

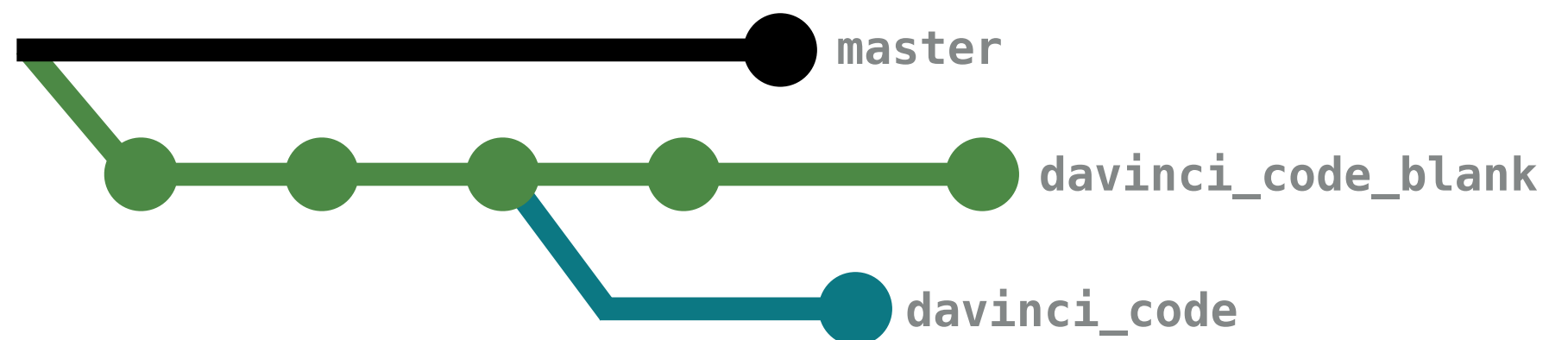
ADAM SPANNBAUER

NATURAL LANGUAGE Q&A SEARCH ENGINE

MATERIALS

All data & source code used can be found at:

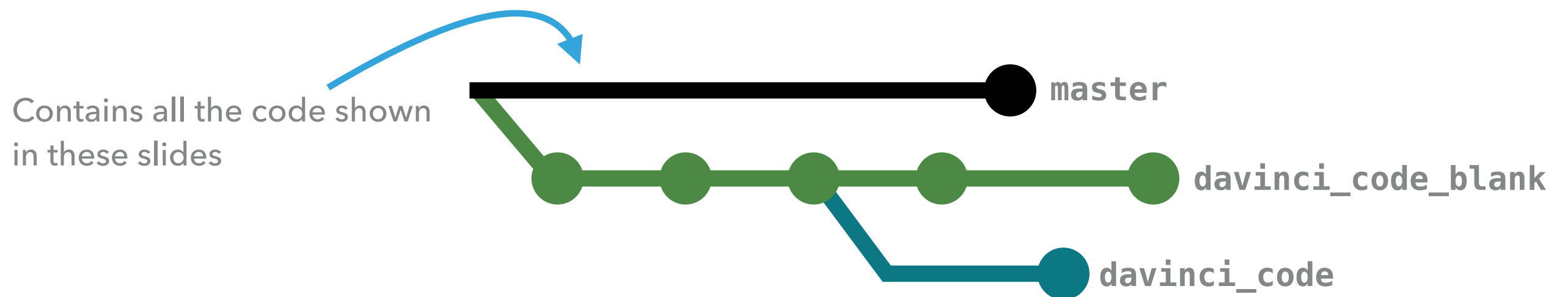
https://github.com/AdamSpannbauer/qa_query



MATERIALS

All data & source code used can be found at:

https://github.com/AdamSpannbauer/qa_query

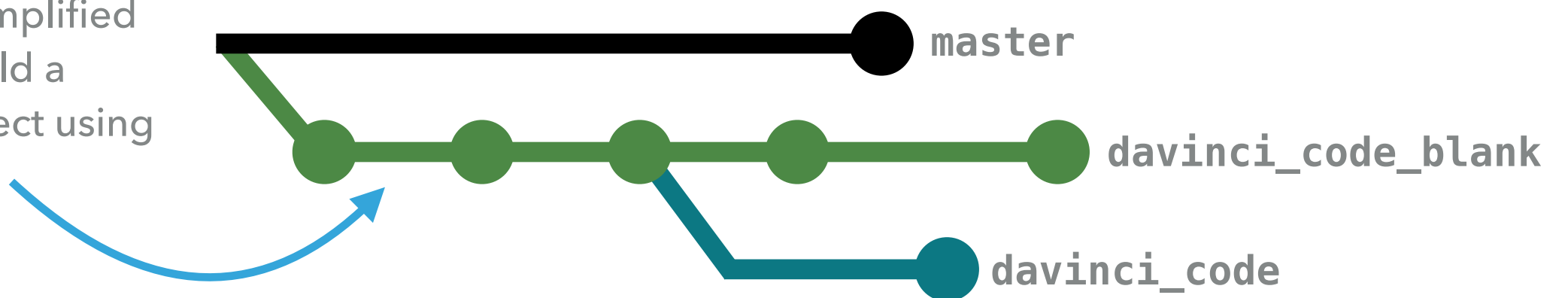


MATERIALS

All data & source code used can be found at:

https://github.com/AdamSpannbauer/qa_query

Contains data & simplified
file structure to build a
version of the project using
The Davinci Code

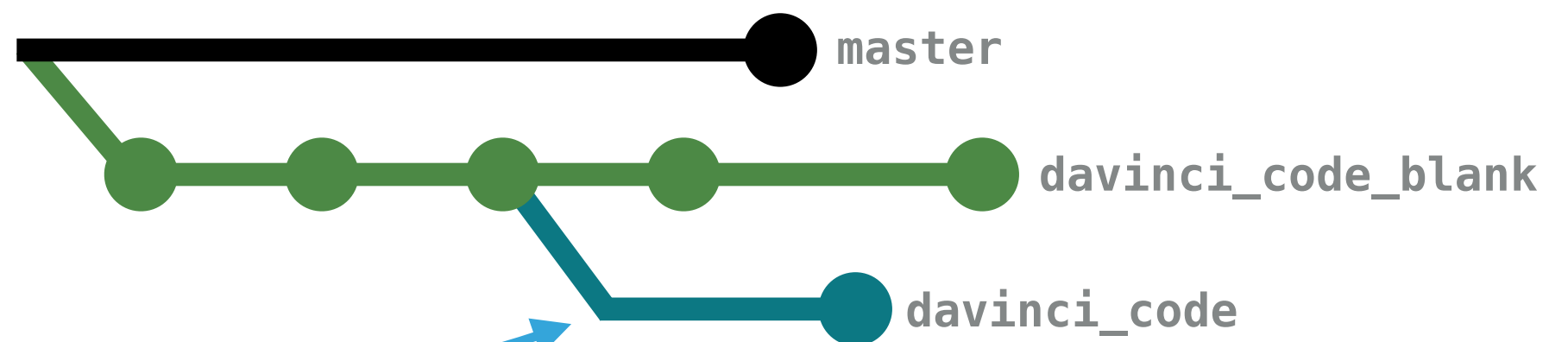


MATERIALS

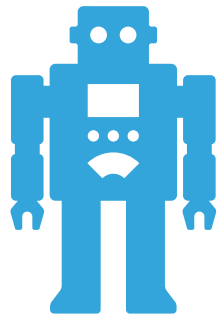
All data & source code used can be found at:

https://github.com/AdamSpannbauer/qa_query

An "answer key" for `davinci_code_blank`. Shows how a completed *The Davinci Code* project could look.



GOAL OF PROJECT

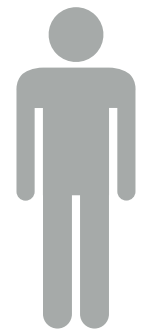


WHAT MARKET DOES FITBIT
COMPETE IN?

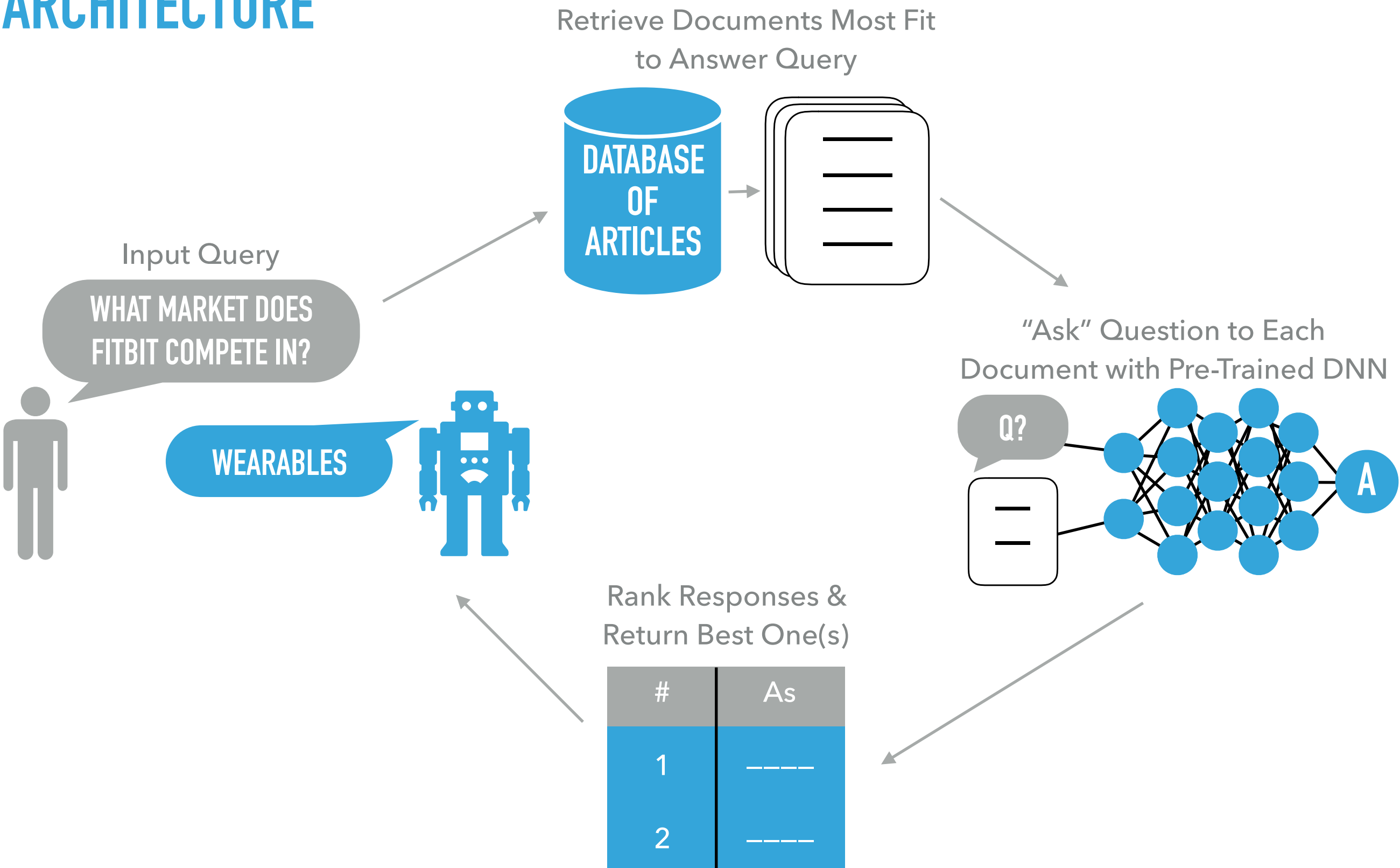
WEARABLES

WHO IS THE CEO OF APPLE?

TIM COOK

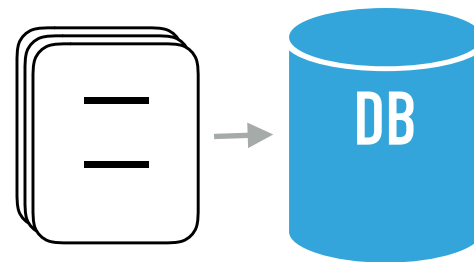


ARCHITECTURE



STEP 0

Populate Database with
Articles to Search



This won't be covered in the scope of this session.

Articles were scraped from [Nasdaq](#); for simplicity these articles were stored as JSON files in [this project's repo](#).

STEP 1 – OVERVIEW



This problem is nothing new and is known as Information Retrieval (IR)

*Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).**

*[Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008](#)

STEP 1 – TOOL SELECTION



Whenever a problem is nothing new, a new solution might not be best solution for the problem.

Some possible solutions for performing IR (not exhaustive):

- ▶ Elasticsearch
- ▶ MongoDB Text Index
- ▶ Whoosh*

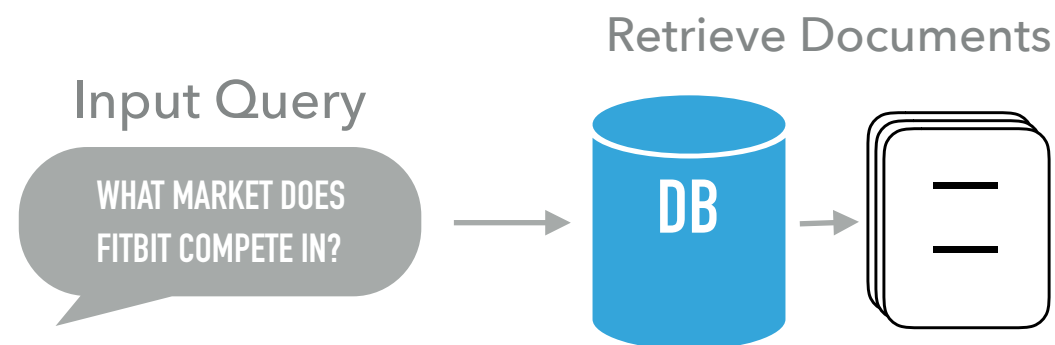
*pure Python and will be our solution today; this was chosen mostly based on simplicity

STEP 1 – GETTING STARTED WITH WHOOSH



[Whoosh](#) was created by Matt Chaput. It started as a quick and dirty search server for the online documentation of the Houdini 3D animation software package. Side Effects Software generously allowed Matt to open source the code in case it might be useful to anyone else who needs a very flexible or pure-Python search engine (or both!).

STEP 1 – HOW WHOOSH WORKS



Whoosh uses [Okapi BM25](#), to *estimate* and rank each document's relevance to an input query. The term estimate is used in this definition because BM25 is a probabilistic approach to scoring documents.

*A relevance score, according to probabilistic information retrieval, ought to reflect the probability a user will consider the result relevant.**

*[Doug Turnbull BM25 The Next Generation of Lucene Relevance](#)

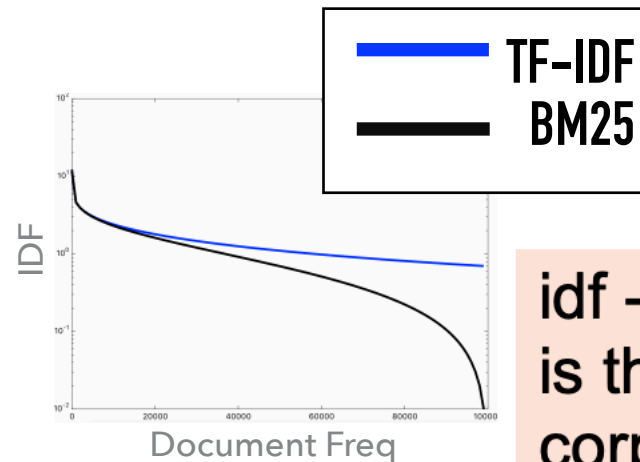
STEP 1 – OKAPI BM25 OVERVIEW



Main Components of Interest:

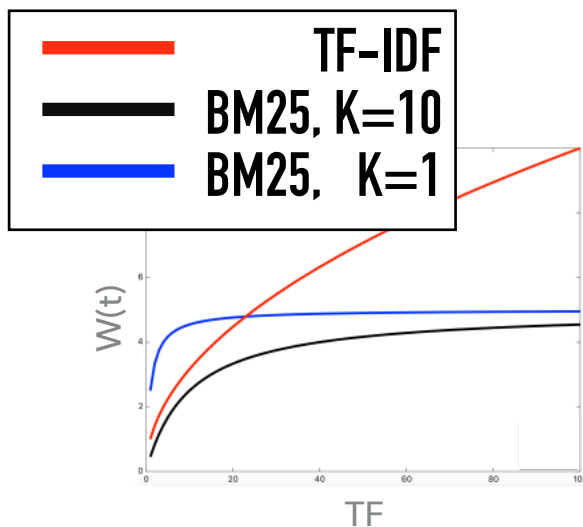
- ▶ Term Frequency (TF) - Words mentioned a lot in a document are more important
- ▶ Inverse Document Frequency (IDF) - Words mentioned too often in a corpus are less important (i.e. "the", "a", etc.)
- ▶ Document Length - A word appearing 20 times in a book isn't as meaningful as a word appearing 20 times in a tweet

STEP 1 - OKAPI BM25 MATH



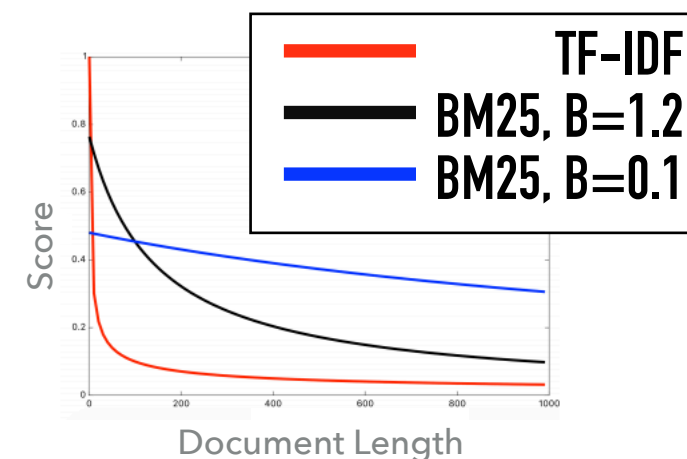
idf - how popular is the term in the corpus?

$$\text{bm25}(d) = \sum_{t \in q, f_{t,d} > 0} \log \left(1 + \frac{N - df_t + 0.5}{df_t + 0.5} \right) \cdot \frac{f_{t,d}}{f_{t,d} + k} \cdot \left(1 - b + b \frac{l(d)}{\text{avgdl}} \right)$$

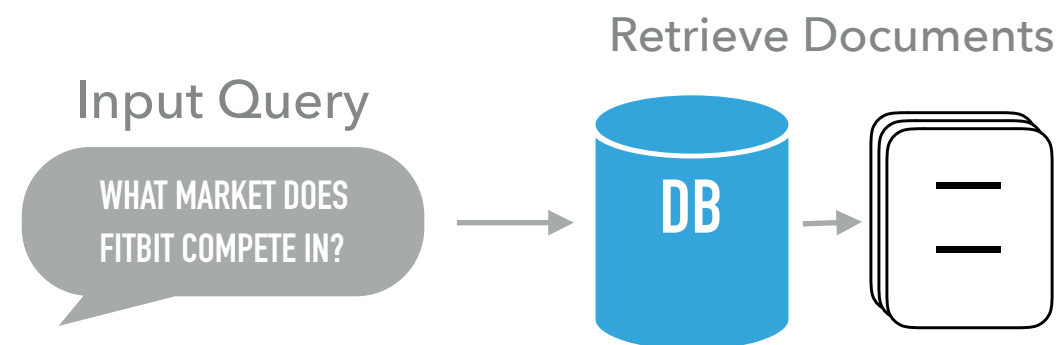


saturation curve - limit influence of tf on the score

length weighing - tweak influence of document length

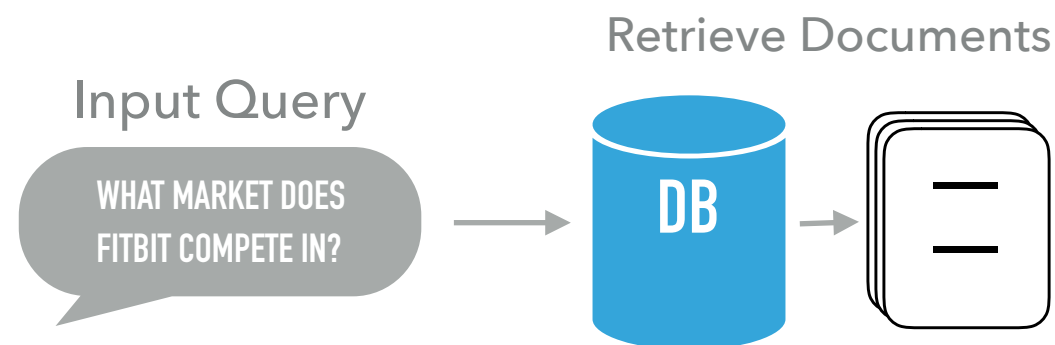


STEP 1 – OKAPI BM25 PRACTICAL APPLICATION KNOWLEDGE



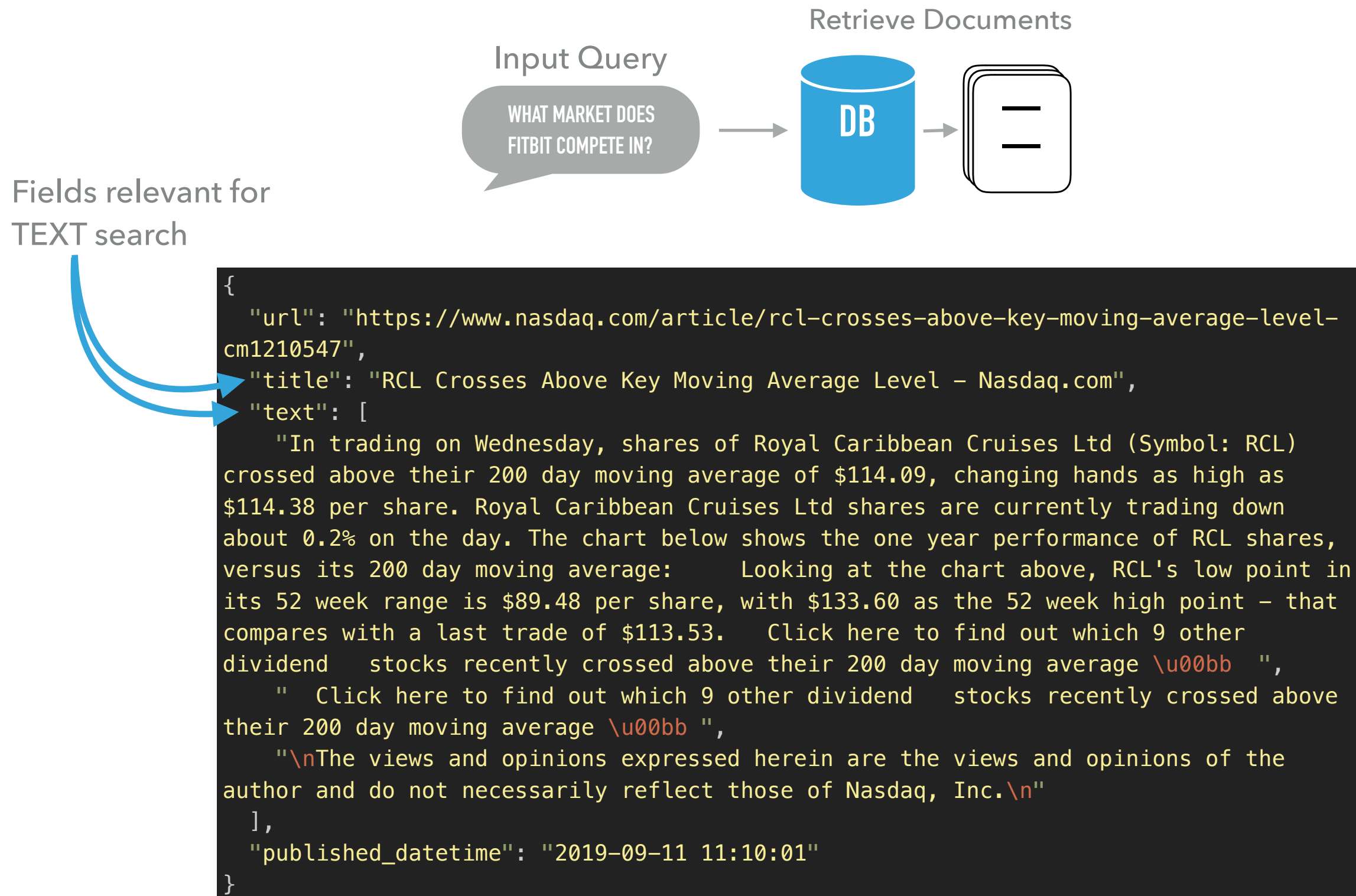
- ▶ Works best for smaller documents; think blog post-ish size and smaller
- ▶ Already implemented for us in Whoosh and other tools (i.e. Elasticsearch)

STEP 1 – WHOOSH DESIGN

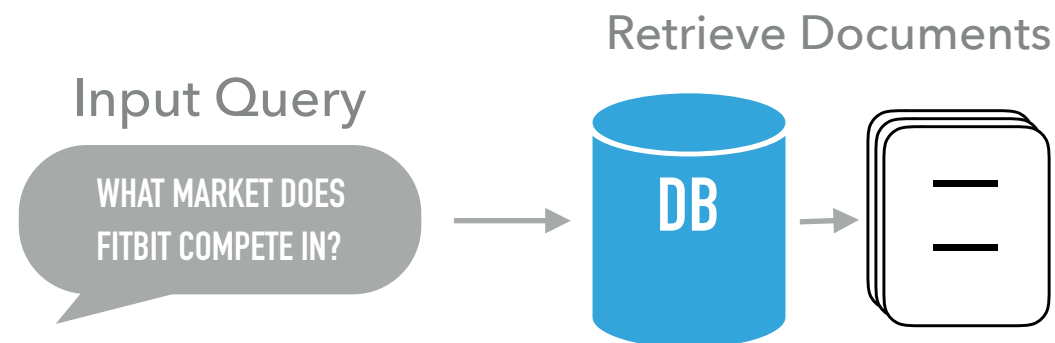


```
{
  "url": "https://www.nasdaq.com/article/rcl-crosses-above-key-moving-average-level-
cm1210547",
  "title": "RCL Crosses Above Key Moving Average Level - Nasdaq.com",
  "text": [
    "In trading on Wednesday, shares of Royal Caribbean Cruises Ltd (Symbol: RCL)
crossed above their 200 day moving average of $114.09, changing hands as high as
$114.38 per share. Royal Caribbean Cruises Ltd shares are currently trading down
about 0.2% on the day. The chart below shows the one year performance of RCL shares,
versus its 200 day moving average: Looking at the chart above, RCL's low point in
its 52 week range is $89.48 per share, with $133.60 as the 52 week high point – that
compares with a last trade of $113.53. Click here to find out which 9 other
dividend stocks recently crossed above their 200 day moving average \u00bb ",
    " Click here to find out which 9 other dividend stocks recently crossed above
their 200 day moving average \u00bb ",
    "\n\nThe views and opinions expressed herein are the views and opinions of the
author and do not necessarily reflect those of Nasdaq, Inc.\n\n",
  ],
  "published_datetime": "2019-09-11 11:10:01"
}
```


STEP 1 – WHOOSH DESIGN



STEP 1 – WHOOSH DESIGN



```
{
  "url": "https://www.nasdaq.com/article/rcl-crosses-above-key-moving-average-level-
cm1210547",
  "title": "RCL Crosses Above Key Moving Average Level - Nasdaq.com",
  "text": [
    "In trading on Wednesday, shares of Royal Caribbean Cruises Ltd (Symbol: RCL)
crossed above their 200 day moving average of $114.09, changing hands as high as
$114.38 per share. Royal Caribbean Cruises Ltd shares are currently trading down
about 0.2% on the day. The chart below shows the one year performance of RCL shares,
versus its 200 day moving average: Looking at the chart above, RCL's low point in
its 52 week range is $89.48 per share, with $133.60 as the 52 week high point – that
compares with a last trade of $113.53. Click here to find out which 9 other
dividend stocks recently crossed above their 200 day moving average \u00bb ",
    " Click here to find out which 9 other dividend stocks recently crossed above
their 200 day moving average \u00bb ",
    "\n\nThe views and opinions expressed herein are the views and opinions of the
author and do not necessarily reflect those of Nasdaq, Inc.\n"
  ],
  "published_datetime": "2019-09-11 11:10:01"
}
```

ID fields

STEP 1 – WHOOSH SCHEMA



```
from whoosh.fields import Schema, TEXT, ID
from whoosh.analysis import StandardAnalyzer

schema = Schema(
    url=ID(stored=True, unique=True),
    published_datetime=ID(stored=True),
    title=TEXT(stored=True, analyzer=StandardAnalyzer()),
    article=TEXT(stored=True, analyzer=StandardAnalyzer()),
)
```

- Fields to be searched are defined with TEXT() & others are defined with ID().

STEP 1 – WHOOSH SCHEMA



```
from whoosh.fields import Schema, TEXT, ID
from whoosh.analysis import StandardAnalyzer

schema = Schema(
    url=ID(stored=True, unique=True),
    published_datetime=ID(stored=True),
    title=TEXT(stored=True, analyzer=StandardAnalyzer()),
    article=TEXT(stored=True, analyzer=StandardAnalyzer()),
)
```

- ▶ We can enforce a uniqueness constraint by setting `unique=True`

STEP 1 – WHOOSH SCHEMA



```
from whoosh.fields import Schema, TEXT, ID
from whoosh.analysis import StandardAnalyzer

schema = Schema(
    url=ID(stored=True, unique=True),
    published_datetime=ID(stored=True),
    title=TEXT(stored=True, analyzer=StandardAnalyzer()),
    article=TEXT(stored=True, analyzer=StandardAnalyzer()),
)
```

- Indicating `stored=True` means that the raw version of the field will be saved in the index. For text fields this can be costly since the post-processed text will be much smaller.

STEP 1 – WHOOSH SCHEMA

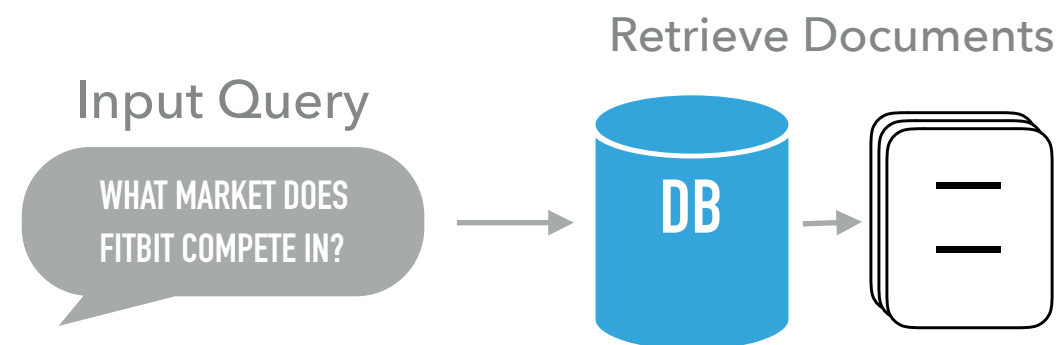


```
from whoosh.fields import Schema, TEXT, ID
from whoosh.analysis import StandardAnalyzer

schema = Schema(
    url=ID(stored=True, unique=True),
    published_datetime=ID(stored=True),
    title=TEXT(stored=True, analyzer=StandardAnalyzer()),
    article=TEXT(stored=True, analyzer=StandardAnalyzer()),
)
```

- ▶ Whoosh comes with some pre-defined “analyzers” that will dictate how stored documents and input queries will be processed.

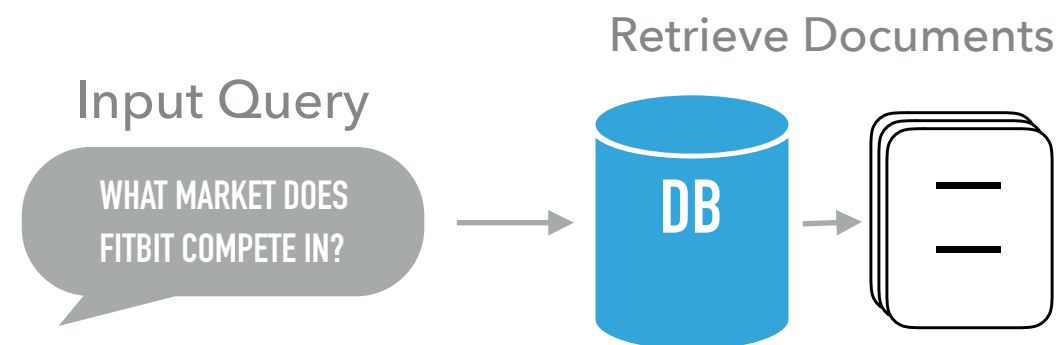
STEP 1 – WHOOSH ANALYZERS



```
>>> from whoosh.analysis import StandardAnalyzer, StemmingAnalyzer
>>>
>>> [token.text for token in StandardAnalyzer().("This is an example analysis")]
['example', 'analysis']
>>> [token.text for token in StemmingAnalyzer().("This is an example analysis")]
['exampl', 'analysi']
```

- ▶ The StandardAnalyzer performs tokenization, lowercasing, & stopwords removal by default
- ▶ The StemmingAnalyzer adds stemming with the [Porter Stemming Algorithm](#)

STEP 1 – BUILDING THE INDEX

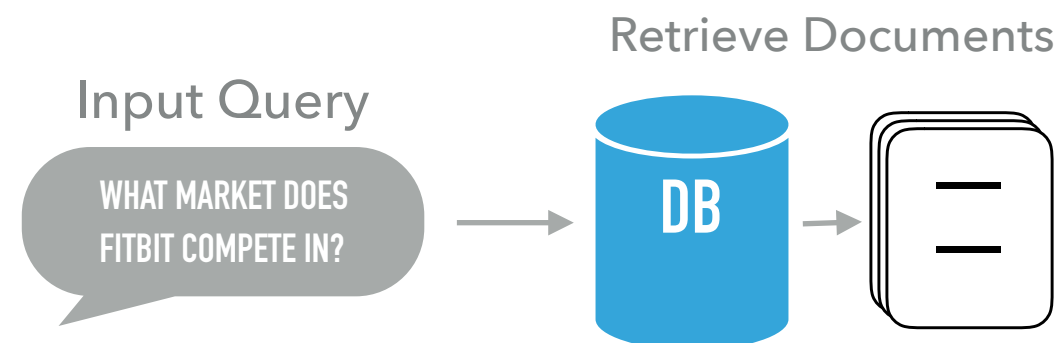


```
schema = Schema(
    url=ID(stored=True, unique=True),
    published_datetime=ID(stored=True),
    title=TEXT(stored=True, analyzer=StemmingAnalyzer()),
    article=TEXT(stored=True, analyzer=StemmingAnalyzer()),
)

idx = whoosh.index.create_in("index_directory", schema=schema, indexname="nasdaq")

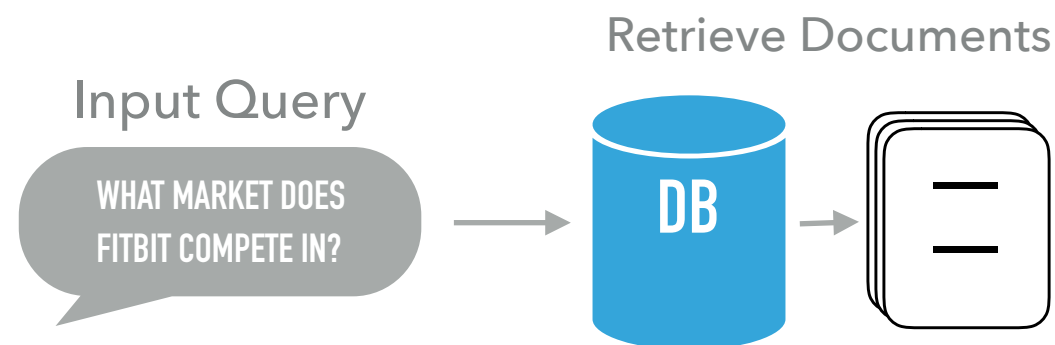
writer = idx.writer()
for i, row in text_df.iterrows():
    writer.update_document(
        url=row['url'],
        published_datetime=row['published_datetime'],
        title=row['title'],
        article=row['text'],
    )
writer.commit()
```


STEP 1 – QUERY PARSING



```
>>> import whoosh.index
... from whoosh.qparser import MultifieldParser, OrGroup, WildcardPlugin
...
... whoosh_idx = whoosh.index.open_dir('whoosh_idx', indexname='nasdaq')
... query_parser = MultifieldParser(['title', 'article'],
...                                 schema=whoosh_idx.schema,
...                                 group=OrGroup)
... query_parser.remove_plugin_class(WildcardPlugin)
...
... parsed_query = query_parser.parse('What market does FitBit compete in?')
>>> print(parsed_query)
Or([Term('title', 'what'), Term('article', 'what'), Term('title', 'market'),
Term('article', 'market'), Term('title', 'doe'), Term('article', 'doe'), Term('title',
'fitbit'), Term('article', 'fitbit'), Term('title', 'compet'), Term('article',
'compet')])
```

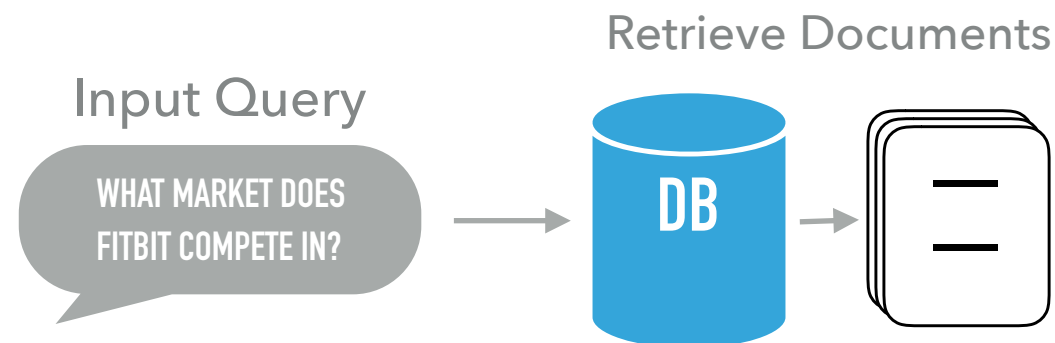
STEP 1 – QUERY PARSING



```
>>> import whoosh.index
... from whoosh.qparser import MultifieldParser, OrGroup, WildcardPlugin
...
... whoosh_idx = whoosh.index.open_dir('whoosh_idx', indexname='nasdaq')
... query_parser = MultifieldParser(['title', 'article'],
...                                 schema=whoosh_idx.schema,
...                                 group=OrGroup)
... query_parser.remove_plugin_class(WildcardPlugin)
...
... parsed_query = query_parser.parse('What market does FitBit compete in?')
>>> print(parsed_query)
Or([Term('title', 'what'), Term('article', 'what'), Term('title', 'market'),
Term('article', 'market'), Term('title', 'doe'), Term('article', 'doe'), Term('title',
'fitbit'), Term('article', 'fitbit'), Term('title', 'compet'), Term('article',
'compet')])
```

We're building a query parser specifically for questions. Maybe our stopwords list should be revisited?

STEP 1 – QUERY PARSING



```
>>> import whoosh.index
... from whoosh.qparser import MultifieldParser, OrGroup, WildcardPlugin
...
... whoosh_idx = whoosh.index.open_dir('whoosh_idx', indexname='nasdaq')
... query_parser = MultifieldParser(['title', 'article'],
...                                 schema=whoosh_idx.schema,
...                                 group=OrGroup)
... query_parser.remove_plugin_class(WildcardPlugin)
...
... parsed_query = query_parser.parse('What market does FitBit compete in?')
>>> print(parsed_query)
Or([Term('title', 'what'), Term('article', 'what'), Term('title', 'market'),
Term('article', 'market'), Term('title', 'doe'), Term('article', 'doe'), Term('title',
'fitbit'), Term('article', 'fitbit'), Term('title', 'compet'), Term('article',
'compet')])
```

Companies and other “Named Entities” are likely key to a user’s queries. Maybe we should give them more importance?

STEP 1 – PERFORMING A SEARCH



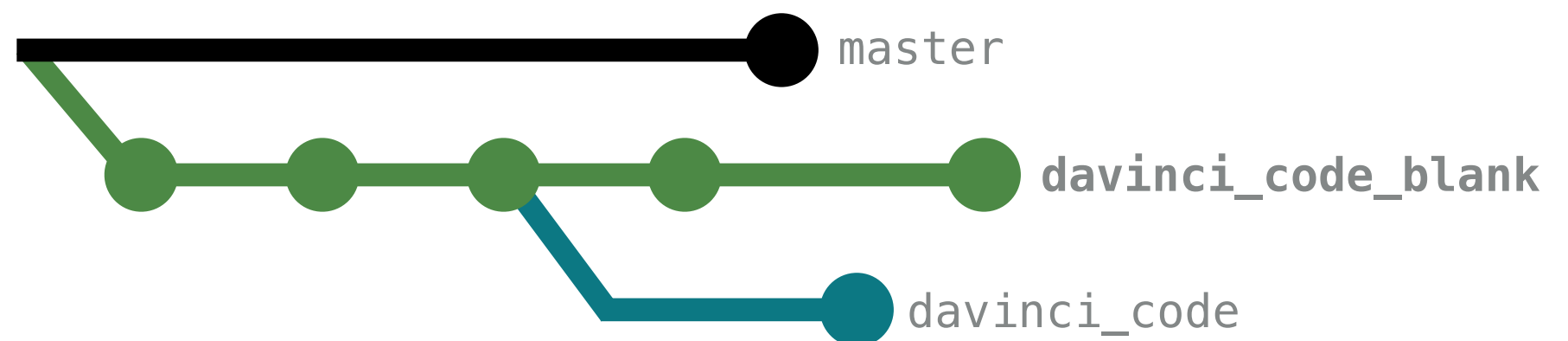
```
>>> import whoosh.index
... from whoosh.qparser import MultifieldParser, OrGroup, WildcardPlugin
...
... whoosh_idx = whoosh.index.open_dir('whoosh_idx', indexname='nasdaq')
... query_parser = MultifieldParser(['title', 'article'],
...                                 schema=whoosh_idx.schema,
...                                 group=OrGroup)
... query_parser.remove_plugin_class(WildcardPlugin)
...
... parsed_query = query_parser.parse('What market does FitBit compete in?')
...
... with whoosh_idx.searcher() as searcher:
...     search_results = searcher.search(parsed_query, limit=1)
...     [print(sr['title']) for sr in search_results]
```

Where Are Wearables and Smart Speakers in 2019? – Nasdaq.com

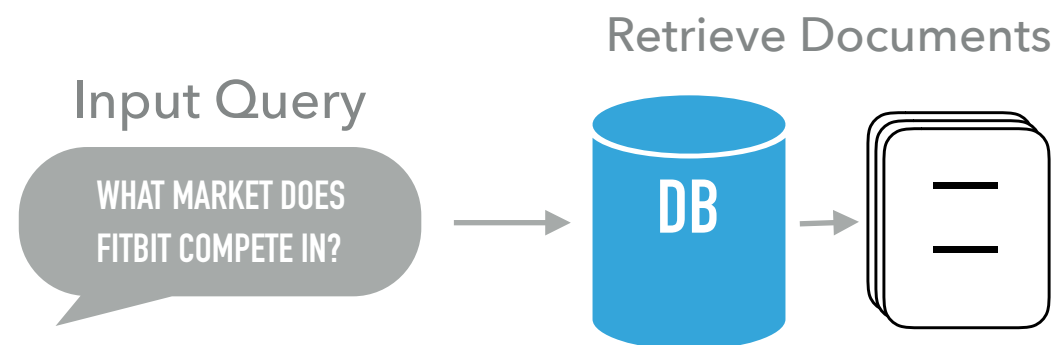
STEP 1 – DO IT YOURSELF



You have now seen enough to complete the files: [build_davinci_index.py](#) & [search_davinci_index.py](#) in the [davinci_code_blank](#) branch. Completed versions of these files can be seen in the [davinci_code](#) branch.



STEP 1 – CUSTOMIZING STOPWORDS



```
import nltk.corpus

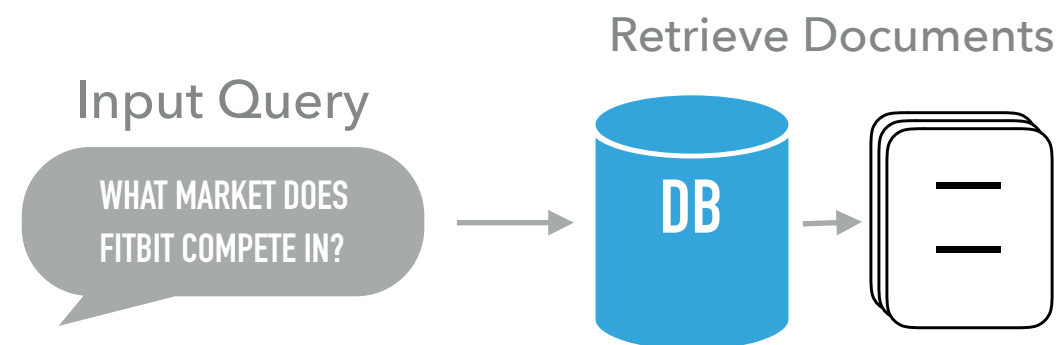
NLTK_STOPWORDS = set(nltk.corpus.stopwords.words('english'))
QUESTION_STOPWORDS = {'who', 'what', 'where', 'when', 'why', 'how'}
QA_STOPWORDS = frozenset(QUESTION_STOPWORDS | NLTK_STOPWORDS)
```

Whoosh's stopword list is fairly minimal, here [nltk](#) is used as a base list and augmented with a custom list of common question words we'd expect.

This new list can be used with our analyzer like so:

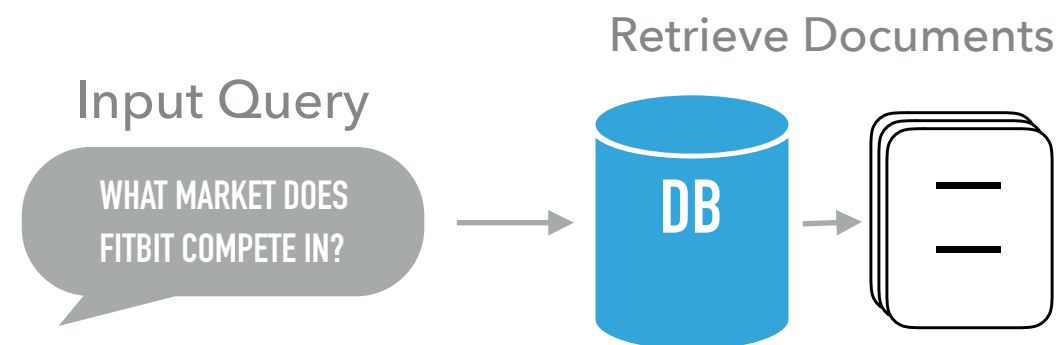
```
StemmingAnalyzer(stoplist=QA_STOPWORDS)
```

STEP 1 – NAMED ENTITY RECOGNITION



*Named Entity Recognition (NER) labels sequences of words in a text which are the names of things, such as person and company names...**

STEP 1 – NAMED ENTITY RECOGNITION



*Named Entity Recognition (NER) labels sequences of words in a text which are the names of things, such as person and company names...**

What market does FitBit compete in?

STEP 1 – NAMED ENTITY RECOGNITION



*Named Entity Recognition (NER) labels sequences of words in a text which are the names of things, such as person and company names...**

What market does FitBit compete in?

Organization

STEP 1 – PYTHON AND NER



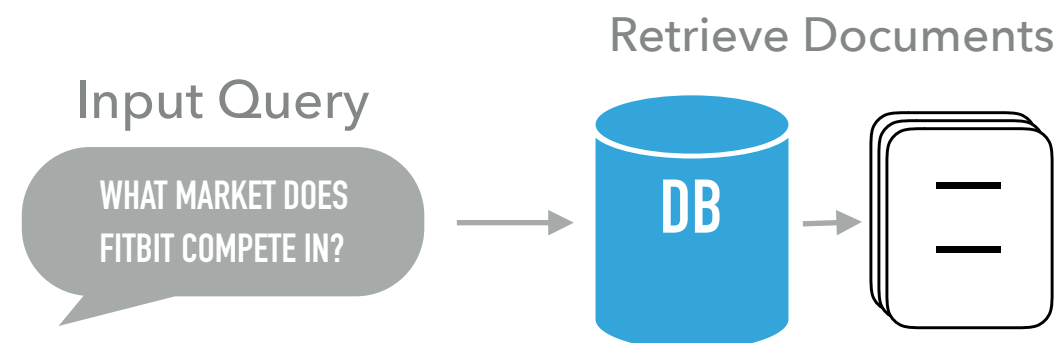
NER is implemented in many different tools, and is still a problem being worked on. A couple of Python implementations can be found in:

► [spaCy](#)

► nltk*

*Will be our choice due to ease of use and already being a dependency in a downstream process we'll see later. spaCy's NER outperforms nltk in general.

STEP 1 – NER IN NLTK



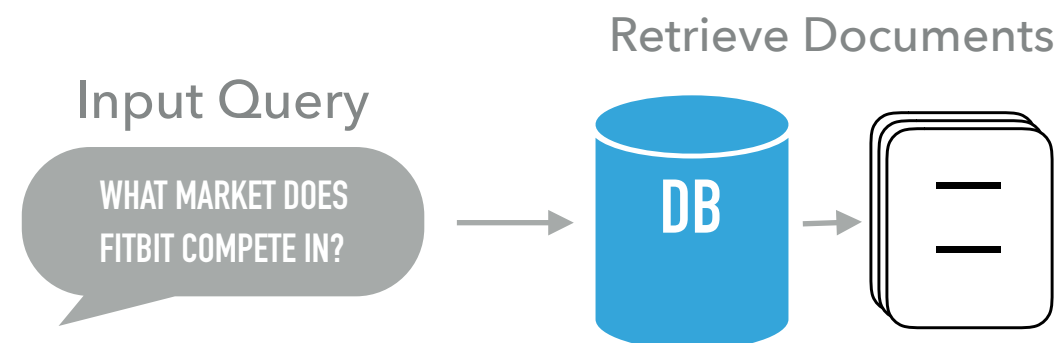
```
QA_NE_TYPES = frozenset(['ORGANIZATION', 'LOCATION', 'FACILITY', 'GPE'])

def ner_extract(text, ne_types=QA_NE_TYPES):
    """Remove non named entities from a string

    :param text: str to remove non named entities from
    :param ne_types: list/set of named entities to keep
    :return: text with non named entities removed
    """
    chunks = nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(text)))
    ne_list = []
    for chunk in chunks:
        if hasattr(chunk, 'label'):
            if chunk.label() in ne_types:
                full_ne = ' '.join(c[0] for c in chunk)
                ne_list.append(full_ne)

    return ' '.join(ne_list)
```

STEP 1 – NER IN NLTK



Ignoring some types like
MONEY & DATE

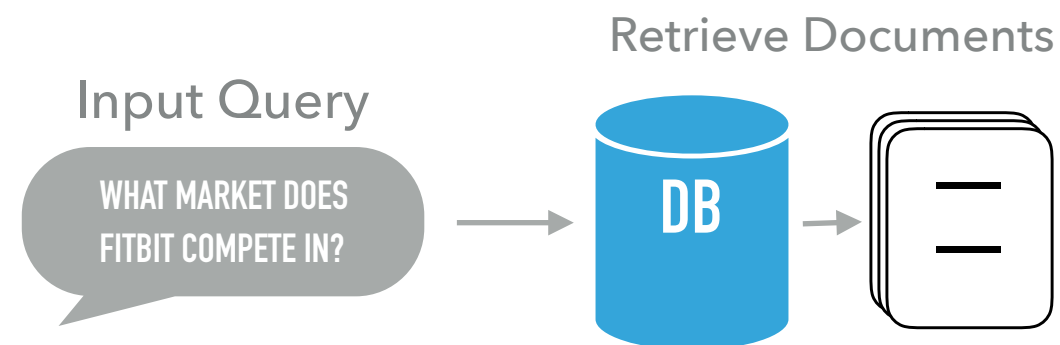
```
QA_NE_TYPES = frozenset(['ORGANIZATION', 'LOCATION', 'FACILITY', 'GPE'])
```

```
def ner_extract(text, ne_types=QA_NE_TYPES):
    """Remove non named entities from a string

    :param text: str to remove non named entities from
    :param ne_types: list/set of named entities to keep
    :return: text with non named entities removed
    """
    chunks = nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(text)))
    ne_list = []
    for chunk in chunks:
        if hasattr(chunk, 'label'):
            if chunk.label() in ne_types:
                full_ne = ' '.join(c[0] for c in chunk)
                ne_list.append(full_ne)

    return ' '.join(ne_list)
```

STEP 1 – NER IN NLTK



```
>>> ner_extract('What market does FitBit compete in?')  
'FitBit'
```

Essentially treats all non Named Entities as stop words.

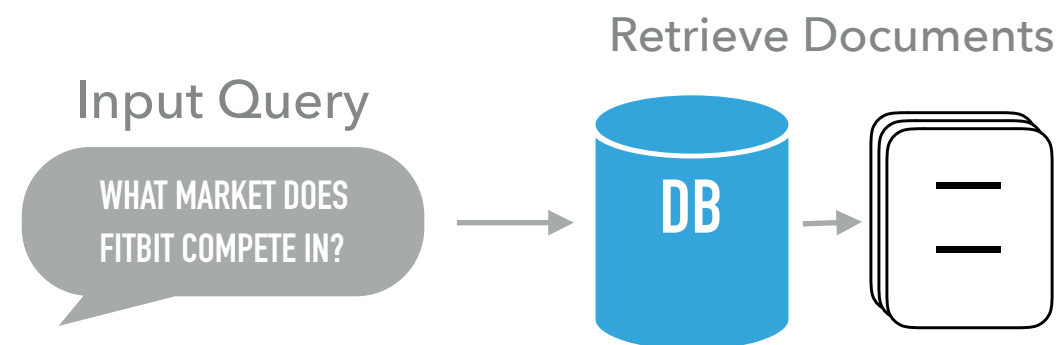
STEP 1 – EXTENDING WHOOSH WITH NER



```
class NERTokenizer(RegexTokenizer):
    """Named Entity centric version of RegexTokenizer"""
    def __init__(self, ne_types=QA_NE_TYPES, expression=default_pattern, gaps=False):
        self.ne_types = ne_types
        super().__init__(expression=expression, gaps=gaps)

    def __call__(self, text, **kwargs):
        text = ner_extract(text=text, ne_types=self.ne_types)
        return super().__call__(text, **kwargs)
```

STEP 1 – EXTENDING WHOOSH WITH NER

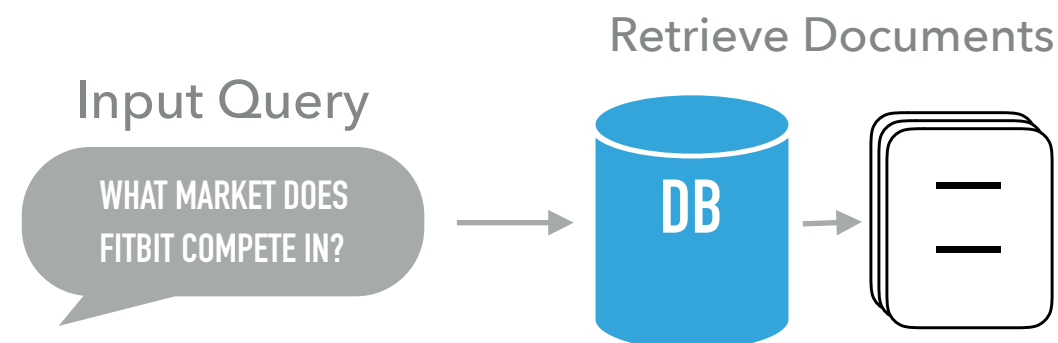


```
class NERTokenizer(RegexTokenizer):
    """Named Entity centric version of RegexTokenizer"""
    def __init__(self, ne_types=QA_NE_TYPES, expression=default_pattern, gaps=False):
        self.ne_types = ne_types
        super().__init__(expression=expression, gaps=gaps)

    def __call__(self, text, **kwargs):
        text = ner_extract(text=text, ne_types=self.ne_types)
        return super().__call__(text, **kwargs)
```

Use inheritance + `super()` to build NER into the `RegexTokenizer`'s process with minimal code.

STEP 1 – EXTENDING WHOOSH WITH NER



```
class NERTokenizer(RegexTokenizer):
    """Named Entity centric version of RegexTokenizer"""
    def __init__(self, ne_types=QA_NE_TYPES, expression=default_pattern, gaps=False):
        self.ne_types = ne_types
        super().__init__(expression=expression, gaps=gaps)

    def __call__(self, text, **kwargs):
        text = ner_extract(text=text, ne_types=self.ne_types)
        return super().__call__(text, **kwargs)
```

NER relies on the structure of the pre-tokenized text, so it needs to go before tokenization.

STEP 1 – TESTING THE NER TOKENIZER



```
>>> [token.text for token in RegexTokenizer("What market does FitBit compete in?")]  
['What', 'market', 'does', 'FitBit', 'compete', 'in']  
>>> [token.text for token in NERTokenizer("What market does FitBit compete in?")]  
['FitBit']
```

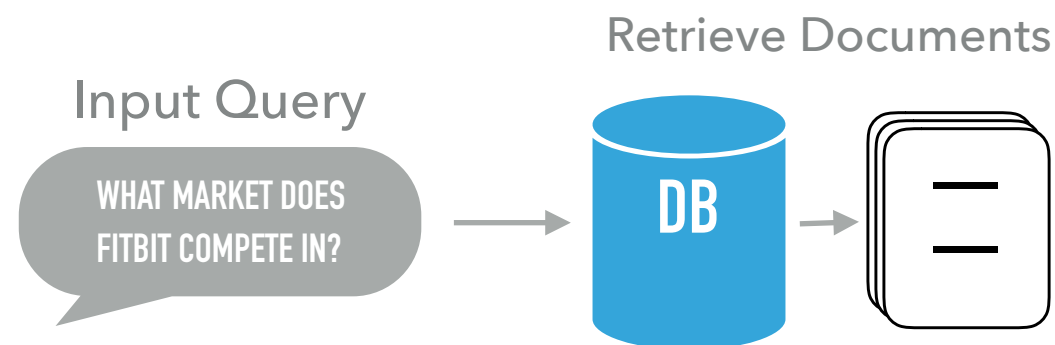
STEP 1 – TESTING THE NER TOKENIZER



```
>>> [token.text for token in RegexTokenizer("What market does FitBit compete in?")]  
['What', 'market', 'does', 'FitBit', 'compete', 'in']  
>>> [token.text for token in NERTokenizer("What market does FitBit compete in?")]  
['FitBit']
```



STEP 1 – Q&A ANALYZER ONE STOP SHOP

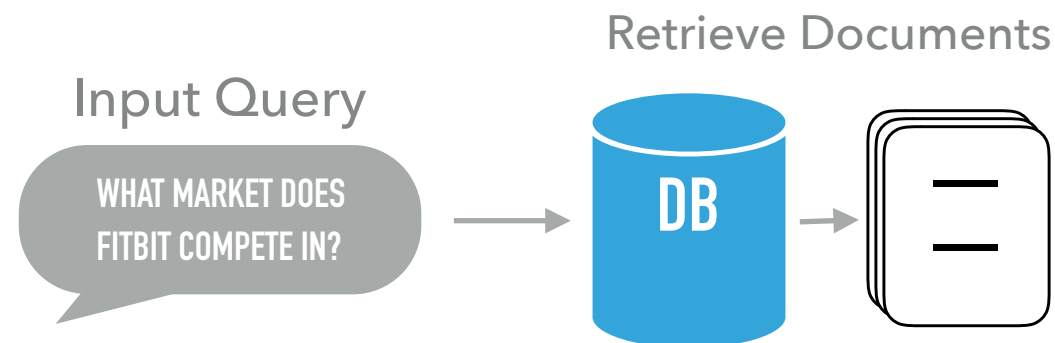


```
# Conforming to camelCase convention used for analyzer functions in whoosh
def NERAnalyzer(ne_types=QA_NE_TYPES, expression=default_pattern, stoplist=QA_STOPWORDS,
                minsize=2, maxsize=None, gaps=False):
    """Named Entity centric version of StandardAnalyzer"""
    chain = NERTokenizer(ne_types, expression, gaps)
    chain |= LowercaseFilter()
    chain |= StopFilter(stoplist=stoplist, minsize=minsize, maxsize=maxsize)

    return chain

def QAAalyzer(ner_tokenize=False, ne_types=QA_NE_TYPES, expression=default_pattern,
              stoplist=QA_STOPWORDS, minsize=2, maxsize=None, gaps=False):
    """Custom analyzer for Nasdaq Q&A task"""
    if not ner_tokenize:
        return StemmingAnalyzer(expression, stoplist, minsize, maxsize, gaps)
    else:
        return NERAnalyzer(ne_types, expression, stoplist, minsize, maxsize, gaps)
```

STEP 1 – Q&A ANALYZER ONE STOP SHOP



Analizers and Filters can be chained with the | operator

```
# Conforming to camelCase convention used for analyzer functions in whoosh
def NERAnalyzer(ne_types=QA_NE_TYPES, expression=default_pattern, stoplist=QA_STOPWORDS,
               minsize=2, maxsize=None, gaps=False):
    """Named Entity centric version of StandardAnalyzer"""
    chain = NERTokenizer(ne_types, expression, gaps)
    chain |= LowercaseFilter()
    chain |= StopFilter(stoplist=stoplist, minsize=minsize, maxsize=maxsize)

    return chain

def QAAnalyzer(ner_tokenize=False, ne_types=QA_NE_TYPES, expression=default_pattern,
               stoplist=QA_STOPWORDS, minsize=2, maxsize=None, gaps=False):
    """Custom analyzer for Nasdaq Q&A task"""
    if not ner_tokenize:
        return StemmingAnalyzer(expression, stoplist, minsize, maxsize, gaps)
    else:
        return NERAnalyzer(ne_types, expression, stoplist, minsize, maxsize, gaps)
```

STEP 1 – SCHEMA REDESIGN



```
schema = Schema(  
    url=ID(stored=True, unique=True),  
    published_datetime=ID(stored=True),  
    title=TEXT(stored=True, analyzer=QAAalyzer()),  
    article=TEXT(stored=True, analyzer=QAAalyzer()),  
    title_named_entities=TEXT(analyzer=QAAalyzer(ner_tokenize=True)),  
    article_named_entities=TEXT(analyzer=QAAalyzer(ner_tokenize=True)),  
)
```

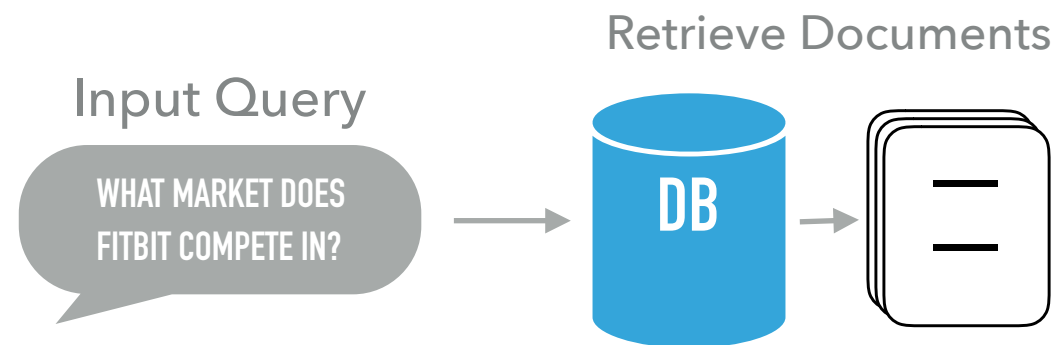
STEP 1 – SCHEMA REDESIGN



```
schema = Schema(
    url=ID(stored=True, unique=True),
    published_datetime=ID(stored=True),
    title=TEXT(stored=True, analyzer=QAAalyzer()),
    article=TEXT(stored=True, analyzer=QAAalyzer()),
    title_named_entities=TEXT(analyzer=QAAalyzer(ner_tokenize=True)),
    article_named_entities=TEXT(analyzer=QAAalyzer(ner_tokenize=True)),
)
```

NER is a speed bottle neck in the indexing process. Rebuilding index with NER takes significantly longer

STEP 1 – SEARCHING NEW SCHEMA



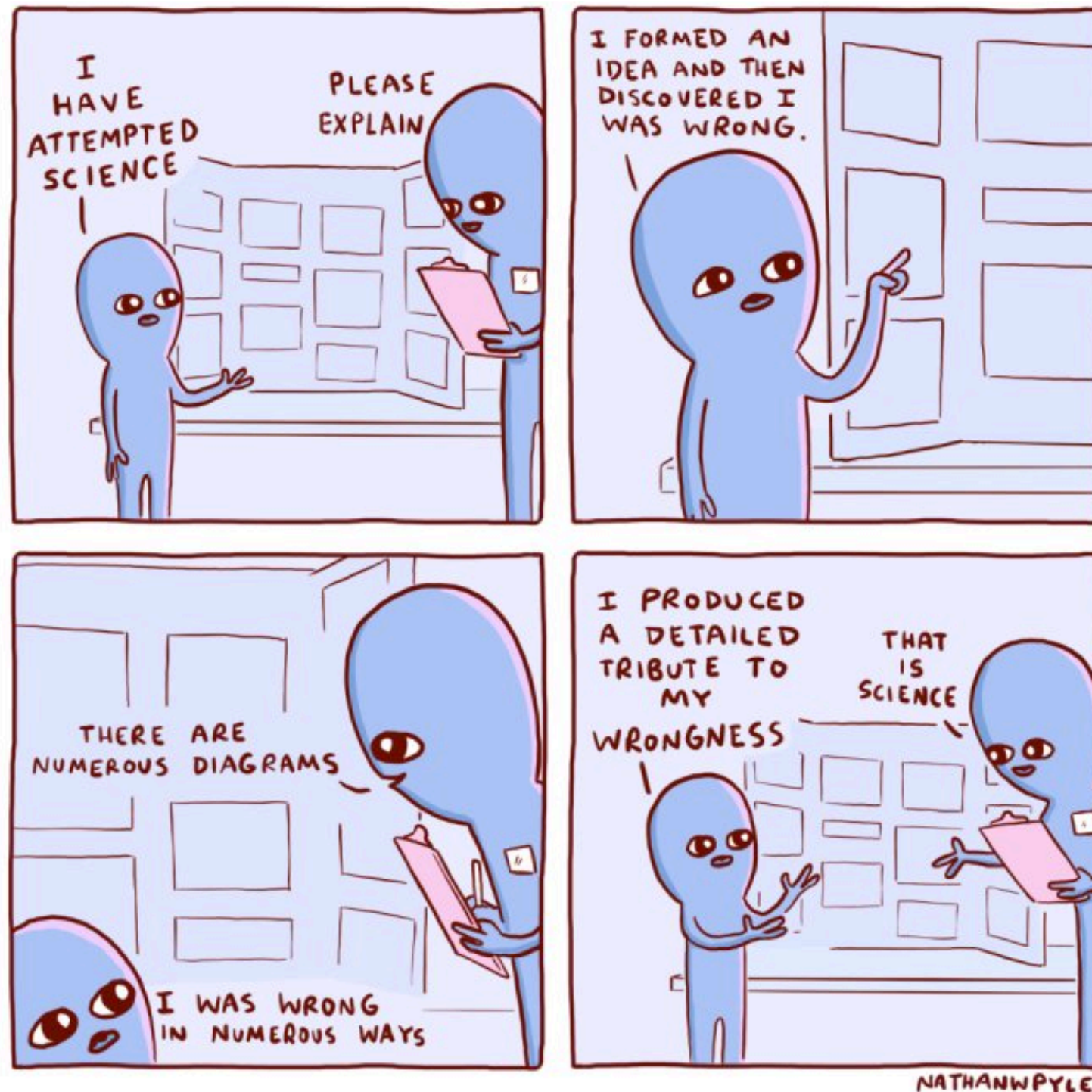
```
>>> import whoosh.index
... from whoosh.qparser import MultifieldParser, OrGroup, WildcardPlugin
...
... whoosh_idx = whoosh.index.open_dir('whoosh_idx', indexname='nasdaq')
... query_parser = MultifieldParser(['title', 'article',
...                                 'title_named_entities', 'article_named_entities'],
...                                 schema=whoosh_idx.schema,
...                                 group=OrGroup)
... query_parser.remove_plugin_class(WildcardPlugin)
...
... parsed_query = query_parser.parse('What market does FitBit compete in?')
...
... with whoosh_idx.searcher() as searcher:
...     search_results = searcher.search(parsed_query, limit=1)
...     [print(sr['title']) for sr in search_results]
```

Apple Watch Sleep Tracking Could Come as Soon as Next Week – Nasdaq.com

STEP 1 – SEARCHING NEW SCHEMA

```
>>> import whoosh
... from whoosh.
...
... whoosh_idx =
... query_parser
...
... query_parser
... parsed_query
...
... with whoosh
... search_r
... [print(s
```

Apple Watch Slee



d_entities'],

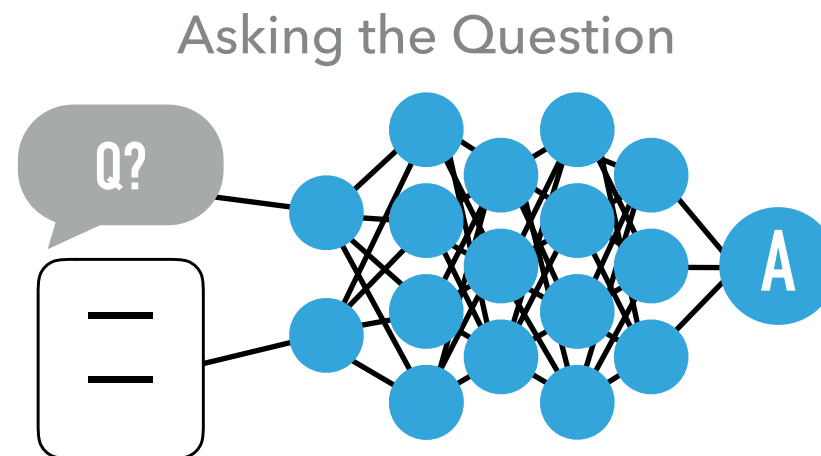
')

STEP 1 – SEARCHING NEW SCHEMA



- ▶ One test case doesn't prove much
- ▶ Adhoc analysis shows the original answer that seems right...
 - ▶ ...mentions FitBit less than our new answer. So our attempt at boosting Named Entities in search results was successful, but possibly misguided.
 - ▶ ...mentions "what" a lot, which would boost its rank. So we might have arrived at the right answer the wrong way.

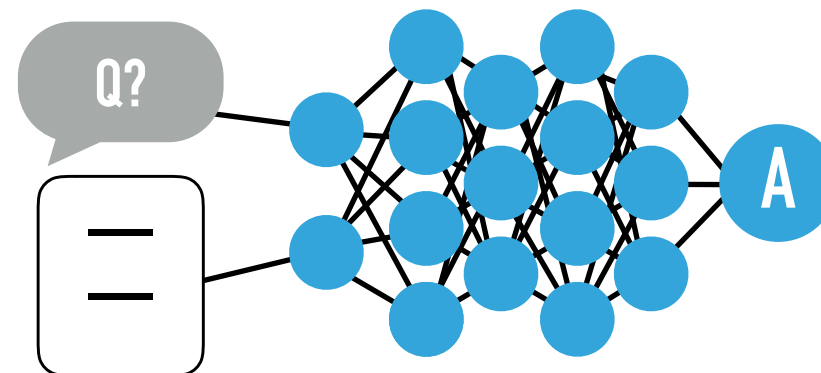
STEP 2 – SQuAD



*Stanford **Q**uestion **A**nswering **D**ataset (SQuAD) is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable.**

STEP 2 – SQuAD LEADERBOARD

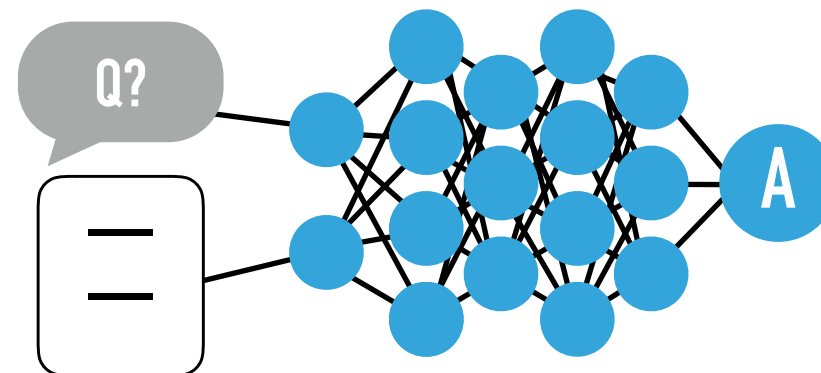
Asking the Question



Rank	Model	EM	F1
	Human Performance Stanford University (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Sep 18, 2019	ALBERT (ensemble model) Google Language ALBERT Team	89.731	92.215
2 Jul 22, 2019	XLNet + DAAF + Verifier (ensemble) PINGAN Omni-Sinitic	88.592	90.859
2 Sep 16, 2019	ALBERT (single model) Google Language ALBERT Team	88.107	90.902

STEP 2 – SQuAD LEADERBOARD

Asking the Question

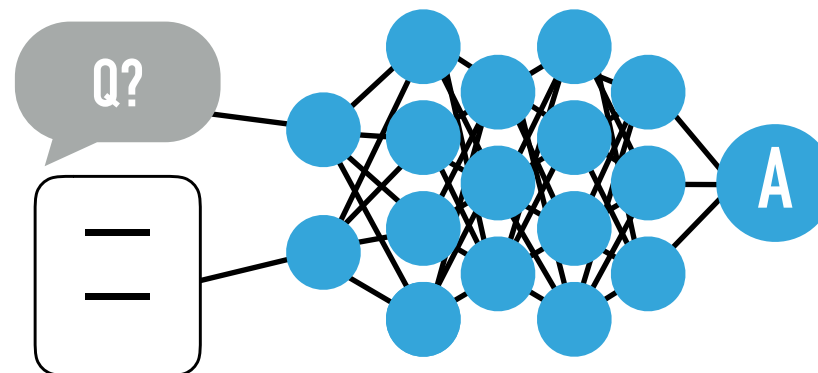


Problem still very actively
being worked on

Rank	Model	EM	F1
	Human Performance Stanford University (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Sep 18, 2019	ALBERT (ensemble model) Google Language ALBERT Team	89.731	92.215
2 Jul 22, 2019	XLNet + DAAF + Verifier (ensemble) PINGAN Omni-Sinitic	88.592	90.859
2 Sep 16, 2019	ALBERT (single model) Google Language ALBERT Team	88.107	90.902

STEP 2 – SQuAD LEADERBOARD

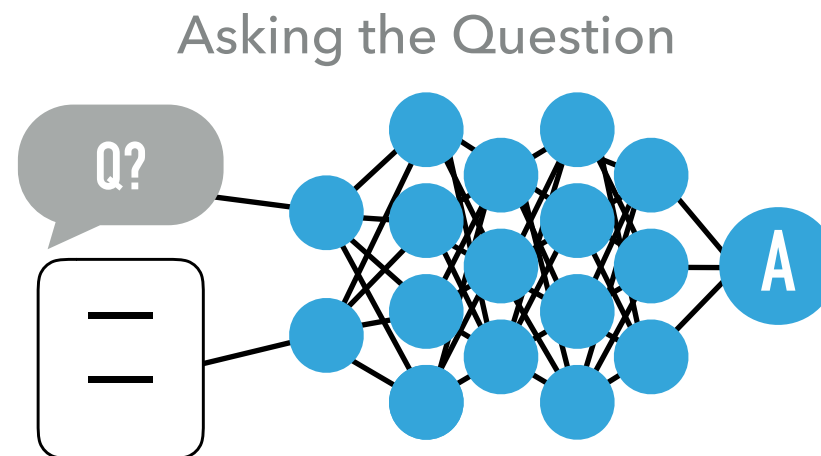
Asking the Question



This ALBERT guy seems to do pretty well

Rank	Model	EM	F1
	Human Performance Stanford University (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Sep 18, 2019	ALBERT (ensemble model) Google Language ALBERT Team	89.731	92.215
2 Jul 22, 2019	XLNet + DAAF + Verifier (ensemble) PINGAN Omni-Sinitic	88.592	90.859
2 Sep 16, 2019	ALBERT (single model) Google Language ALBERT Team	88.107	90.902

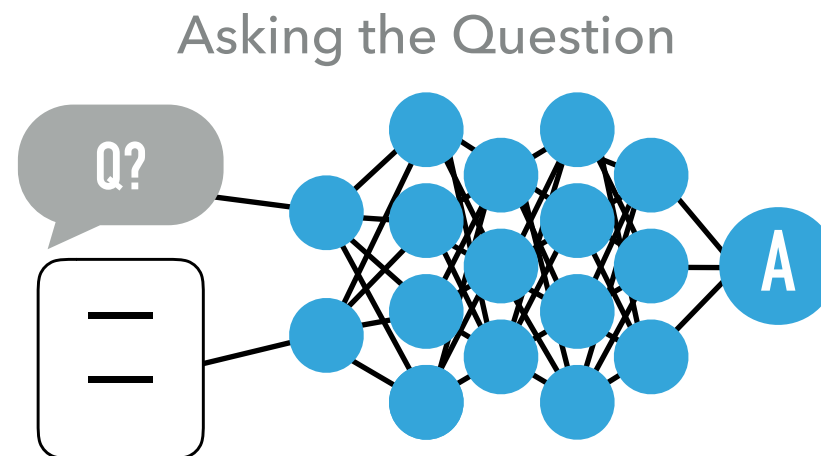
STEP 2 – ALBERT'S OLDER BROTHER, BERT



BERT (**B**idirectional **E**ncoder **R**epresentations for **T**ransformers) from Google (2018) is the predecessor to the ALBERT (**A** Lite **B**ERT).

BERT came on to the scene and immediately started topping the leaderboard of a lot of NLP tasks like SQuAD.

STEP 2 – BERT

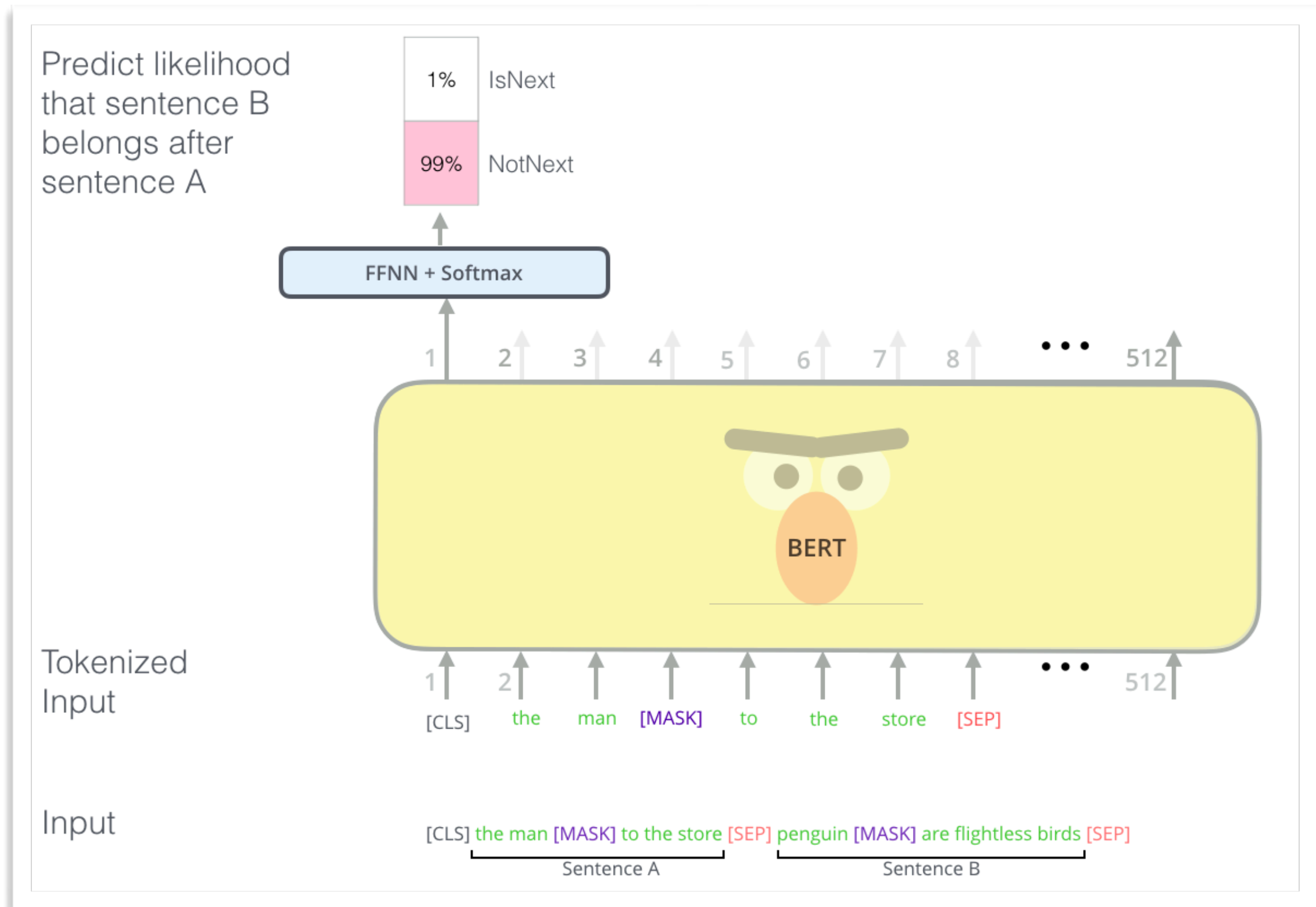


Predecessor language models, like GPT-2, are trained with the intention of predicting the next word given the previous words.

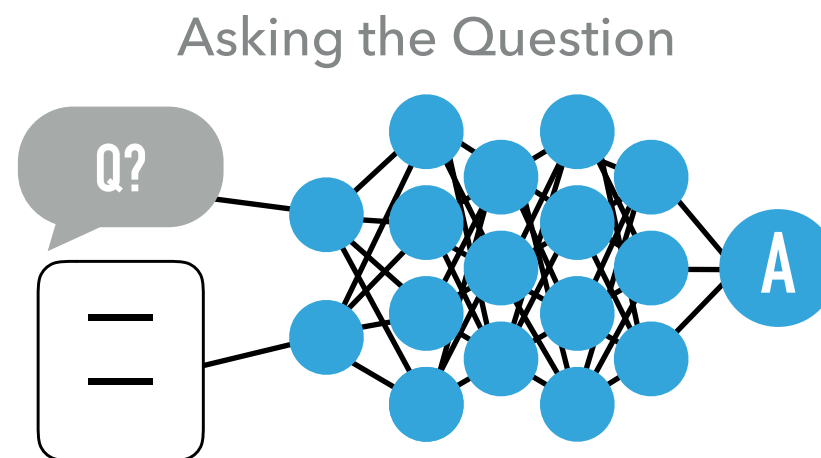


The direction can be reversed and combined with the other, like in ELMo, but these embeddings are trained uni-directionally.





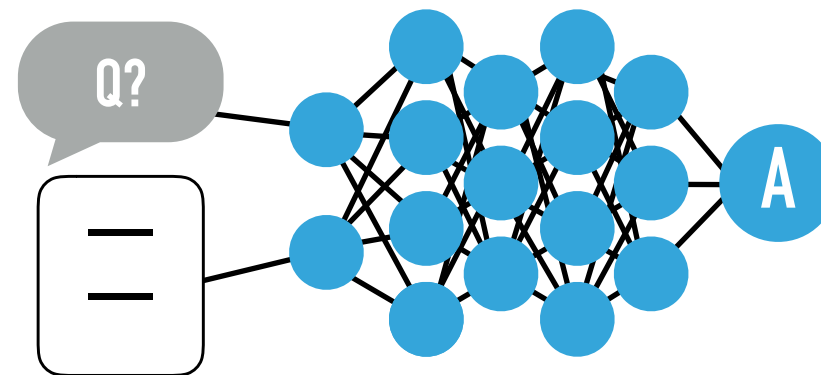
STEP 2 – BERT PRACTICAL APPLICATION KNOWLEDGE



- ▶ It works really well on a lot a of NLP/NLU tasks (including SQuAD)
- ▶ Some SQuAD specific BERT models are already pertained, open-sourced, and packaged. Like the one provided by [DeepPavlov](#)

STEP 2 – DEEPPAVLOV + BERT + SQuAD

Asking the Question

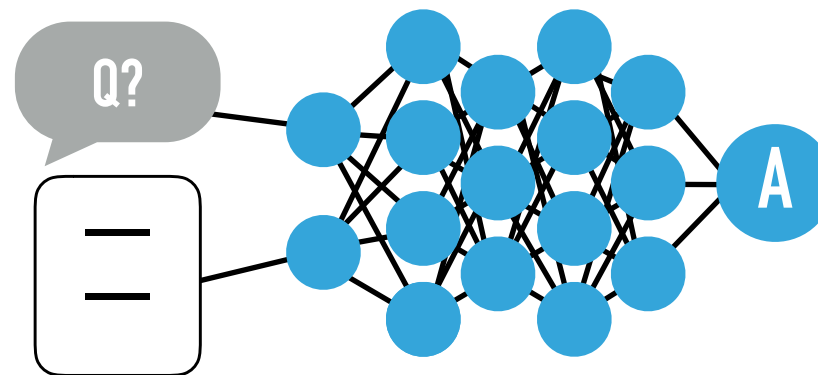


The pre-trained model we'll be using was trained on SQuAD 1.1. In SQuAD 2.0 the label of "No Answer" was introduced. Models trained with 1.1 will always provide what they think is the best answer (even if the answer is terrible).

2.0 was chosen based on accuracy given experimentation with the DeepPavlov provided models.

STEP 2 – USING THE MODEL

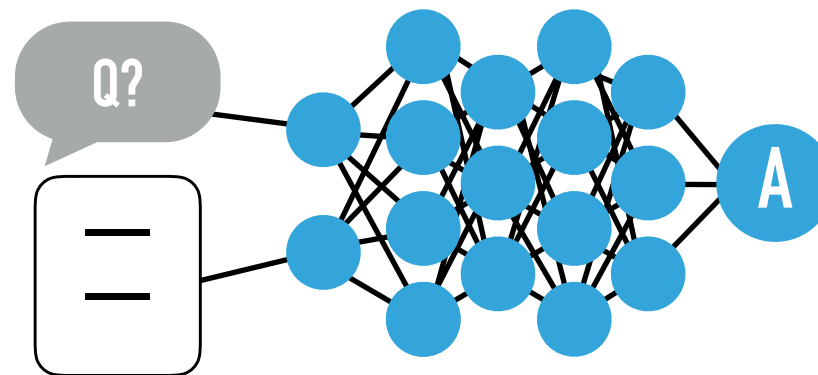
Asking the Question



```
>>> from deeppavlov import build_model, configs
...
... document = ['The rain in Spain falls mainly on the plain.']
... question = ['Where does the rain in Spain fall?']
...
... model = build_model(configs.squad.squad, download=False)
... model(document, question)
[['the plain'], [34], [132021.109375]]
```

STEP 2 – USING THE MODEL

Asking the Question

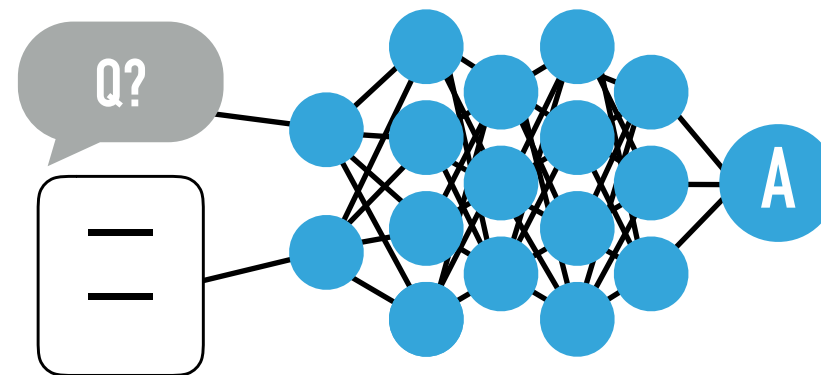


```
>>> from deeppavlov import build_model, configs
...
... document = ['The rain in Spain falls mainly on the plain.']
... question = ['Where does the rain in Spain fall?']
...
... model = build_model(configs.squad.squad, download=False)
... model(document, question)
[['the plain'], [34], [132021.109375]]
```

The answer itself

STEP 2 – USING THE MODEL

Asking the Question

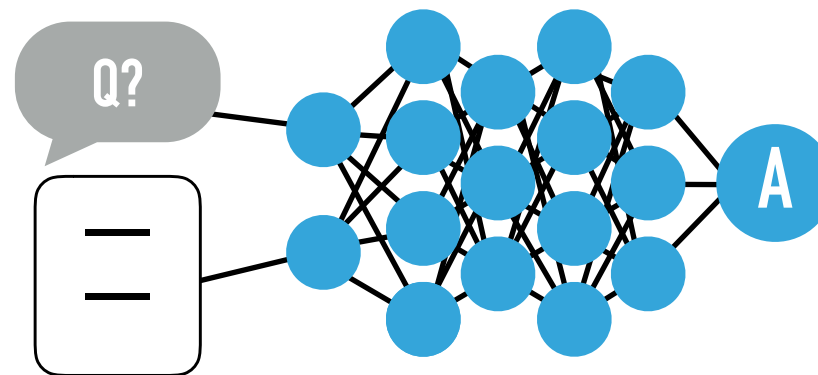


```
>>> from deeppavlov import build_model, configs
...
... document = ['The rain in Spain falls mainly on the plain.']
... question = ['Where does the rain in Spain fall?']
...
... model = build_model(configs.squad.squad, download=False)
... model(document, question)
[['the plain'], [34], [132021.109375]]
```

The answer's first
character's index in the
document

STEP 2 – USING THE MODEL

Asking the Question



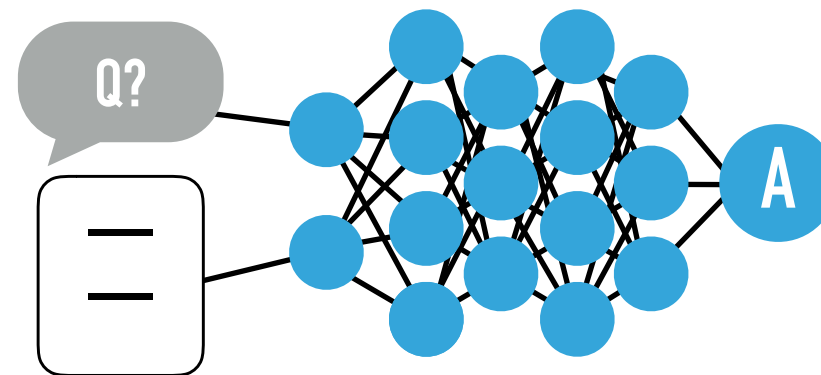
```
>>> from deeppavlov import build_model, configs
...
... document = ['The rain in Spain falls mainly on the plain.']
... question = ['Where does the rain in Spain fall?']
...
... model = build_model(configs.squad.squad, download=False)
... model(document, question)
[['the plain'], [34], [132021.109375]]
```

A measure of answer
confidence

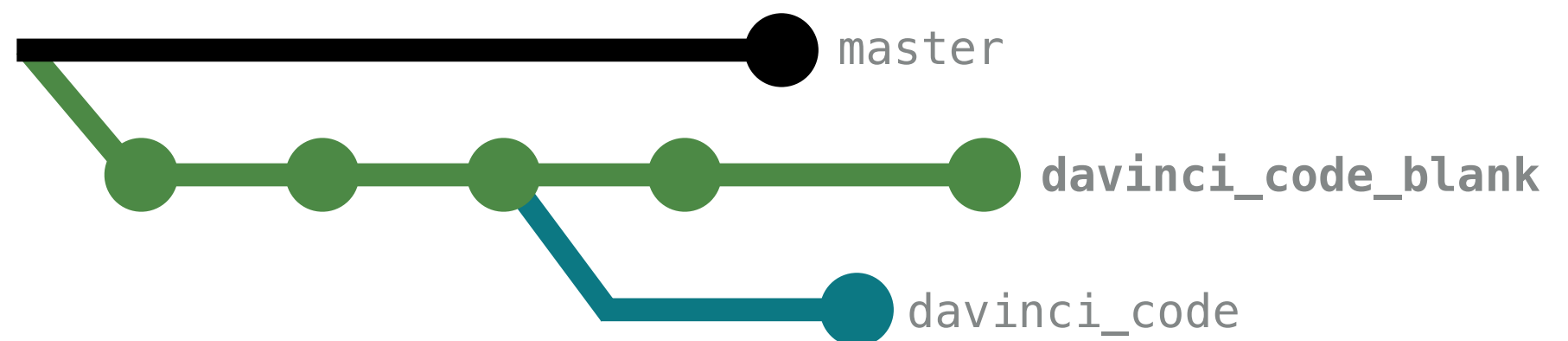


STEP 2 – DO IT YOURSELF

Asking the Question

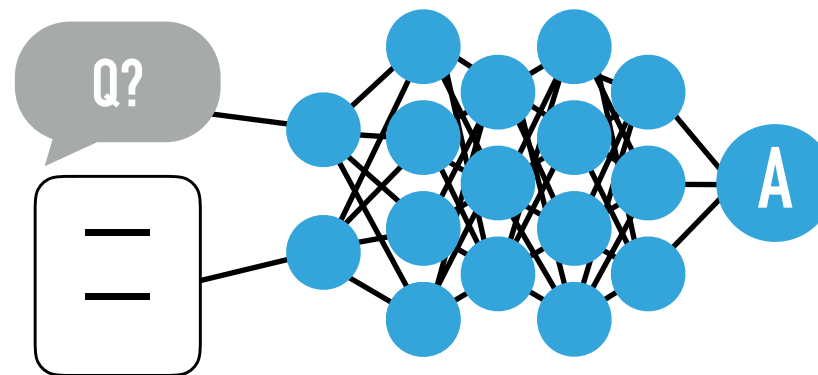


You have now seen enough to complete the file: [bert_squad_example.py](#) in the [davinci_code_blank](#) branch. Completed version of this file can be seen in the [davinci_code](#) branch.



STEP 2 – WRAPPING UP THE MODEL

Asking the Question

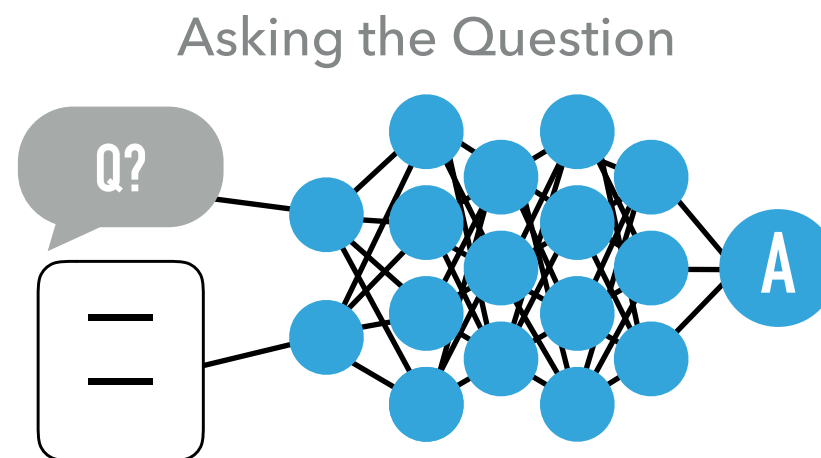


```
from deeppavlov import build_model, configs

class BertSquad:
    def __init__(self, config=configs.squad.squad, download=False):
        self.model = build_model(config, download=download)

    def ask_question(self, document, question):
        answer = self.model(document, question)
        return answer
```

STEP 2 – WRAPPING UP THE MODEL



```
from deeppavlov import build_model, configs

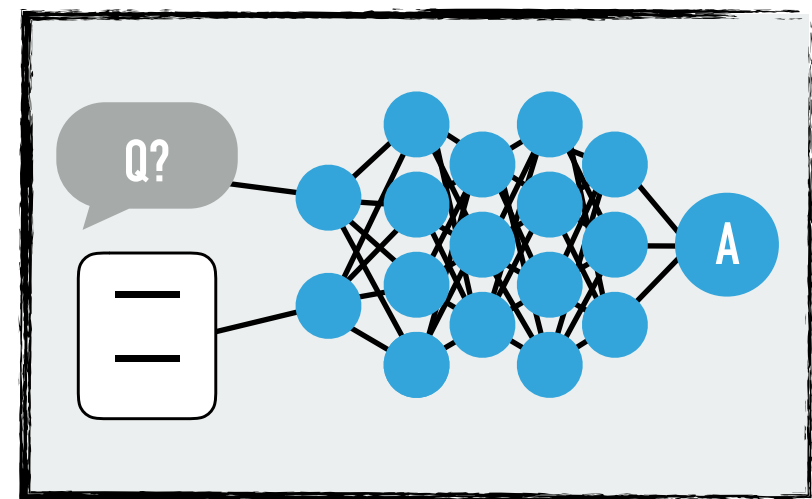
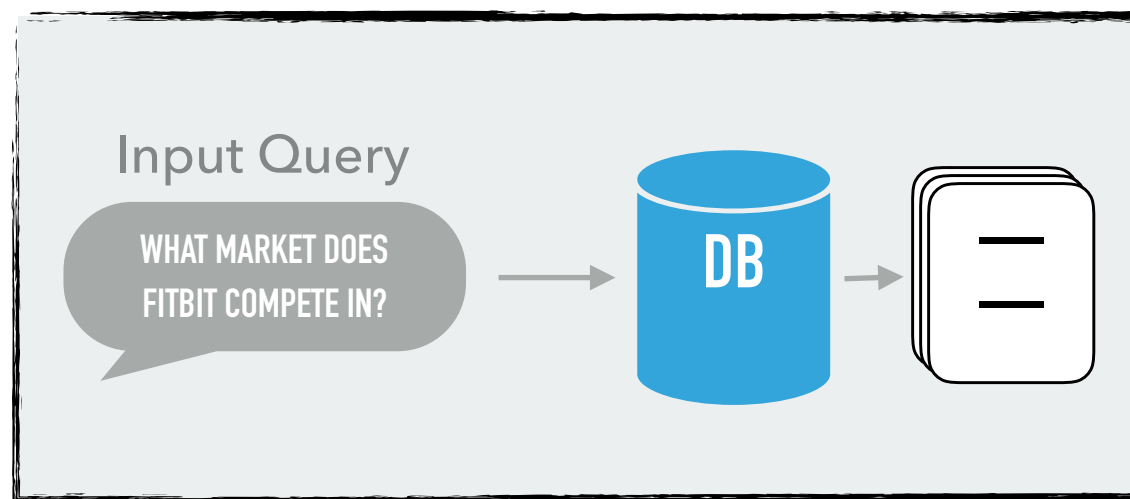
class BertSquad:
    def __init__(self, config=configs.squad.squad, download=False):
        self.model = build_model(config, download=download)

    def ask_question(self, document, question):
        answer = self.model(document, question)
        return answer
```

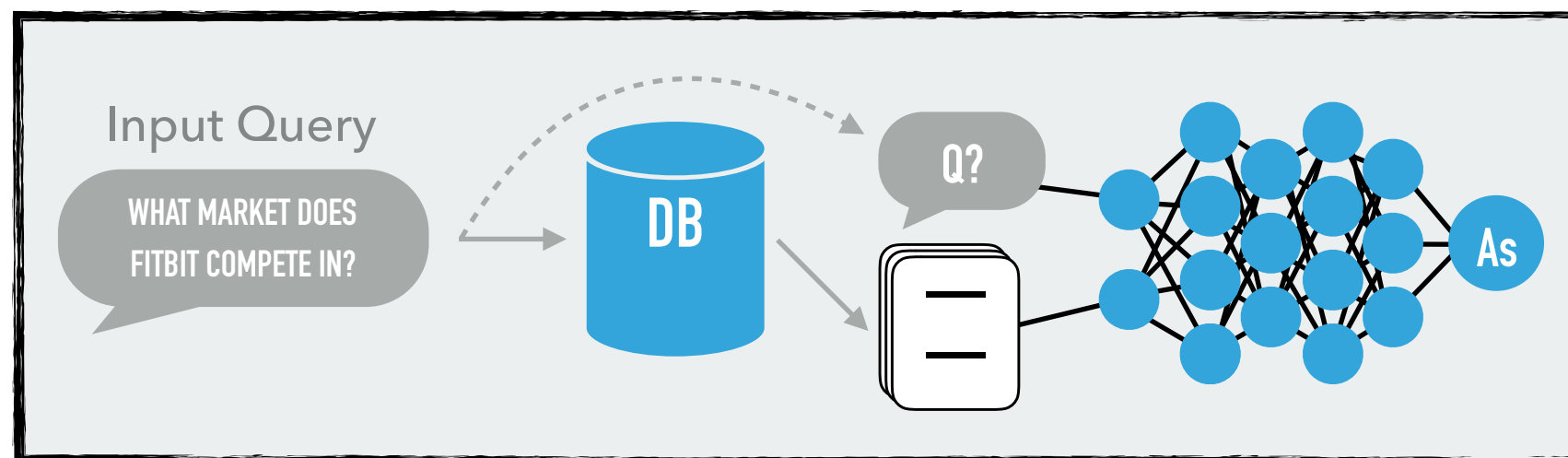
Admittedly, not an incredibly high payoff for creating this class. Potentially future-proofs downstream usage from re-writes due to DeepPavlov API changes.

STEP 2.5 – COMBINING SEARCH AND BERT

Where we are:



Where we want to be:



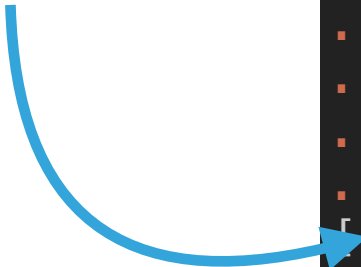
STEP 2.5 – COMBINING SEARCH AND BERT

```
>>> import whoosh.index
... from whoosh.qparser import MultifieldParser, OrGroup, WildcardPlugin
... from qa_query import BertSquad
...
... # Init Q&A process
... bert_squad = BertSquad()
...
... # Init search process
... whoosh_idx = whoosh.index.open_dir('whoosh_idx', indexname='nasdaq')
... query_parser = MultifieldParser(['title', 'article'],
...                                 schema=whoosh_idx.schema,
...                                 group=OrGroup)
... query_parser.remove_plugin_class(WildcardPlugin)
...
... # Perform Q&A query
... question = 'What market does FitBit compete in?'
... parsed_query = query_parser.parse(question)
... with whoosh_idx.searcher() as searcher:
...     search_results = searcher.search(parsed_query, limit=3)
...     result_texts = [sr['article'] for sr in search_results]
...     answers = [bert_squad.ask_question([t], [question]) for t in result_texts]
...
... print(answers)
[[['wearables'], [517], [8197.130859375]],
 [['crowded market'], [1791], [44439.89453125]],
 [['Apple'], [57], [1502.902587890625]]]
```

STEP 2.5 – COMBINING SEARCH AND BERT

```
>>> import whoosh.index
... from whoosh.qparser import MultifieldParser, OrGroup, WildcardPlugin
... from qa_query import BertSquad
...
... # Init Q&A process
... bert_squad = BertSquad()
...
... # Init search process
... whoosh_idx = whoosh.index.open_dir('whoosh_idx', indexname='nasdaq')
... query_parser = MultifieldParser(['title', 'article'],
...                                 schema=whoosh_idx.schema,
...                                 group=OrGroup)
... query_parser.remove_plugin_class(WildcardPlugin)
...
... # Perform Q&A query
... question = 'What market does FitBit compete in?'
... parsed_query = query_parser.parse(question)
... with whoosh_idx.searcher() as searcher:
...     search_results = searcher.search(parsed_query, limit=3)
...     result_texts = [sr['article'] for sr in search_results]
...     answers = [bert_squad.ask_question([t], [question]) for t in result_texts]
...
... print(answers)
[['wearables'], [517], [8197.130859375]],
[['crowded market'], [1791], [44439.89453125]],
[['Apple'], [57], [1502.902587890625]]]
```

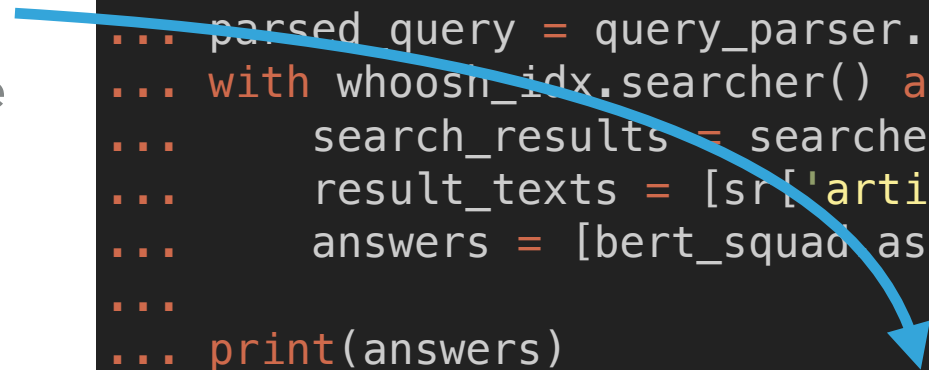
Advertised Q&A
achieved; I've
officially held up
my end of the
bargain



STEP 2.5 – COMBINING SEARCH AND SQuAD

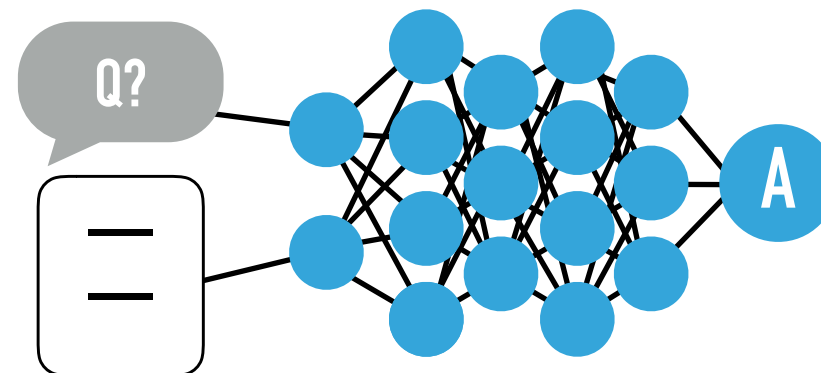
```
>>> import whoosh.index
... from whoosh.qparser import MultifieldParser, OrGroup, WildcardPlugin
... from qa_query import BertSquad
...
... # Init Q&A process
... bert_squad = BertSquad()
...
... # Init search process
... whoosh_idx = whoosh.index.open_dir('whoosh_idx', indexname='nasdaq')
... query_parser = MultifieldParser(['title', 'article'],
...                                 schema=whoosh_idx.schema,
...                                 group=OrGroup)
... query_parser.remove_plugin_class(WildcardPlugin)
...
... # Perform Q&A query
... question = 'What market does FitBit compete in?'
... parsed_query = query_parser.parse(question)
... with whoosh_idx.searcher() as searcher:
...     search_results = searcher.search(parsed_query, limit=3)
...     result_texts = [sr['article'] for sr in search_results]
...     answers = [bert_squad.ask_question([t], [question]) for t in result_texts]
...
... print(answers)
[[['wearables'], [517], [8197.130859375]],
 [['crowded market'], [1791], [44439.89453125]],
 [['Apple'], [57], [1502.902587890625]]]
```

Best search result
and best SQuAD
result in
disagreement.
Who should we
trust?

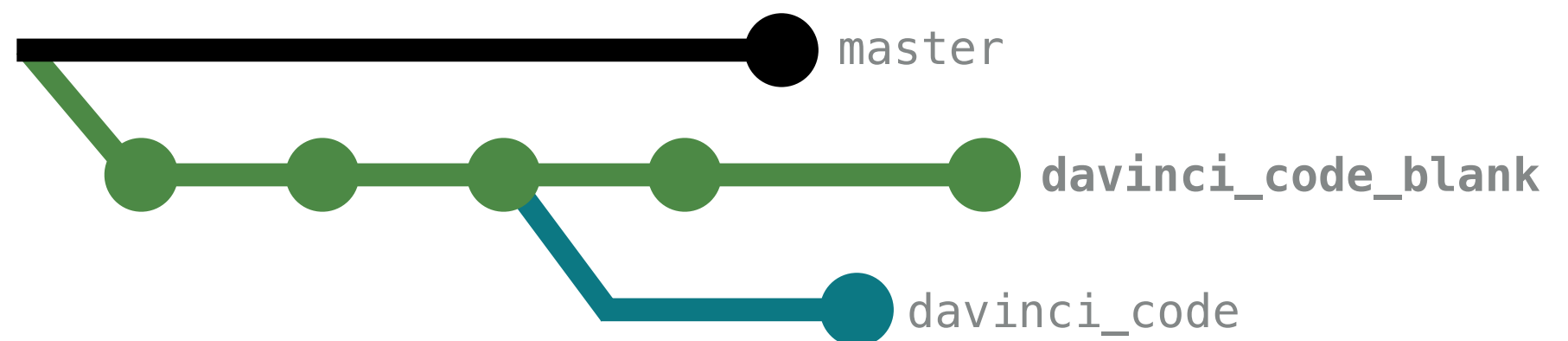


STEP 2 – DO IT YOURSELF

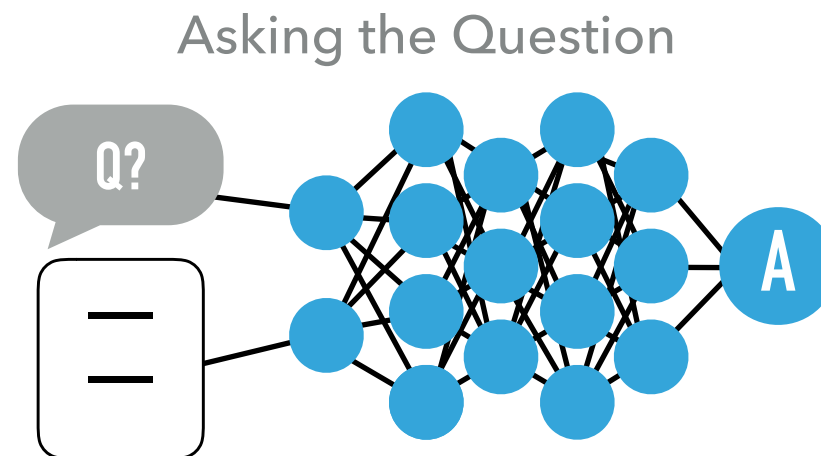
Asking the Question



You have now seen enough to complete the file: [qa_search_davinci_index.py](#) in the [davinci_code_blank](#) branch. Completed version of this file can be seen in the [davinci_code](#) branch.



STEP 2 – DO IT YOURSELF



Some test questions:

- ▶ What university is Robert Langdon a professor at?*
- ▶ What is the notable poetic structure of the poem inside the cryptex box?*
- ▶ What subject is Robert Langdon a professor of?
- ▶ What does Opus Dei mean?

STEP 3 – RANKING ANSWERS

Ranking Answers

#	As
1	_____
2	_____

- ▶ From Whoosh we get a measure of the best document to ask
- ▶ From the SQuAD model we get a measure of the best answer
- ▶ If we're presenting a list of possible answers, rather than relying on returning 1 correct answer, then we can be more lax on this ranking

STEP 3 – RANKING ANSWERS

Ranking Answers

#	As
1	_____
2	_____

Why not both? ㄟ_(\ツ)_/

- ▶ From Whoosh we get a measure of the best document to ask
- ▶ From the SQuAD model we get a measure of the best answer
- ▶ If we're presenting a list of possible answers, rather than relying on returning 1 correct answer, then we can be more lax on this ranking

STEP 3 – PRETTYING UP RESULTS

Ranking Answers

#	As
1	-----
2	-----

("exit" to quit) Question: What market does FitBit compete in?

	Search Rank	Squad Rank	Combined Rank		Answer	Context
0	2.0	6.0	1.5		crowded market	... rather crowded ma...
1	7.0	1.0	1.5		oncology	...te in the oncology ...
2	8.0	3.0	3.0		streaming services market	...trate the streaming ...
3	5.0	7.0	4.0		leash	...ck a long leash. Boe...
4	4.0	9.0	5.0		iPhones	...io of new iPhones – ...
5	1.0	13.0	6.5		wearables	...hough, is wearables,...
6	9.0	5.0	6.5		murky	... 're in a "murky" mar...
7	15.0	2.0	8.0		ad market	...ce in the ad market ...
8	14.0	4.0	9.0		HelloSign and Adobe (ADBE) Sign	...etes with HelloSign ...
9	10.0	11.0	10.0		mass	...s but the mass marke...

STEP 3 – PRETTYING UP RESULTS

Ranking Answers

#	As
1	-----
2	-----

Minimal CLI frontend using infinite while loop and input()

```
("exit" to quit) Question: What market does FitBit compete in?

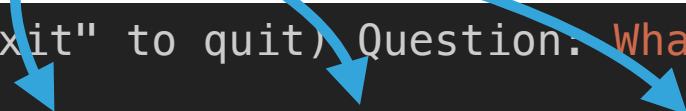
Search Rank  Squad Rank  Combined Rank  Answer  Context
0  2.0        6.0        1.5        crowded market  ... rather crowded ma...
1  7.0        1.0        1.5        oncology        ...te in the oncology ...
2  8.0        3.0        3.0        streaming services market  ...trate the streaming ...
3  5.0        7.0        4.0        leash          ...ck a long leash. Boe...
4  4.0        9.0        5.0        iPhones        ...io of new iPhones - ...
5  1.0        13.0       6.5        wearables      ...hough, is wearables,...
6  9.0        5.0        6.5        murky         ...'re in a "murky" mar...
7  15.0       2.0        8.0        ad market     ...ce in the ad market ...
8  14.0       4.0        9.0        HelloSign and Adobe (ADBE) Sign  ...etes with HelloSign ...
9  10.0      11.0       10.0       mass         ...s but the mass marke...
```

STEP 3 – PRETTYING UP RESULTS

Ranking Answers

#	As
1	-----
2	-----

No clear winner (based on more than this single Q&A)


 ("exit" to quit) Question: What market does FitBit compete in?

	Search Rank	Squad Rank	Combined Rank		Answer	Context
0	2.0	6.0	1.5	crowded market		... rather crowded ma...
1	7.0	1.0	1.5	oncology		...te in the oncology ...
2	8.0	3.0	3.0	streaming services market		...trate the streaming ...
3	5.0	7.0	4.0	leash		...ck a long leash. Boe...
4	4.0	9.0	5.0	iPhones		...io of new iPhones – ...
5	1.0	13.0	6.5	wearables		...hough, is wearables,...
6	9.0	5.0	6.5	murky		... 're in a "murky" mar...
7	15.0	2.0	8.0	ad market		...ce in the ad market ...
8	14.0	4.0	9.0	HelloSign and Adobe (ADBE) Sign		...etes with HelloSign ...
9	10.0	11.0	10.0	mass		...s but the mass marke...

STEP 3 – PRETTYING UP RESULTS

Ranking Answers

#	As
1	-----
2	-----

Can use character index returned from SQuAD to give more context.

("exit" to quit) Question: What market does FitBit compete in?

	Search Rank	Squad Rank	Combined Rank	Answer	Context
0	2.0	6.0	1.5	crowded market	... rather crowded ma...
1	7.0	1.0	1.5	oncology	...te in the oncology ...
2	8.0	3.0	3.0	streaming services market	...trate the streaming ...
3	5.0	7.0	4.0	leash	...ck a long leash. Boe...
4	4.0	9.0	5.0	iPhones	...io of new iPhones – ...
5	1.0	13.0	6.5	wearables	...hough, is wearables,...
6	9.0	5.0	6.5	murky	... 're in a "murky" mar...
7	15.0	2.0	8.0	ad market	...ce in the ad market ...
8	14.0	4.0	9.0	HelloSign and Adobe (ADBE) Sign	...etes with HelloSign ...
9	10.0	11.0	10.0	mass	...s but the mass marke...

STEP 3 – PRETTYING UP RESULTS

Ranking Answers

#	As
1	-----
2	-----

Can use character index returned from SQuAD to give more context.

("exit" to quit) Question: What market does FitBit compete in?

	Search Rank	Squad Rank	Combined Rank	Answer	Context
0	2.0	6.0	1.5	crowded market	... rather crowded ma...
1	7.0	1.0	1.5	oncology	...te in the oncology ...
2	8.0	3.0	3.0	streaming services market	...trate the streaming ...
3	5.0	7.0	4.0	leash	...ck a long leash. Boe...
4	4.0	9.0	5.0	iPhones	...io of new iPhones – ...
5	1.0	13.0	6.5	wearables	...hough, is wearables,...
6	9.0	5.0	6.5	murky	... 're in a "murky" mar...
7	15.0	2.0	8.0	ad market	...ce in the ad market ...
8	14.0	4.0	9.0	HelloSign and Adobe (ADBE) Sign	...etes with HelloSign ...
9	10.0	11.0	10.0	mass	...s but the mass marke...

STEP 3 – PRETTYING UP RESULTS

Ask NASDAQ Articles

Who is Tim Cook?

Ask

Answer	Search Rank	Squad Rank	Combined Rank
Apple's CEO	5	5	1
a subsidiary of Telecom Italia SpA	8	3	2.5
a subsidiary of Telecom Italia SpA	9	2	2.5
CEO	2	10	4
CEO	1	13	6
CEO	3	11	6
a subsidiary of Telecom Italia SpA	7	7	6
a subsidiary of Telecom Italia SpA , Moody	6	9	8
CEO	4	14	9
chairman and CEO, Concho Resources	18	1	10

STEP 3 – PRETTYING UP RESULTS

Fancier frontend made with [Plotly Dash](#)

Ask NASDAQ Articles

Who is Tim Cook?

Ask

Answer	Search Rank	Squad Rank	Combined Rank
Apple's CEO	5	5	1
a subsidiary of Telecom Italia SpA	8	3	2.5
a subsidiary of Telecom Italia SpA	9	2	2.5
CEO	2	10	4
CEO	1	13	6
CEO	3	11	6
a subsidiary of Telecom Italia SpA	7	7	6
a subsidiary of Telecom Italia SpA , Moody	6	9	8
CEO	4	14	9
chairman and CEO, Concho Resources	18	1	10

STEP 4 – HAVING FUN WITH IT – EXAMPLE INPUT/OUTPUT

Ask NASDAQ Articles

Who is Elon Musk?

Ask

Answer	Search Rank	Squad Rank	Combined Rank
eccentric CEO	1	3	1.5
Chief Executive	3	1	1.5
SpaceX	4	7	3.5
SpaceX	6	5	3.5
SpaceX	7	5	5.5
Tesla Chief Executive	10	2	5.5
NASDAQ: TSLA	2	11	7.5
SpaceX	8	5	7.5
Li Xiaopeng	5	9	9
a former punk band member	9	8	10

STEP 4 – HAVING FUN WITH IT – EXAMPLE INPUT/OUTPUT

Ask NASDAQ Articles

Who has the best chicken sandwich?

Ask

Answer	Search Rank	Squad Rank	Combined Rank
Wendy	1	9	1
Corbyn	12	1	2
Jay Schottenstein	8	6	3.5
Tyson	10	4	3.5
Tyson Foods Q3 Earnings Meet Estimates	7	8	6.5
Chick-fil-A brand	3	12	6.5
Costco	13	2	6.5
KFC	4	11	6.5
David and Tom	9	7	9
Tyson Foods	2	16	10

STEP 4 – HAVING FUN WITH IT – EXAMPLE INPUT/OUTPUT

Ask NASDAQ Articles

What is the value of APPL?

Ask

Answer	Search Rank	Squad Rank	Combined Rank
\$869 million.	4	3	1
\$16.10	8	1	2
0.1%	3	8	3
\$46	10	2	4.5
28.3%	6	6	4.5
concern to us	2	12	6
\$50 billion	13	4	7.5
over 30%	12	5	7.5
\$10 billion or more	5	14	9
\$11.4 million	1	20	10

STEP 4 – HAVING FUN WITH IT – EXAMPLE INPUT/OUTPUT

Ask NASDAQ Articles

Where is Facebook located?

Ask

Answer	Search Rank	Squad Rank	Combined Rank
Twitter	3	2	1
in the Chinese market	5	6	2
Los Angeles	2	10	3.5
Menlo Park, California	9	3	3.5
Menlo Park	12	1	5
U.S. and Canada	1	15	6.5
Europe	4	12	6.5
Oasis	11	8	8
Alphabet Inc	7	13	10
tech giants	13	7	10