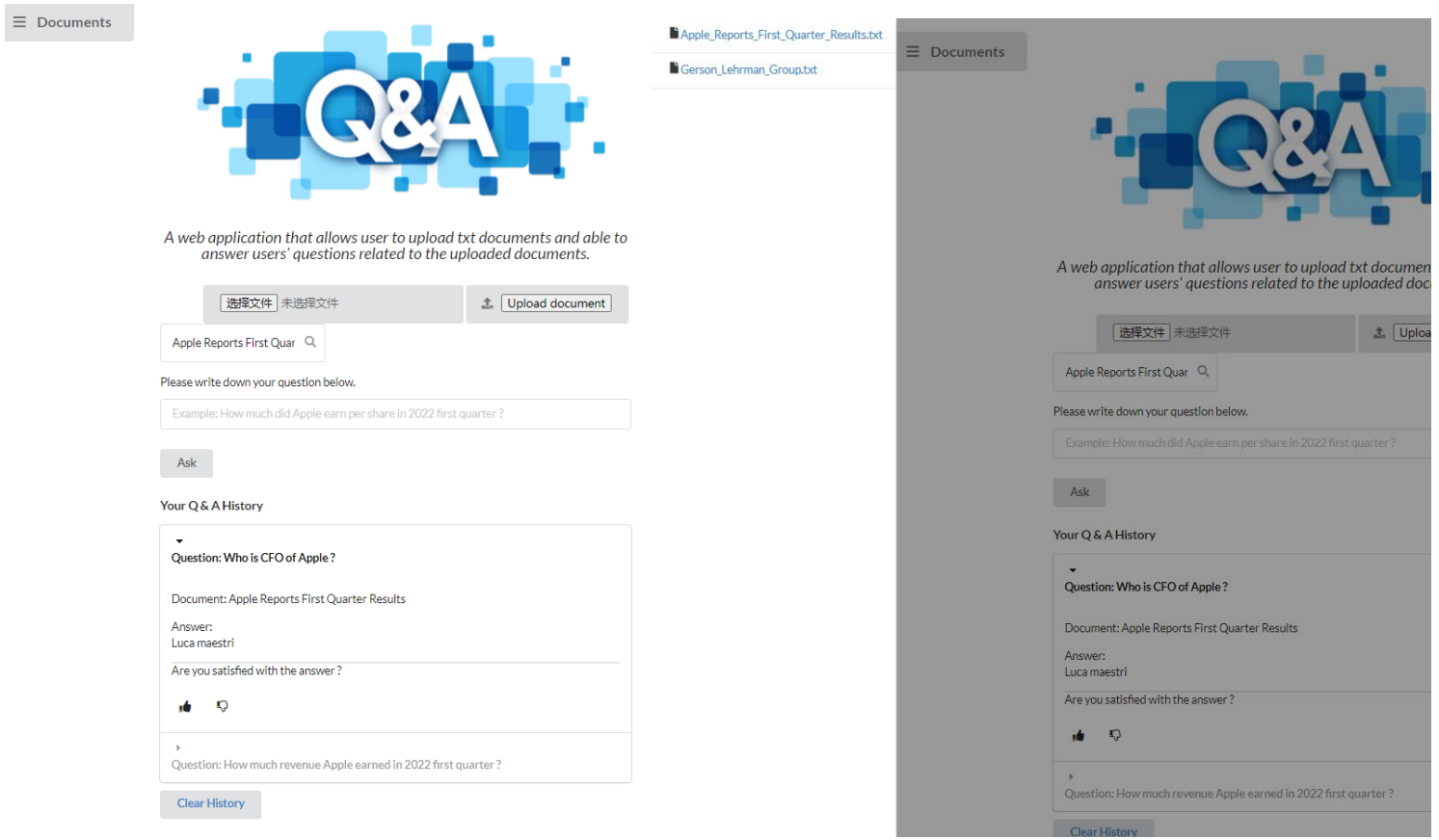
Author: Nelson LIN
Email: nelsonlin0321@outlook.com

Project Scope

Business Challenge:
How to leverage tone of unstructured document and bring significant value to users ?

Technical Solution:
An application that' able to ingest unstructured documents as knowledge corpus and answer questions related these document.

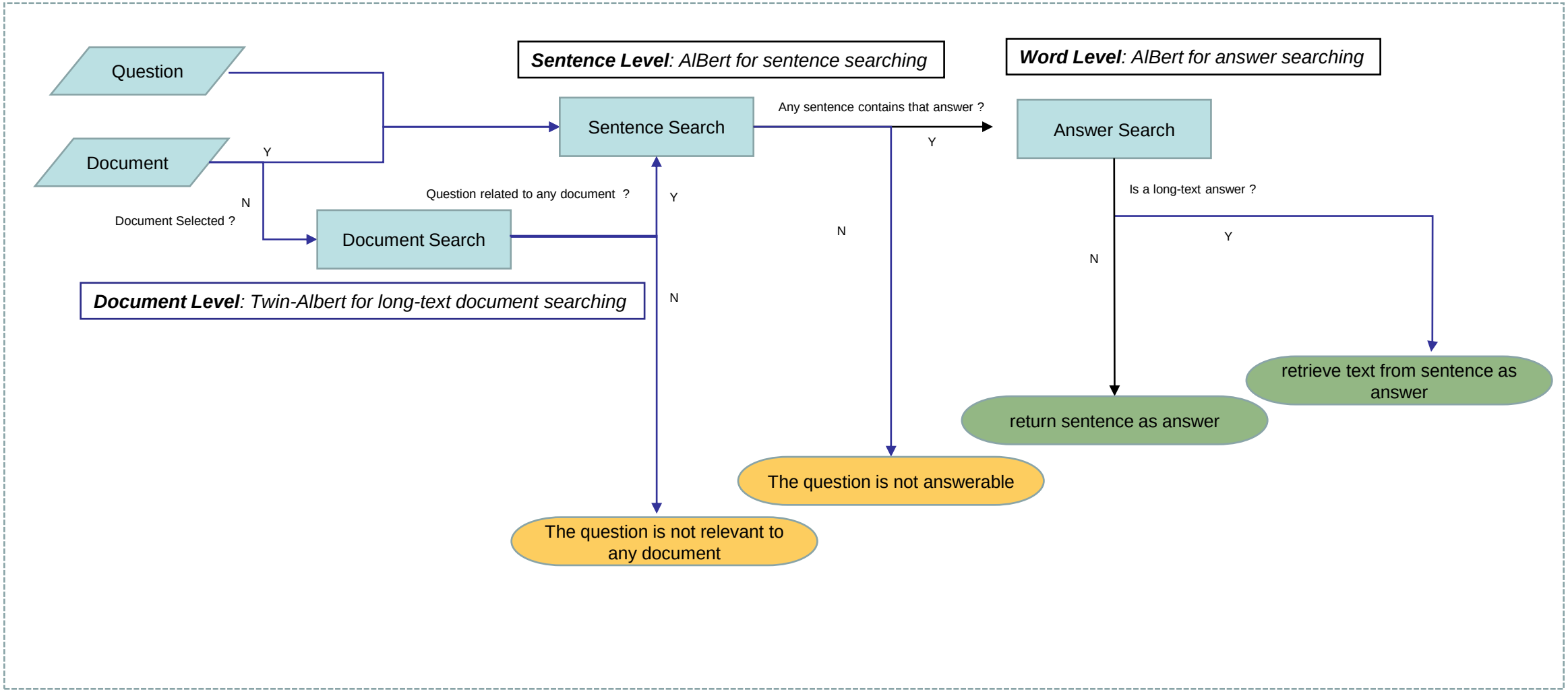
Application Demonstration:



Tech Stack:
Frontend: jQuery, Semantic UI(Semantic UI (semantic-ui.com))
Backend: Flask, Jinja
Modeling: PyTorch, transformers, ALBERT
Data: QNLI v2(QNLI Dataset | Papers With Code) and SQUAD v2 (The Stanford Question Answering Dataset (rajpurkar.github.io))

Project Source Codes:
Nelsonlin0321/DocumentQuestionAnswering (github.com)

Project Architecture:



Models: Albert-Based Models

AlBERT: A Lite BERT (ALBERT) architecture that has significantly fewer parameters than a traditional BERT architecture.

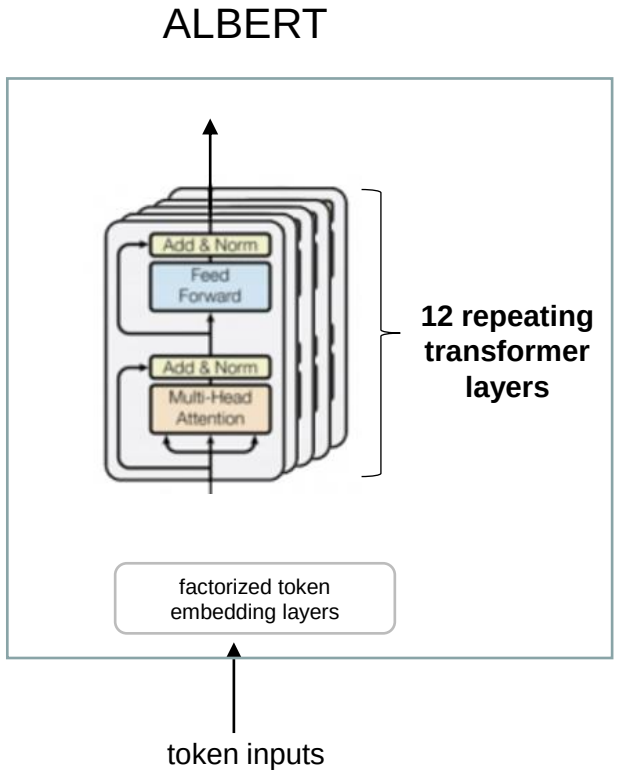
Keys features:

Parameter reduction techniques:

1) Factorized word embedding parameterization: Reduce number of vocabulary parameter.
2) Cross-layer parameter sharing: 18x fewer parameters and can be trained about 1.7x faster than BERT Large

Pretrained Task:

1) Introduce a self-supervised loss for sentence-order prediction (SOP) instead NSP: Avoids topic prediction and instead focuses on modeling inter-sentence coherence.
2) Mask 3-gram for masked word prediction rather than individual ones : Improving pre-training by representing and predicting spans.

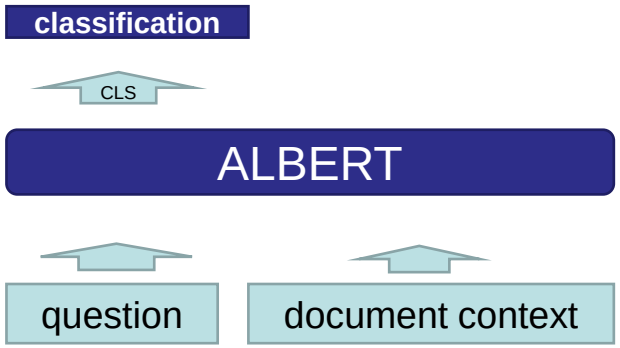


ALBERT: A Lite BERT for Self-supervised Learning of Language Representations: [1909.11942] ALBERT: A Lite BERT for Self-supervised Learning of Language Representations (arxiv.org)

Twins-Alberts for long-text document searching

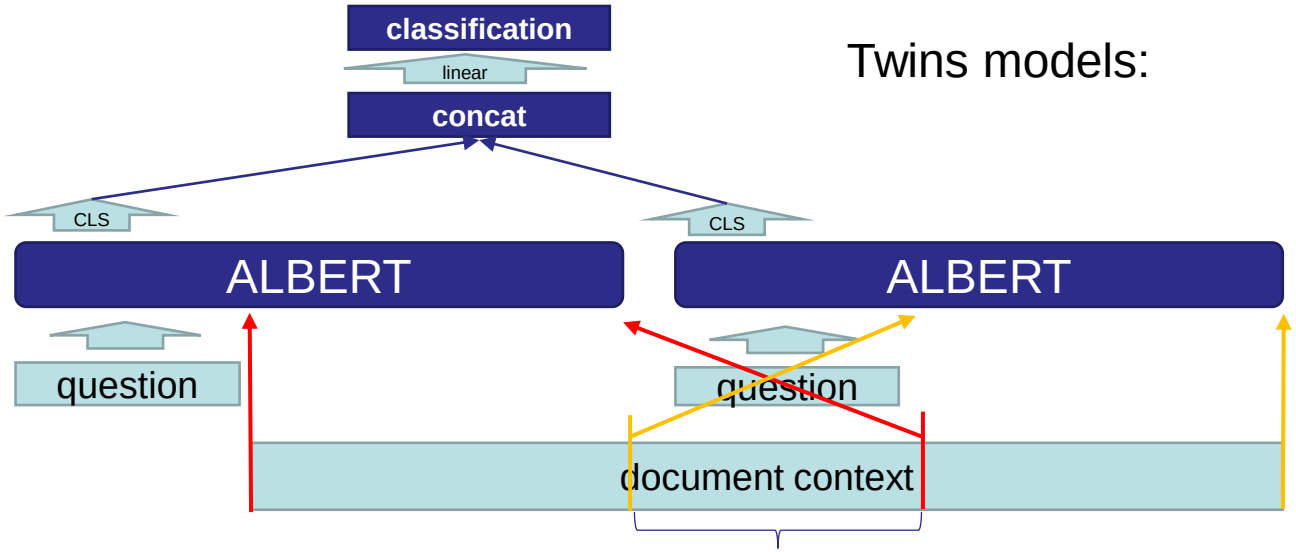
Purpose: to find any document relevant to the query.

Traditional single model:



Limitations:
1) the model can handle only 512 token length
2) loss information of tail of document

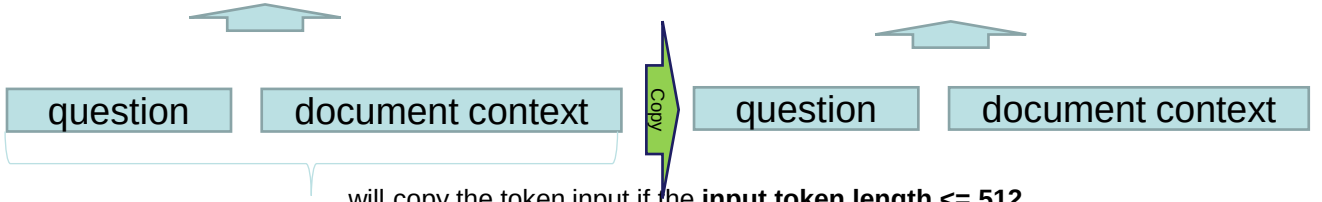
Twins models:



document overlap if the **input token length > 512**
otherwise the document. Overlapping length = 32 etc.

Document overlapping: knowledge sharing to stabilize twins models and make sure the answer exists in at least one input document.
Copying the same token input if length <=512: Avoid the right model receiving padding noise to make it similar to the left one.

The number of token more can be feed into model is **512 – question token length – 3 (due to special token)**



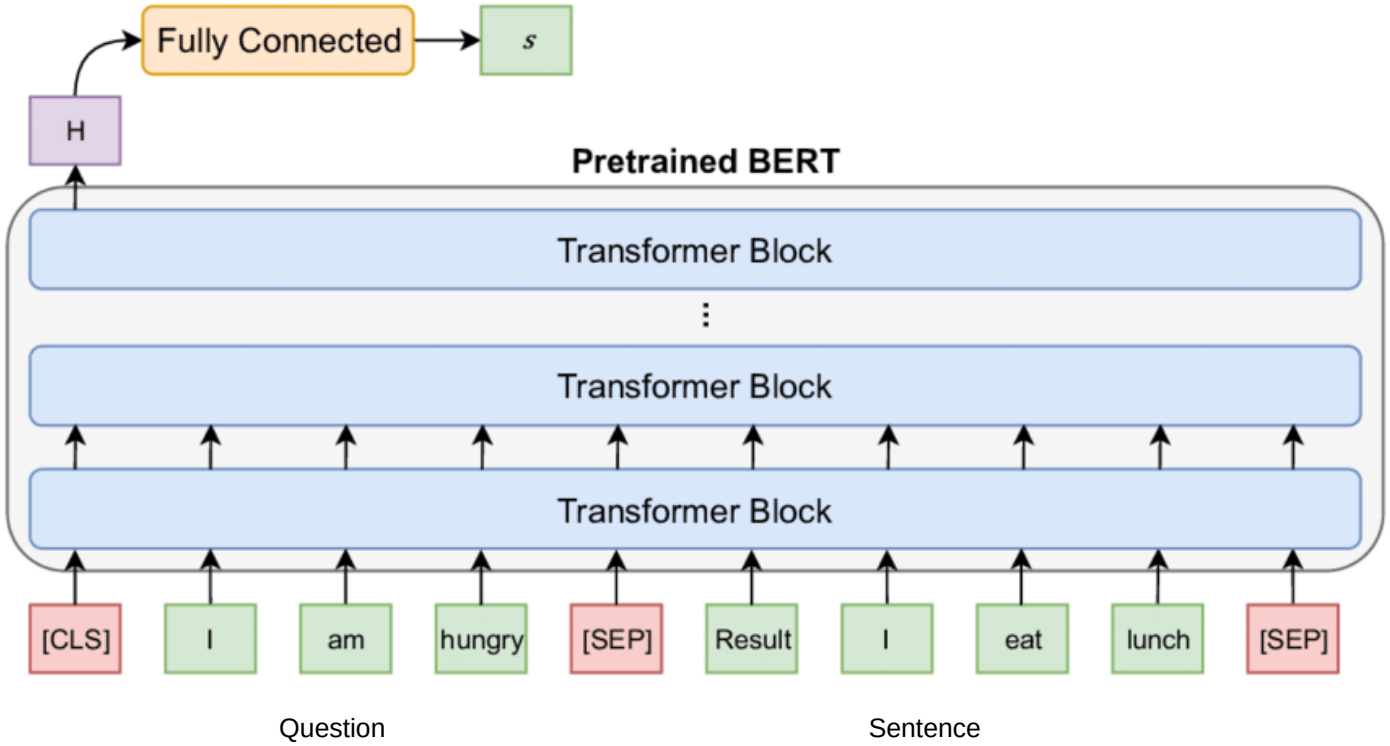
will copy the token input if the **input token length <= 512**

Dev Set created from SQUAD2 to predict if the question is relevant to the document:

{'Accuracy': 0.991, 'Recall': 0.990, 'Precision': 0.993, 'F1': 0.991, 'AUC': 0.999, 'Loss': 0.0277}

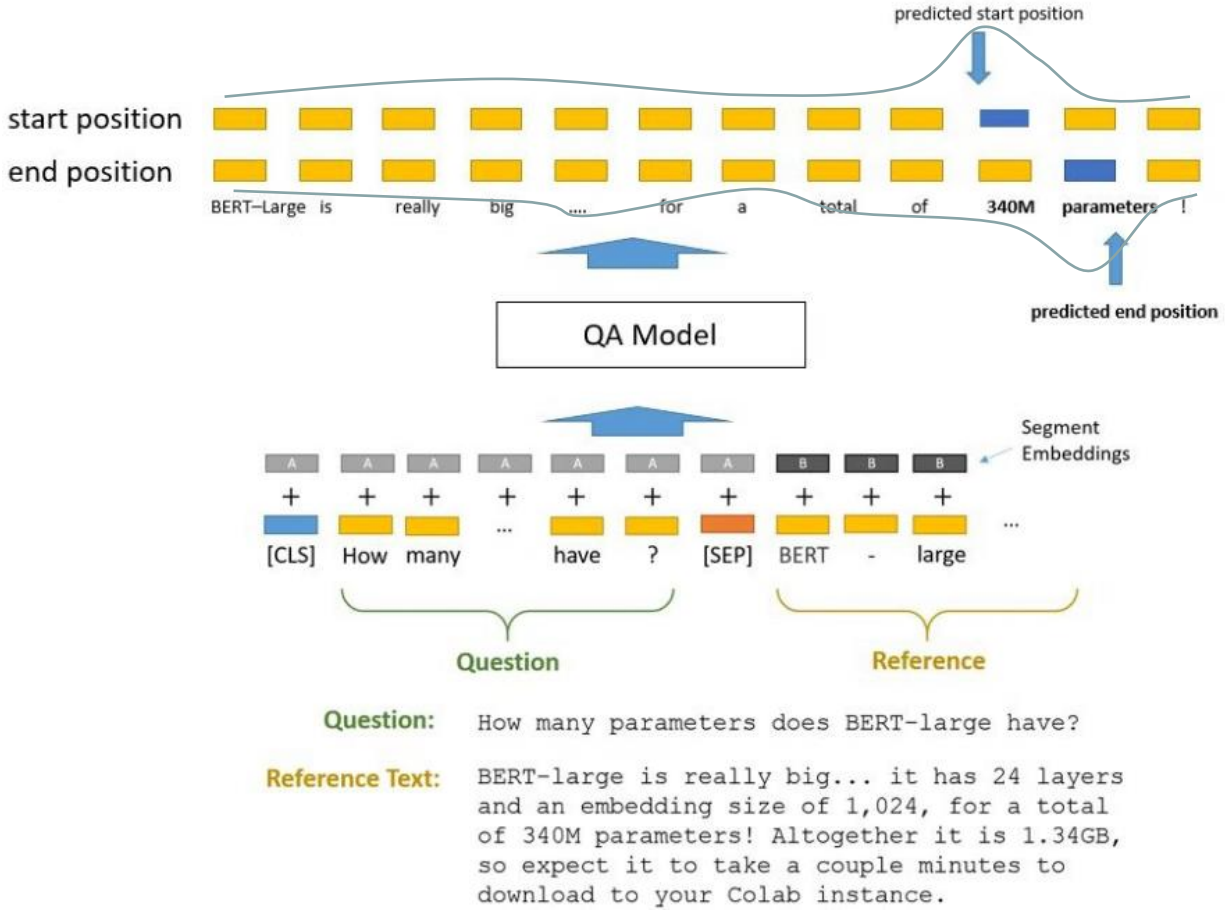
AlBert for sentence searching

Purpose: to find any sentence that contains question's answer.



AlBert for answer searching

Purpose: to extract answer from a sentence that contains question's answer.



The fine-tuned model outputs are the two vectors that distribute the primality of start index and end index to locate the answer.

SQUAD2 Dev Set to extract answer from context

{'Exact': 78.657,'F1', 81.936)

Limitations and Future Works

Limitations:

1) Only txt document is allowed to upload.
2) Flask may not best for DL deployment and models serving.
3) Users share the same question and answering history page. They can see the Q&A from others.
4) Models are good at open domain questions but may not good at others.

Future Works:

1) Develop parsing data pipeline to ingest other types of documents, like PDF, words.
2) Deployed models using torchserve for large scale model serving inference.
3) Develop user Registration and authentication to serve multiple users individually.
4) Deploy the project on Cloud to make it more accessible.
5) Continuously collect feedback from user to make model continuously learn.
6) Develop a labeling system to collect more training data to make model goods at a specific domain.

Thanks