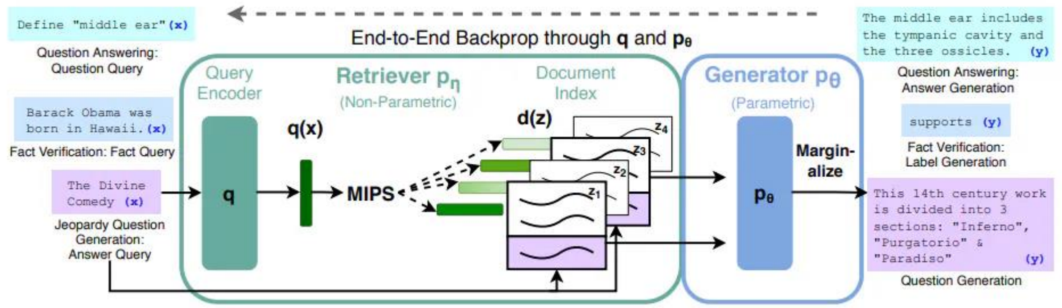


Define "middle ear" (x)
Question Answering: Question Query
Barack Obama was born in Hawaii. (x)
Fact Verification: Fact Query
The Divine Comedy (x)
Jeopardy Question Generation: Answer Query
End-to-End Backprop through q and pθ
Query Encoder
Retriever pη
(Non-Parametric)
Document Index
q
q(x)
MIPS
d(z)
z4
z3
z2
z1
Generator pθ
(Parametric)
pθ
Margin-alize
The middle ear includes the tympanic cavity and the three ossicles. (y)
Question Answering: Answer Generation
supports (y)
Fact Verification: Label Generation
This 14th century work is divided into 3 sections: "Inferno", "Purgatorio" & "Paradiso" (y)
Question Generation

Figure 1: Overview of our approach. We combine a pre-trained retriever (*Query Encoder* + *Document Index*) with a pre-trained seq2seq model (*Generator*) and fine-tune end-to-end. For query x, we use Maximum Inner Product Search (MIPS) to find the top-K documents z_i. For final prediction y, we treat z as a latent variable and marginalize over seq2seq predictions given different documents.

Retrieval Augmented Generation (RAG)

Umar Jamil

Outline

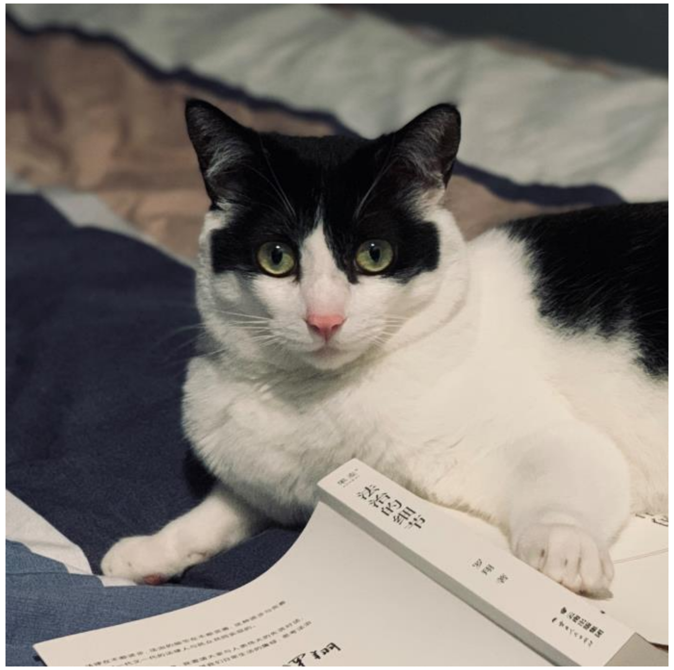
- Intro to Large Language Models
- RAG pipeline
- Embedding vectors
    - Sentence BERT
- Vector DB
    - Algorithms (HNSW)

Prerequisites

- Structure of the Transformer model and how the attention mechanism works.
- BERT (MLM task, [cls] token)

Extra

- As usual, my cat 奥利奥 will also be part of this video.



Outline

- Intro to Large Language Models
- RAG pipeline
- Embedding vectors
  - Sentence BERT
- Vector DB
  - Algorithms (HNSW)

What is a language model?

A language model is a probabilistic model that assign probabilities to sequence of words.
In practice, a language model allows us to compute the following:

$$P\left[\ \underbrace{\text{“China”}}_{\textbf{Next Token}}\ \middle|\ \underbrace{\text{“Shanghai is a city in”}}_{\textbf{Prompt}}\ \right]$$

We usually train a neural network to predict these probabilities. A neural network trained on a large corpora of text is known as a Large Language Model (LLM).

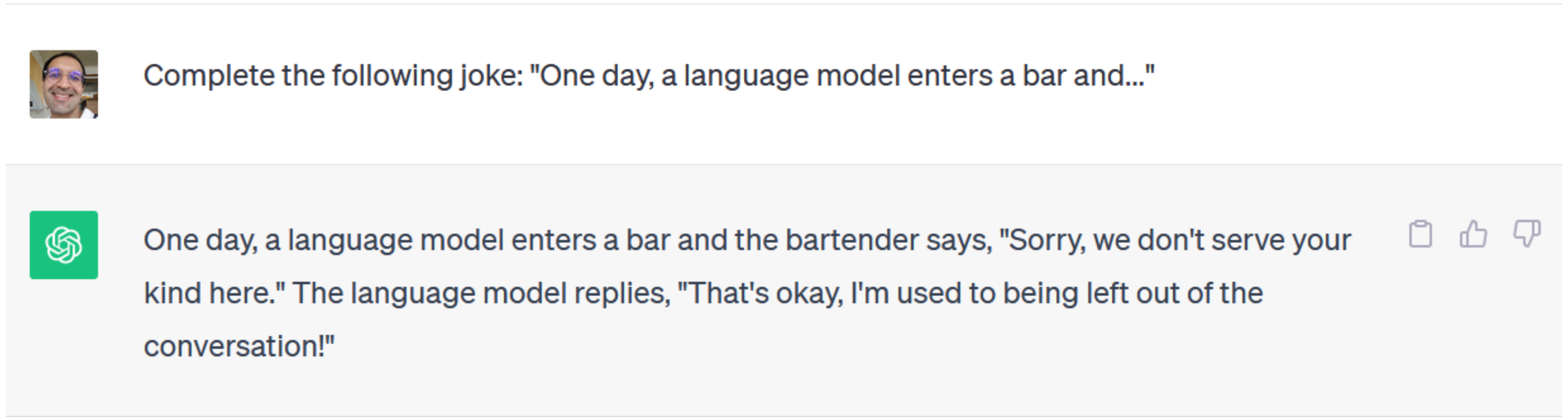
How do we train and inference a Language Model?

Training

A language model is trained on a corpora of text, that is, a large collection of documents. Often, Language Models are trained on the entire Wikipedia and millions of web pages. This allows the Language Model to acquire as much knowledge as possible.
We usually train a Transformer-based neural network as Language Model.

Inference

To inference a Language Model, we build a prompt and let the Language Model generate the rest by iteratively adding tokens.



Complete the following joke: "One day, a language model enters a bar and..."

One day, a language model enters a bar and the bartender says, "Sorry, we don't serve your kind here." The language model replies, "That's okay, I'm used to being left out of the conversation!"

You are what you eat

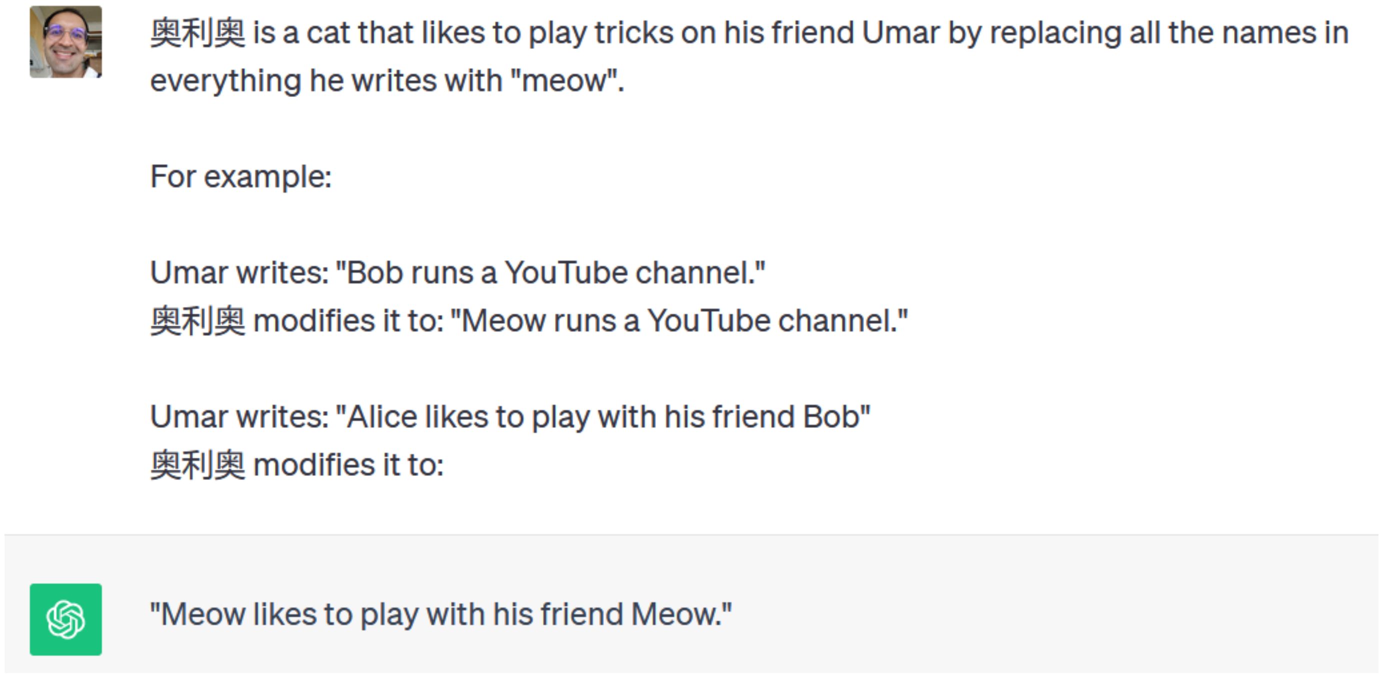
A language model can only output text and information that it was trained upon. This means, that if we train a language model only on English content, very probably it won’t be able to output Japanese or French. To teach new concepts, we need to fine-tune the model.

The cons of fine-tuning

- It can be expensive.
- The number of parameters of the model may not be sufficient to capture all the knowledge we want to teach to it. That's why LLaMA was introduced with 7B, 13B and 70B parameters.
- Fine-Tuning is not additive. It may replace existing knowledge of the model with new knowledge. For example, a language model trained on English that is (heavily) fine-tuned on Japanese may "forget" English.

Prompt Engineering to the rescue!

It is possible to "teach" a language model how to perform a new task by playing with the prompt. For example, by using "few-shot" prompting. The following is an example:



奥利奥 is a cat that likes to play tricks on his friend Umar by replacing all the names in everything he writes with "meow".

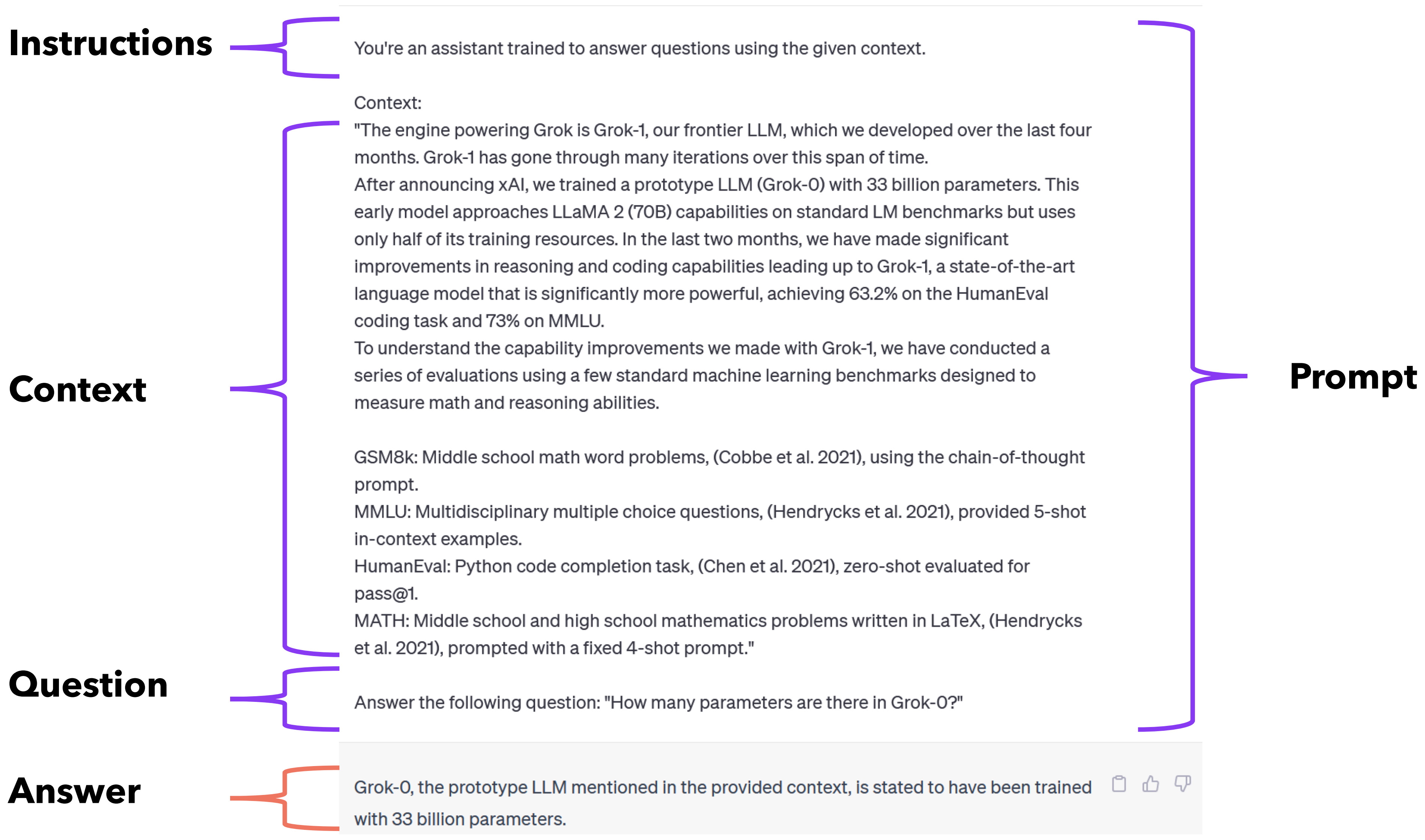
For example:

Umar writes: "Bob runs a YouTube channel."
奥利奥 modifies it to: "Meow runs a YouTube channel."

Umar writes: "Alice likes to play with his friend Bob"
奥利奥 modifies it to:

"Meow likes to play with his friend Meow."

QA with Prompt Engineering



You're an assistant trained to answer questions using the given context.

Context:
"The engine powering Grok is Grok-1, our frontier LLM, which we developed over the last four months. Grok-1 has gone through many iterations over this span of time.
After announcing xAI, we trained a prototype LLM (Grok-0) with 33 billion parameters. This early model approaches LLaMA 2 (70B) capabilities on standard LM benchmarks but uses only half of its training resources. In the last two months, we have made significant improvements in reasoning and coding capabilities leading up to Grok-1, a state-of-the-art language model that is significantly more powerful, achieving 63.2% on the HumanEval coding task and 73% on MMLU.
To understand the capability improvements we made with Grok-1, we have conducted a series of evaluations using a few standard machine learning benchmarks designed to measure math and reasoning abilities.

GSM8k: Middle school math word problems, (Cobbe et al. 2021), using the chain-of-thought prompt.
MMLU: Multidisciplinary multiple choice questions, (Hendrycks et al. 2021), provided 5-shot in-context examples.
HumanEval: Python code completion task, (Chen et al. 2021), zero-shot evaluated for pass@1.
MATH: Middle school and high school mathematics problems written in LaTeX, (Hendrycks et al. 2021), prompted with a fixed 4-shot prompt."

Answer the following question: "How many parameters are there in Grok-0?"

Grok-0, the prototype LLM mentioned in the provided context, is stated to have been trained with 33 billion parameters.

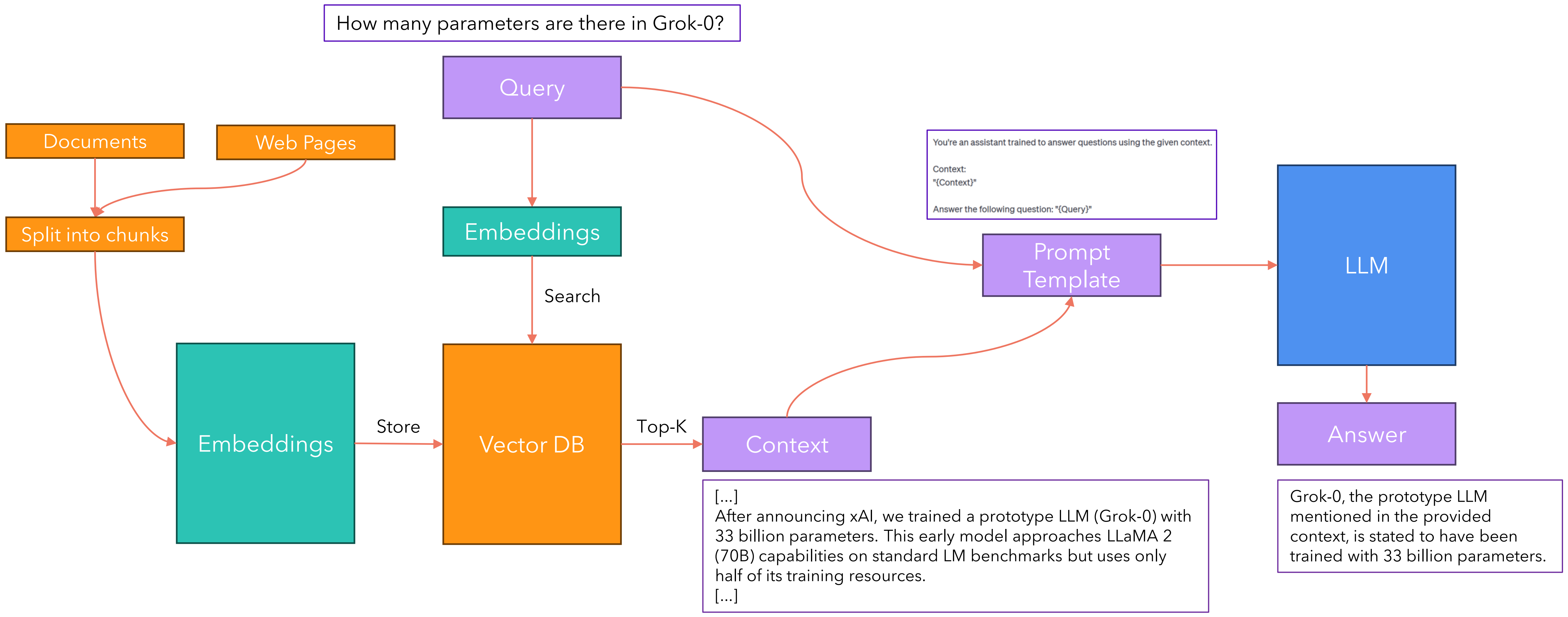
The pros of fine-tuning

- Higher quality results compared to prompt engineering.
- Smaller context size (input size) during inference since we don’t need to include the context and instructions.

Outline

- Intro to Large Language Models
- RAG pipeline
- Embedding vectors
  - Sentence BERT
- Vector DB
  - Algorithms (HNSW)

QA with Retrieval Augmented Generation



You're an assistant trained to answer questions using the given context.

Context:
"{Context}"

Answer the following question: "{Query}"

[...]
After announcing xAI, we trained a prototype LLM (Grok-0) with 33 billion parameters. This early model approaches LLaMA 2 (70B) capabilities on standard LM benchmarks but uses only half of its training resources.
[...]

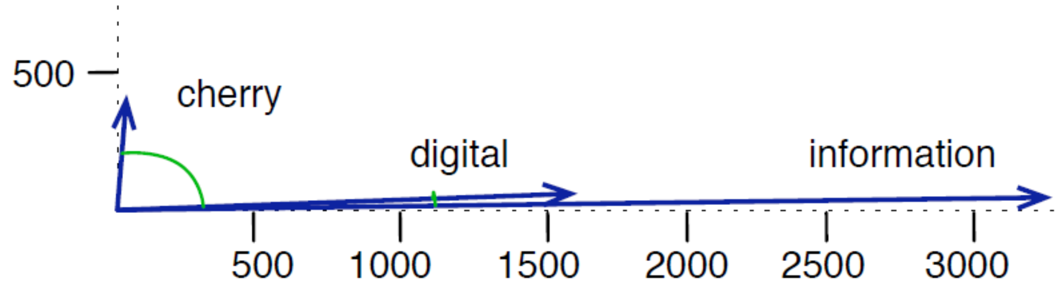
Grok-0, the prototype LLM mentioned in the provided context, is stated to have been trained with 33 billion parameters.

Outline

- Intro to Large Language Models
- RAG pipeline
- Embedding vectors
    - Sentence BERT
- Vector DB
    - Algorithms (HNSW)

Why do we use vectors to represent words?

Given the words “**cherry**”, “**digital**” and “**information**”, if we represent the embedding vectors using only 2 dimensions (X, Y) and we plot them, we hope to see something like this: the angle between words with similar meaning is small, while the angle between words with different meaning is big. So, the embeddings “capture” the meaning of the words they represent by projecting them into a high-dimensional space.



Source: Speech and Language Processing 3rd Edition Draft, Dan Jurafsky and James H. Martin

We commonly use the **cosine similarity**, which is based on the **dot product** between the two vectors.

Word embeddings: the ideas

- Words that are synonyms tend to occur in the same context (surrounded by the same words).
    - For example, the word "*teacher*" and "*professor*" usually occur surrounded by the words "*school*", "*university*", "*exam*", "*lecture*", "*course*", etc..
- The inverse can also be true: words that occur in the same context tend to have similar meanings. This is known as the **distributional hypothesis**.
- This means that to capture the meaning of the word, we also need to have access to its context (the words surrounding it).
- This is why we employ the Self-Attention mechanism in the Transformer model to capture contextual information for every token. The Self-Attention mechanism relates every token to all the other tokens in the sentence.

Word embeddings: the Cloze task

- Imagine I give you the following sentence:
  Rome is the ________ of Italy, which is why it hosts many government buildings.
  Can you tell me what is the missing word?

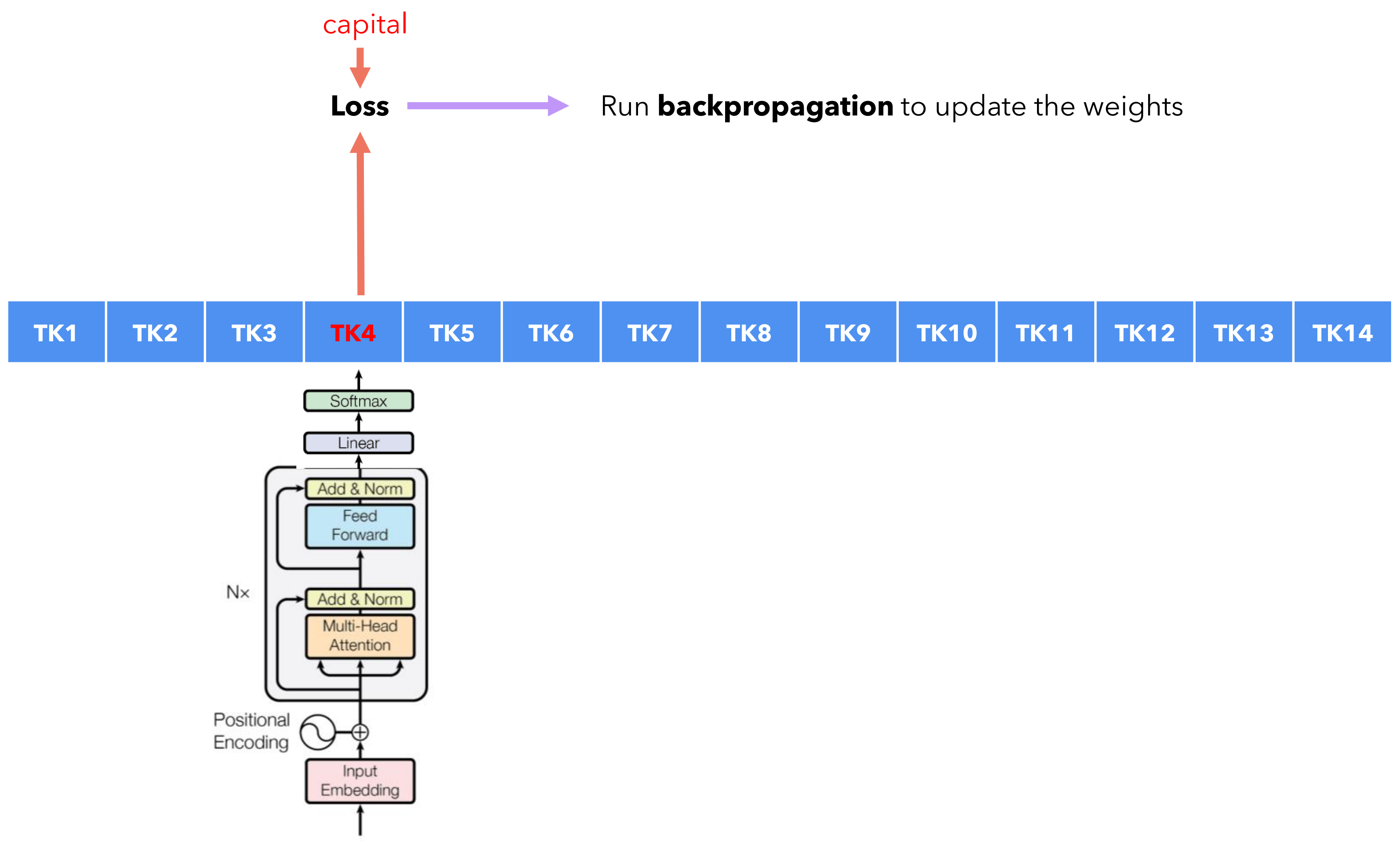
- Of course! The missing word is "*capital*", because by looking at the rest of the sentence, it is the one that makes the most sense.

- This is how we train BERT: we want the Self-Attention mechanism to relate all the input tokens with each other, so that BERT has enough information about the "context" of the missing word to predict it.

How do we train embedding vectors in BERT?

Target (1 token):

Output (14 tokens):



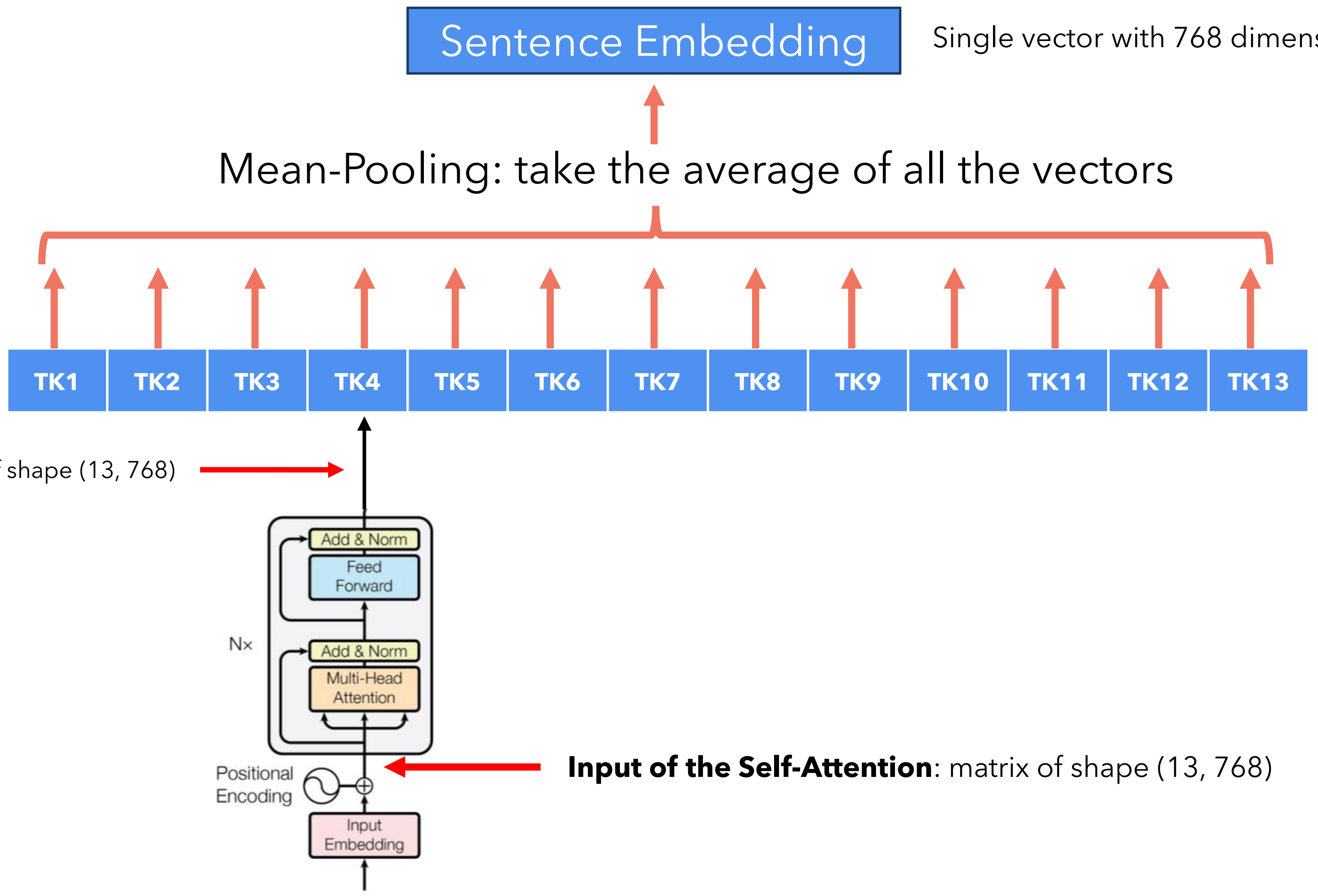
Run **backpropagation** to update the weights

Input (14 tokens): Rome is the [mask] of Italy, which is why it hosts many government buildings.

Sentence Embeddings

- We can use the Self-Attention mechanism also to capture the “meaning” of an entire sentence.
- We can use a pre-trained BERT model to produce embeddings of entire sentences. Let’s see how

Sentence Embeddings with BERT



Single vector with 768 dimensions

Output (13 tokens):

Output of the Self-Attention: matrix of shape (13, 768)

Input of the Self-Attention: matrix of shape (13, 768)

Input (13 tokens):

Our professor always gives us lots of assignments to do in the weekend.

Sentence Embeddings: comparison

- How can we compare Sentence Embeddings to see if two sentences have similar "meaning"? We could use the cosine similarity, which measures the cosine of the angle between the two vectors. A small angle results in a high cosine similarity score.

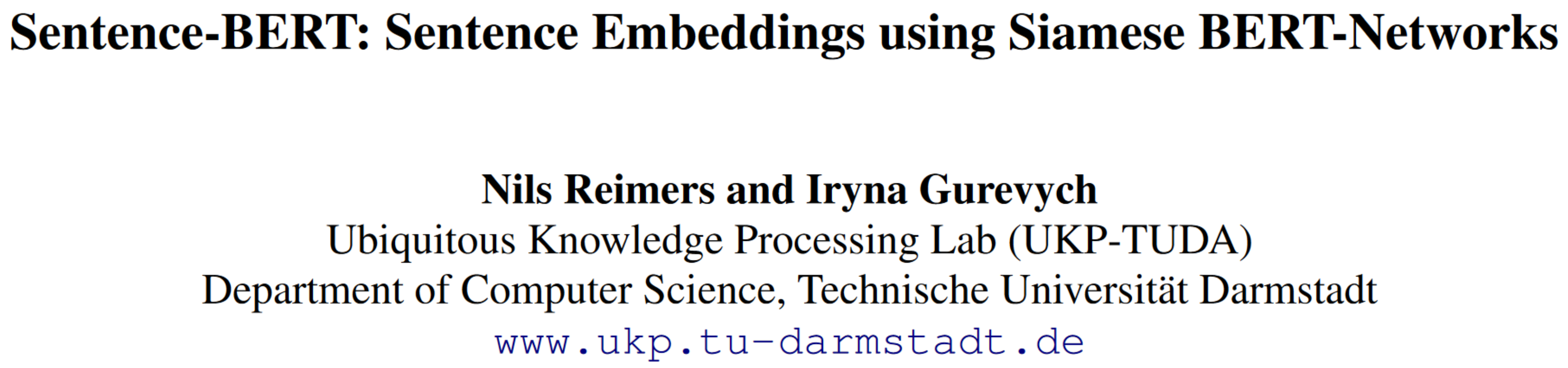
$$\text{cosine similarity} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

- **But there's a problem**: nobody told BERT that the embeddings it produces should be comparable with the cosine similarity, that is, two similar sentences should be represented by vectors pointing to the same direction in space. How can we teach BERT to produce embeddings that can be compared with a similarity function of our choice?

Outline

- Intro to Large Language Models
- RAG pipeline
- Embedding vectors
  - Sentence BERT
- Vector DB
  - Algorithms (HNSW)

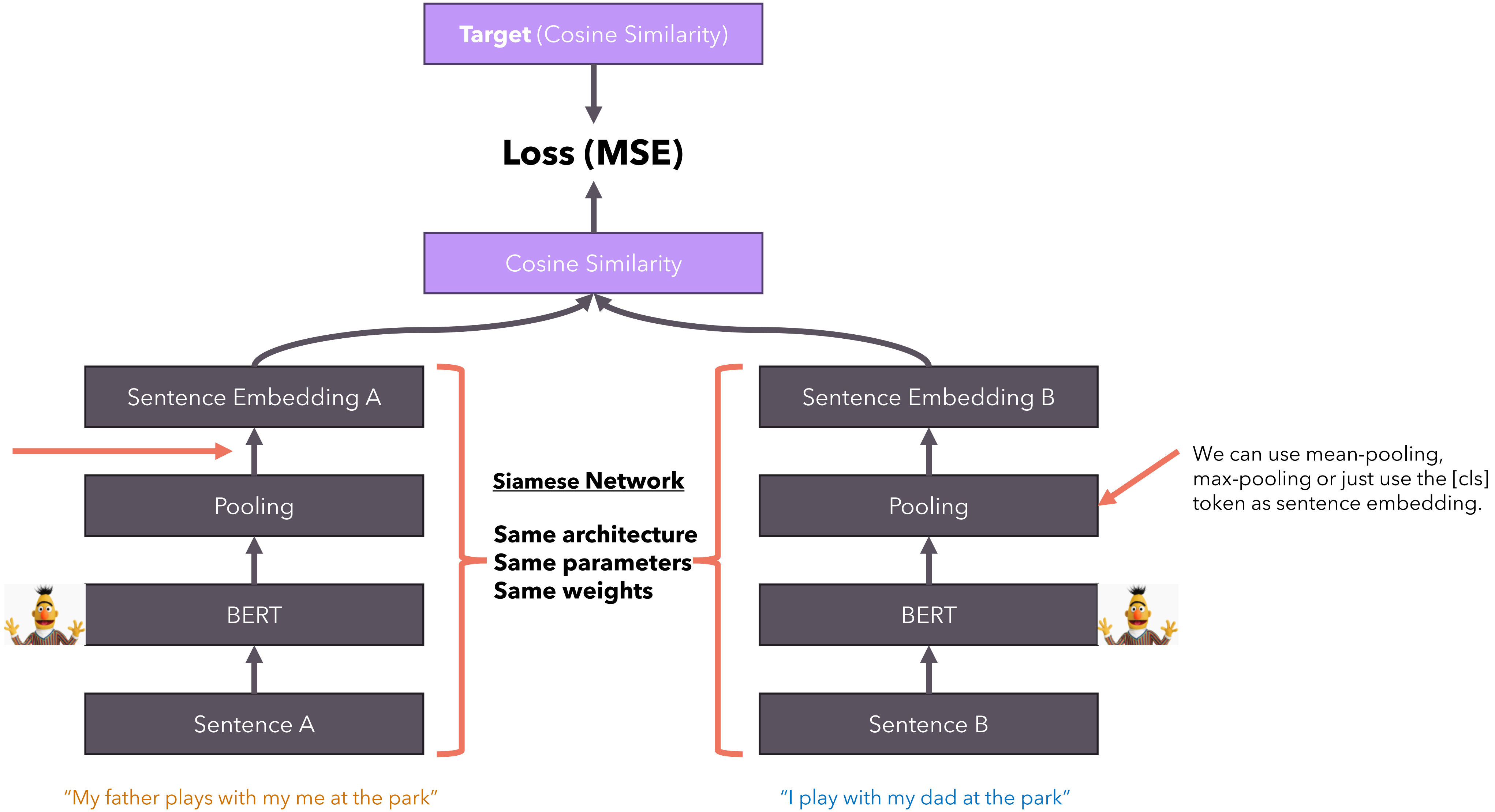
Introducing Sentence BERT



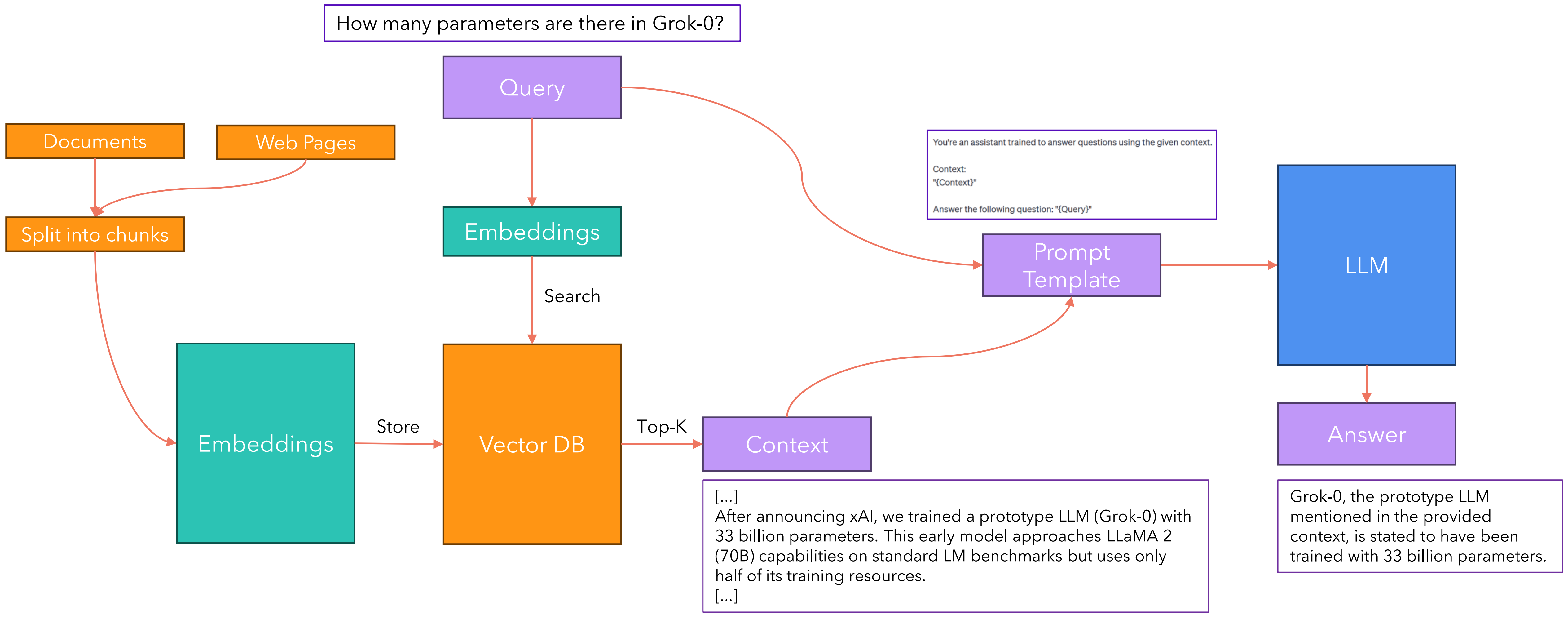
Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks

Nils Reimers and Iryna Gurevych
Ubiquitous Knowledge Processing Lab (UKP-TUDA)
Department of Computer Science, Technische Universität Darmstadt
www.ukp.tu-darmstadt.de

Sentence BERT: architecture


Target (Cosine Similarity)
Loss (MSE)
Cosine Similarity
Sentence Embedding A
Pooling
BERT
Sentence A
Sentence Embedding B
Pooling
BERT
Sentence B
We can apply a linear layer to reduce the size of the embedding vector, for example from 768 to 512
We can use mean-pooling, max-pooling or just use the [cls] token as sentence embedding.
Siamese Network
Same architecture
Same parameters
Same weights
"My father plays with my me at the park"
"I play with my dad at the park"

QA with Retrieval Augmented Generation



You're an assistant trained to answer questions using the given context.

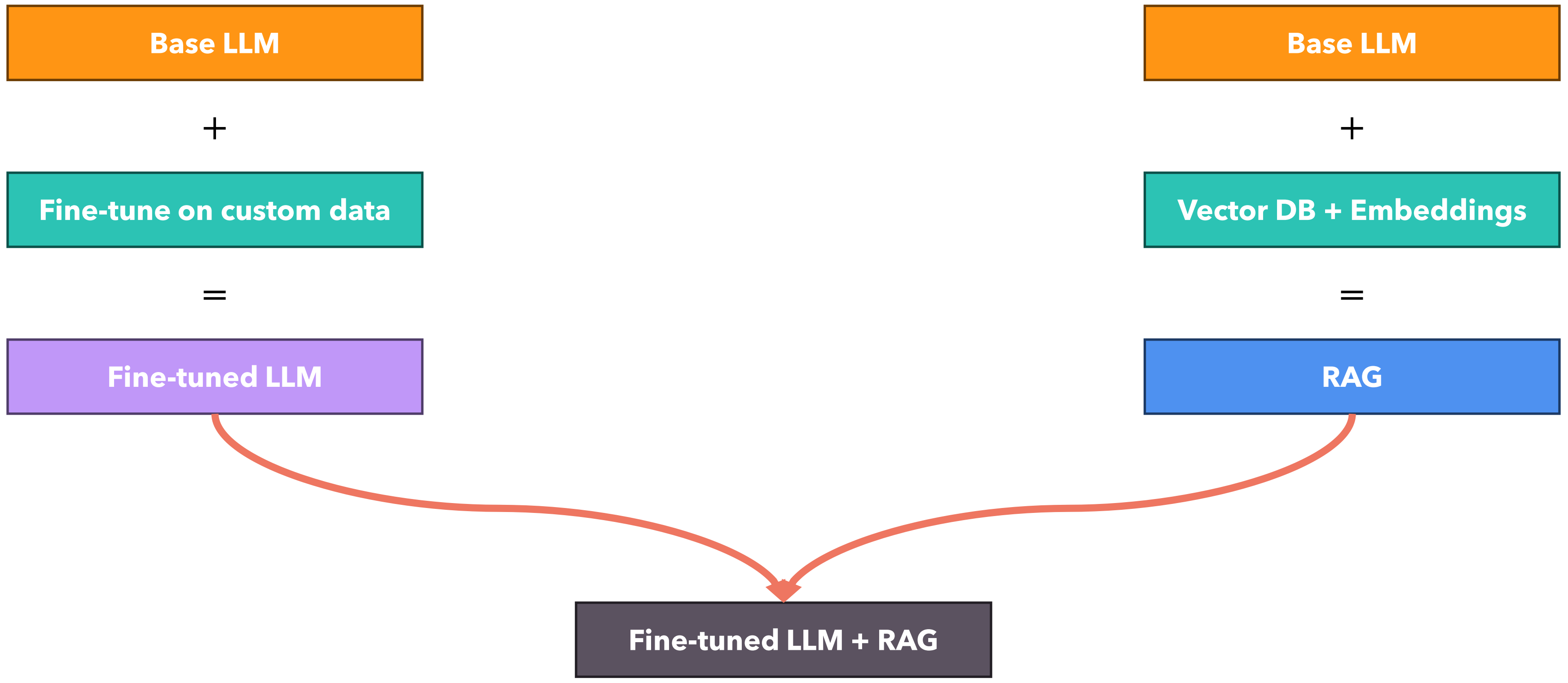
Context:
"{Context}"

Answer the following question: "{Query}"

[...]
After announcing xAI, we trained a prototype LLM (Grok-0) with 33 billion parameters. This early model approaches LLaMA 2 (70B) capabilities on standard LM benchmarks but uses only half of its training resources.
[...]

Grok-0, the prototype LLM mentioned in the provided context, is stated to have been trained with 33 billion parameters.

Strategies to teach new concepts to LLM

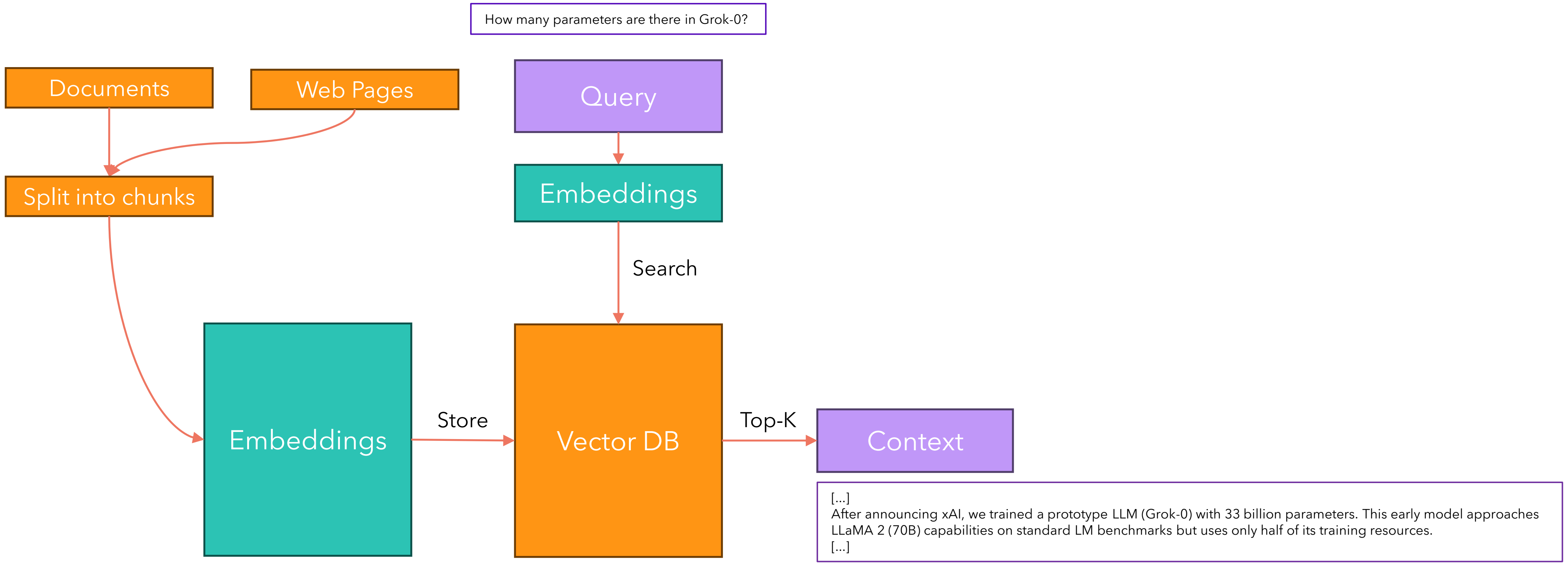


Outline

- Intro to Large Language Models
- RAG pipeline
- Embedding vectors
    - Sentence BERT
- Vector DB
    - Algorithms (HNSW)

Vector DB: introduction

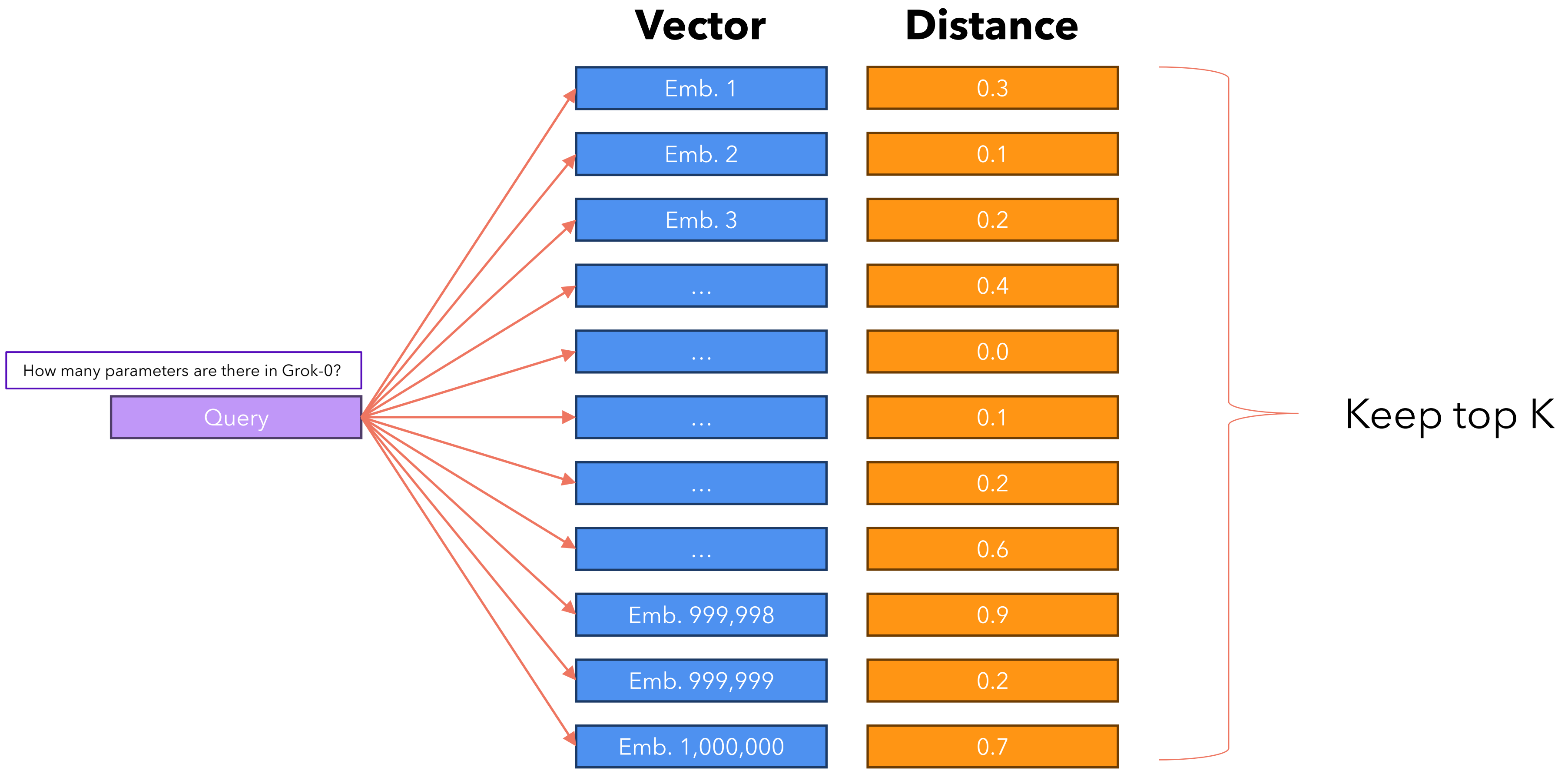
A vector database stores vectors of fixed dimensions (called embeddings) such that we can then query the database to find all the embeddings that are closest (most similar) to a given query vector using a distance metric, which is usually the cosine similarity, but we can also use the Euclidean distance. The database uses a variant of the KNN (K Nearest Neighbor) algorithm or another similarity search algorithm. Vector DBs are also used for finding similar songs (e.g. Spotify), images (e.g. Google Images) or products (e.g. Amazon).



[...]
After announcing xAI, we trained a prototype LLM (Grok-0) with 33 billion parameters. This early model approaches LLaMA 2 (70B) capabilities on standard LM benchmarks but uses only half of its training resources.
[...]

K-NN: a naïve approach

Imagine we want to search for the query in our database: a simple way would be comparing the query with all the vectors, sorting them by distance, and keeping the top K.



If there are N embedding vectors and each has D dimensions, the computational complexity is in the order of **O(N*D), too slow!**

Outline

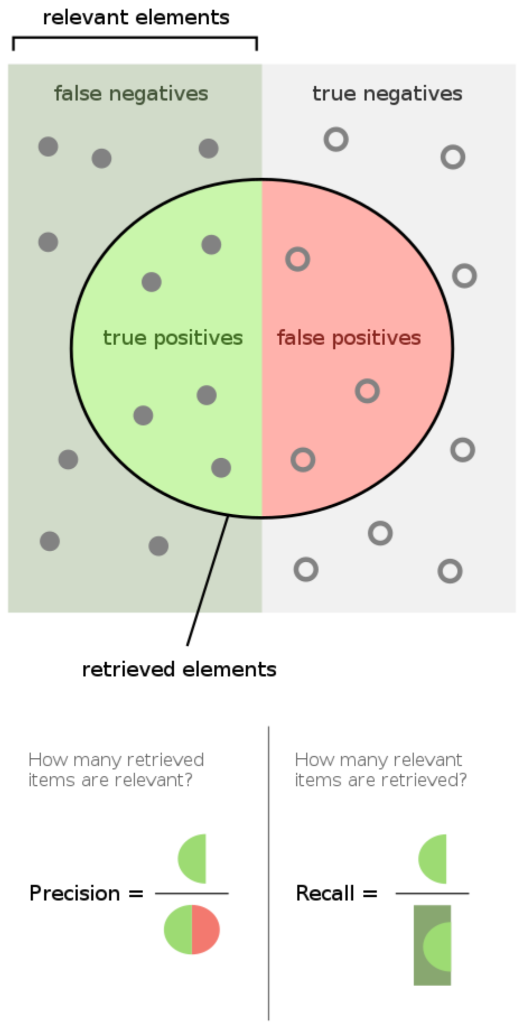
- Intro to Large Language Models
- RAG pipeline
- Embedding vectors
    - Sentence BERT
- Vector DB
    - Algorithms (HNSW)

Similarity Search: let's trade precision for speed

The naïve approach we used before, always produces accurate results, since it compares the query with all the stored vectors, but what if we reduced the number of comparison, but still obtain accurate results with high probability?

The metric we usually care about in Similarity Search is recall.

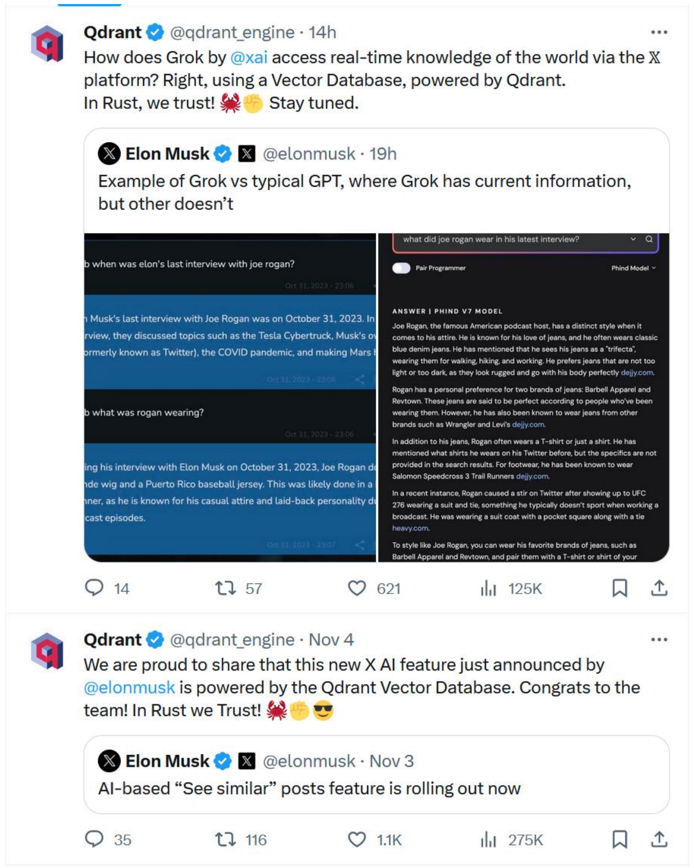
In this video we will explore an algorithm for Approximate Nearest Neighbors, called **Hierarchical Navigable Small Worlds (HNSW)**.



Source: Wikipedia

HNSW in the real world

It is the same algorithm that powers **Qdrant**, the open source Vector DB used by **Twitter's** (X) **Grok** LLM, which can access tweets in real time.



Qdrant @qdrant_engine · 14h

How does Grok by @xai access real-time knowledge of the world via the 𝕏 platform? Right, using a Vector Database, powered by Qdrant.
In Rust, we trust! 🦀✊ Stay tuned.

Elon Musk @elonmusk · 19h

Example of Grok vs typical GPT, where Grok has current information, but other doesn't

b when was elon's last interview with joe rogan?

Oct 31, 2023 · 23:06

Musk's last interview with Joe Rogan was on October 31, 2023. In
rview, they discussed topics such as the Tesla Cybertruck, Musk's o
ormerly known as Twitter), the COVID pandemic, and making Mars

Oct 31, 2023 · 23:06

b what was rogan wearing?

Oct 31, 2023 · 23:06

ing his interview with Elon Musk on October 31, 2023, Joe Rogan d
nde wig and a Puerto Rico baseball jersey. This was likely done in a
nner, as he is known for his casual attire and laid-back personality d
cast episodes.

Oct 31, 2023 · 23:07

what did joe rogan wear in his latest interview?

Pair Programmer   Phind Model

ANSWER | PHIND V7 MODEL

Joe Rogan, the famous American podcast host, has a distinct style when it comes to his attire. He is known for his love of jeans, and he often wears classic blue denim jeans. He has mentioned that he sees his jeans as a "trifecta", wearing them for walking, hiking, and working. He prefers jeans that are not too light or too dark, as they look rugged and go with his body perfectly dejjy.com.

Rogan has a personal preference for two brands of jeans: Barbell Apparel and Revtown. These jeans are said to be perfect according to people who've been wearing them. However, he has also been known to wear jeans from other brands such as Wrangler and Levi's dejjy.com.

In addition to his jeans, Rogan often wears a T-shirt or just a shirt. He has mentioned what shirts he wears on his Twitter before, but the specifics are not provided in the search results. For footwear, he has been known to wear Salomon Speedcross 3 Trail Runners dejjy.com.

In a recent instance, Rogan caused a stir on Twitter after showing up to UFC 276 wearing a suit and tie, something he typically doesn't sport when working a broadcast. He was wearing a suit coat with a pocket square along with a tie heavy.com.

To style like Joe Rogan, you can wear his favorite brands of jeans, such as Barbell Apparel and Revtown, and pair them with a T-shirt or shirt of your

14 · 57 · 621 · 125K

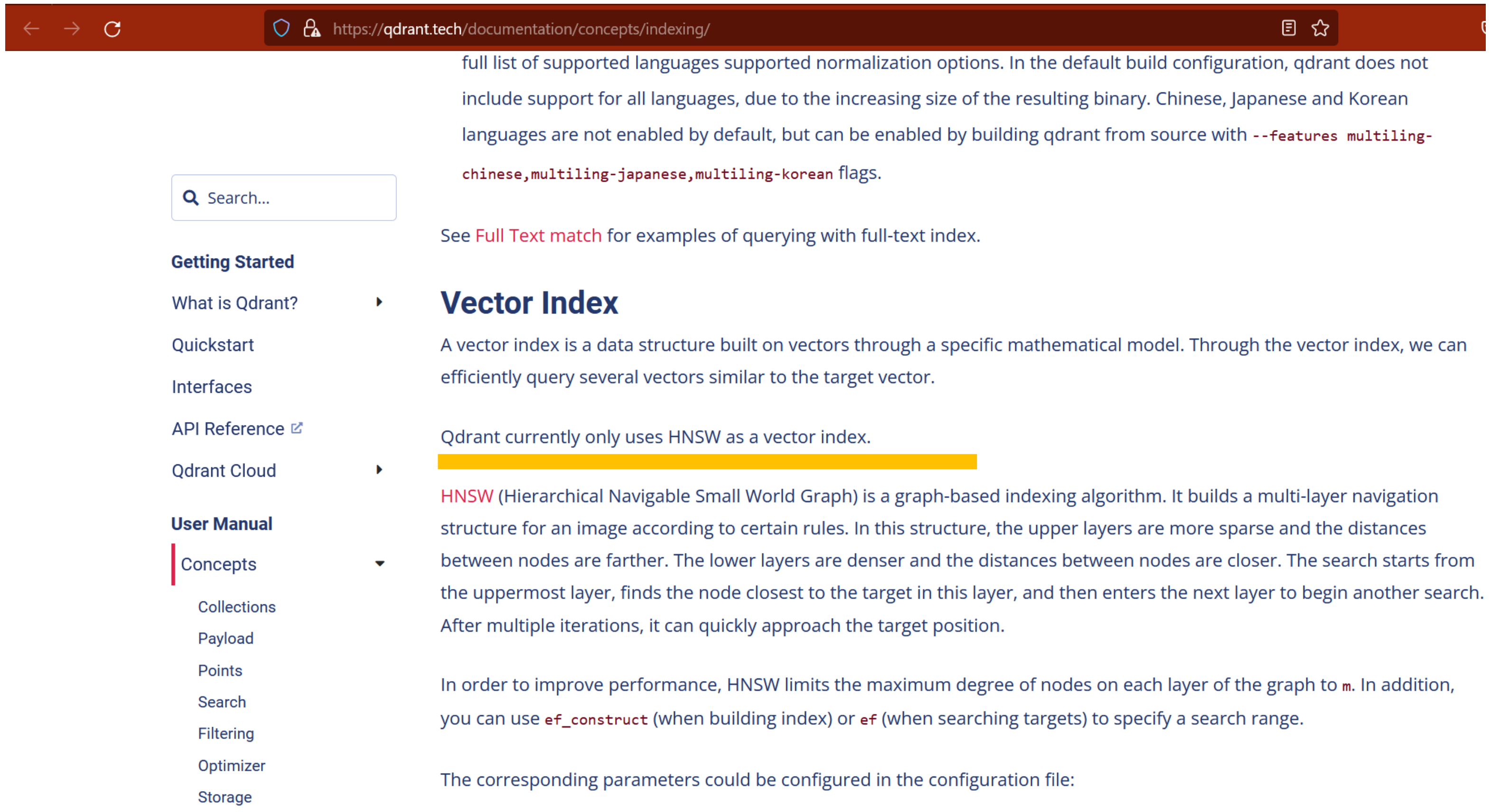
Qdrant @qdrant_engine · Nov 4

We are proud to share that this new X AI feature just announced by @elonmusk is powered by the Qdrant Vector Database. Congrats to the team! In Rust we Trust! 🦀✊😎

Elon Musk @elonmusk · Nov 3

AI-based "See similar" posts feature is rolling out now

35 · 116 · 1.1K · 275K



https://qdrant.tech/documentation/concepts/indexing/

full list of supported languages supported normalization options. In the default build configuration, qdrant does not include support for all languages, due to the increasing size of the resulting binary. Chinese, Japanese and Korean languages are not enabled by default, but can be enabled by building qdrant from source with `--features multiling-chinese,multiling-japanese,multiling-korean` flags.

See Full Text match for examples of querying with full-text index.

Vector Index

A vector index is a data structure built on vectors through a specific mathematical model. Through the vector index, we can efficiently query several vectors similar to the target vector.

Qdrant currently only uses HNSW as a vector index.

HNSW (Hierarchical Navigable Small World Graph) is a graph-based indexing algorithm. It builds a multi-layer navigation structure for an image according to certain rules. In this structure, the upper layers are more sparse and the distances between nodes are farther. The lower layers are denser and the distances between nodes are closer. The search starts from the uppermost layer, finds the node closest to the target in this layer, and then enters the next layer to begin another search. After multiple iterations, it can quickly approach the target position.

In order to improve performance, HNSW limits the maximum degree of nodes on each layer of the graph to `m`. In addition, you can use `ef_construct` (when building index) or `ef` (when searching targets) to specify a search range.

The corresponding parameters could be configured in the configuration file:

HNSW: idea #1

HNSW is an evolution of the **Navigable Small Worlds** algorithm for Approximate Nearest Neighbors, which is based on the concept of **Six Degrees of Separation**.

Milgram's experiment aimed to test the social connections among people in the United States. The participants, who were initially located in Nebraska and Kansas, were given a letter to be delivered to a specific person in Boston. However, they were not allowed to send the letter directly to the recipient. Instead, they were instructed to send it to someone they knew on a first-name basis, who they believed might have a better chance of knowing the target person.

At the end of Milgram's small-world experiment, Milgram found that most of the letters reached the final recipient in five or six steps, creating the concept that people all over the world are all connected by six degrees of separation.

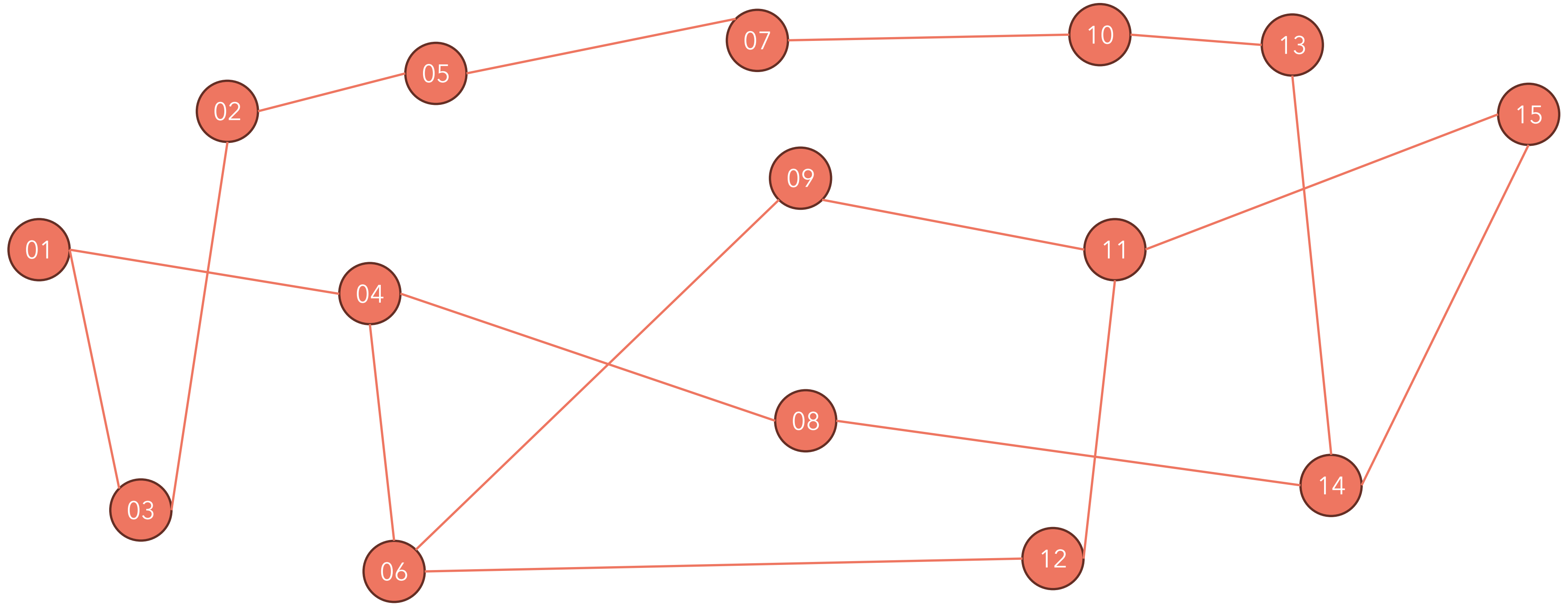
Facebook found in 2016 that its 1.59 billion active users were connected on average by 3.5 degrees of separation: https://research.facebook.com/blog/2016/02/three-and-a-half-degrees-of-separation/

This means that you and Mark Zuckerberg are only 3.5 connections apart!

Navigable Small Worlds

The NSW algorithm builds a graph that – just like Facebook friends – connects close vectors with each other but keeping the total number of connections small. For example, every vector may be connected to up to 6 other vectors (to mimic the Six Degrees of Separation).

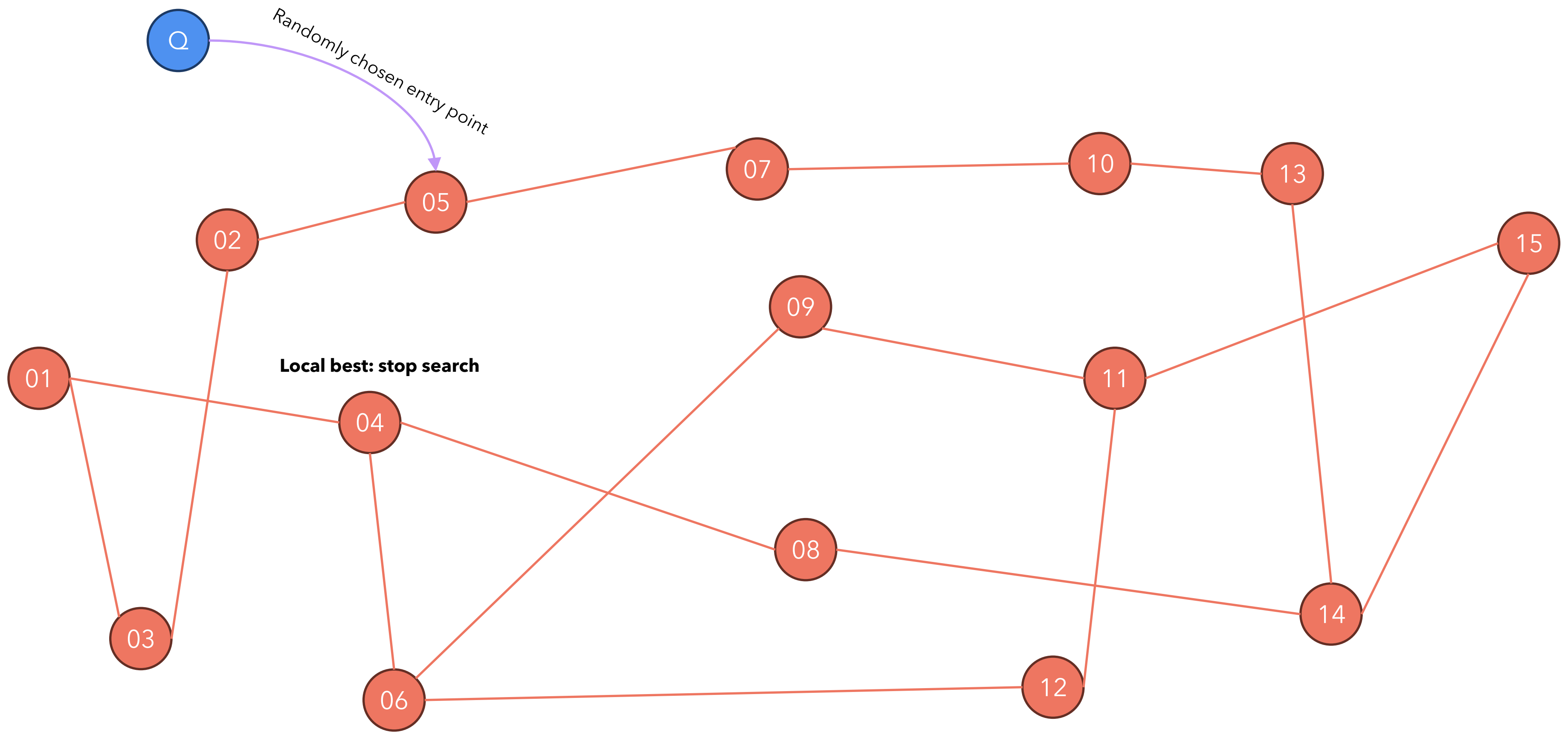
Node	Text
01	[…] The Transformer is a model […]
02	[…] Diagnose cancer with AI […]
03	[…] A transformer-based model […]
04	[…] The Transformer has 6 layers […]
05	[…] An MRI machine that costs 1$ […]
06	[…] The dot-product is a […]
07	[…] Big-Pharma is not so big […]
08	[…] Cross-Attention is a great […]
09	[…] To solve an ODE […]
10	[…] We are aging too fast […]
11	[…] Open-source models like […]
12	[…] MathBERT: a new model […]
13	[…] AI to control aging […]
14	[…] Attention is all you need […]
15	[…] LLaMA 2 has 7B params […]



Navigable Small Worlds: searching for K-NN

Given the following query: "*How many Encoder layers are there in the Transformer model?*"
How does the algorithm find the K Nearest Neighbors?

Node	Text
01	[…] The Transformer is a model […]
02	[…] Diagnose cancer with AI […]
03	[…] A transformer-based model […]
04	[…] The Transformer has 6 layers […]
05	[…] An MRI machine that costs 1$ […]
06	[…] The dot-product is a […]
07	[…] Big-Pharma is not so big […]
08	[…] Cross-Attention is a great […]
09	[…] To solve an ODE […]
10	[…] We are aging too fast […]
11	[…] Open-source models like […]
12	[…] MathBERT: a new model […]
13	[…] AI to control aging […]
14	[…] Attention is all you need […]
15	[…] LLaMA 2 has 7B params […]



We repeat the search with randomly chosen starting points and then keep the top K among all the visited nodes.

Navigable Small Worlds: inserting a new vector

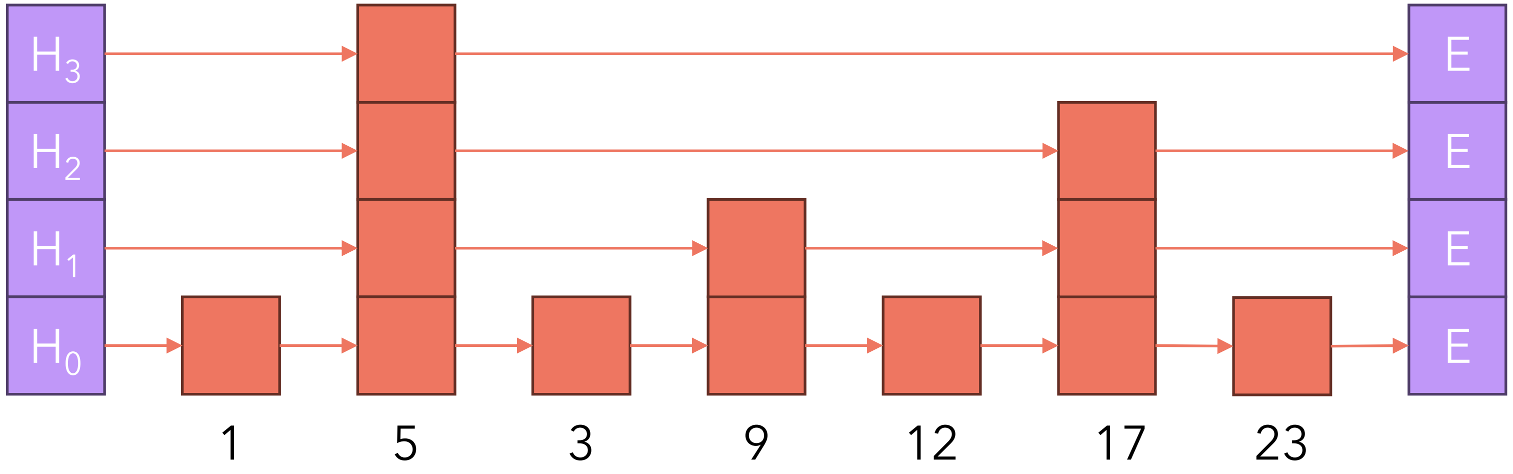
We can insert a new vector by searching the top KNN with the searching algorithm described before and making an edge between the vector and the top K results.

HNSW: idea #2

To go from NSW (Navigable Small Worlds) to HNSW (Hierarchical Navigable Small Worlds), we need to introduce the algorithm behind the data structure known as Skip-List.

The skip list is a data structure that maintains a sorted list and allows search and insertion with an average of $O(\log N)$ time complexity.

Let's search the number **9**

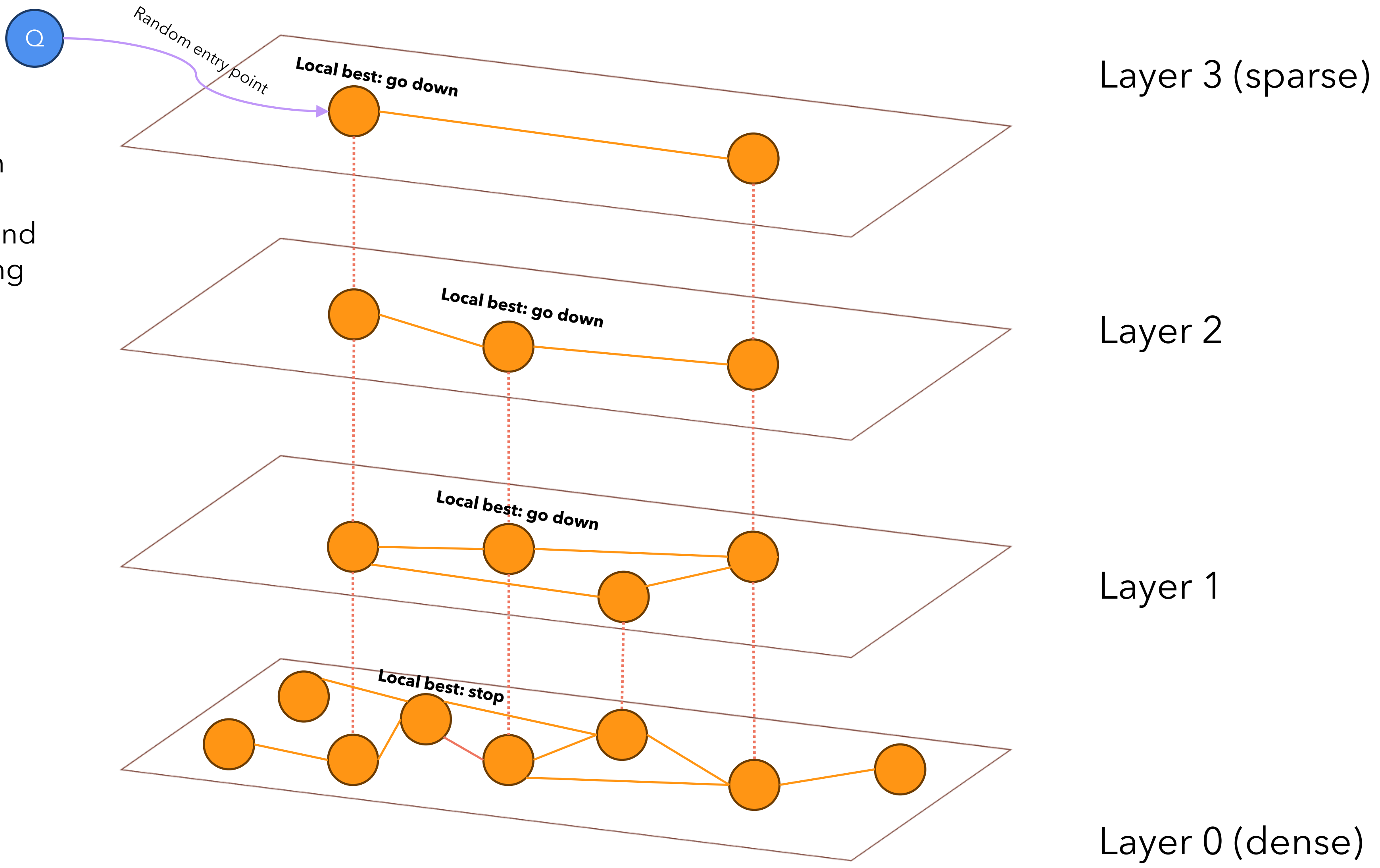


HNSW: Hierarchical Navigable Small Worlds

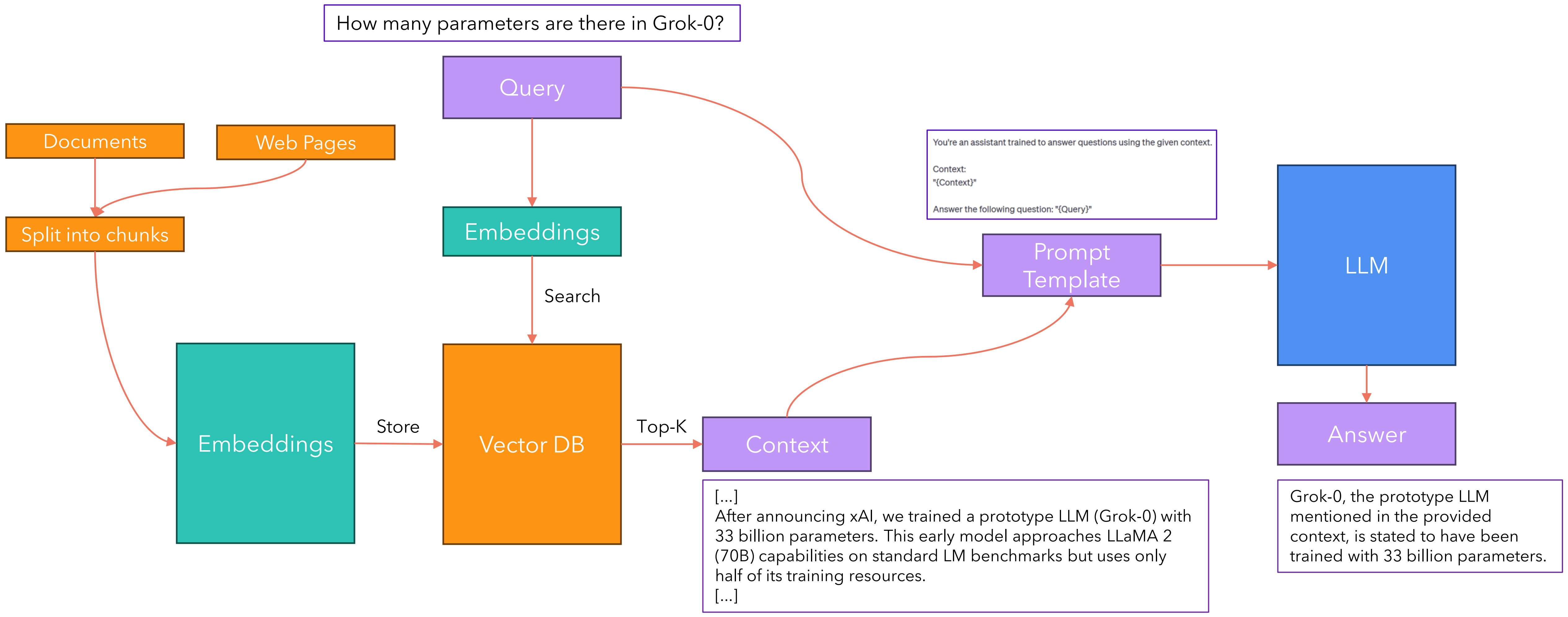
HNSW: Hierarchical Navigable Small Worlds

Let's search!

We repeat the search with randomly chosen starting points (on the top layer) and then keep the top K among all the visited nodes.



QA with Retrieval Augmented Generation



You're an assistant trained to answer questions using the given context.

Context:
"{Context}"

Answer the following question: "{Query}"

[...]
After announcing xAI, we trained a prototype LLM (Grok-0) with 33 billion parameters. This early model approaches LLaMA 2 (70B) capabilities on standard LM benchmarks but uses only half of its training resources.
[...]

Grok-0, the prototype LLM mentioned in the provided context, is stated to have been trained with 33 billion parameters.

Thanks for watching!
Don't forget to subscribe for more amazing content on AI and Machine Learning!