Pak Angels Essential Gen-AI Training
Module 2
Demystifying GenAI

Abdullah Zahid
Silicon Valley, CA

A little bit about me

- Beaconhouse School System – Faisalabad - O & A Levels
- Graduated from the University of Miami with a Bachelor's in CS and Economics
- Worked as a ML engineer at my first job focusing on NLP using Google's BERT
- Currently working as a full-stack SWE while getting my MSc in Deep Learning
- Enjoy chatting about new ideas and advancements in tech, fitness or finance
- Feel free to hit me up
  - LinkedIn (https://www.linkedin.com/in/m-abdullah-zahid)
  - IG: @az_98

Module 2 Agenda

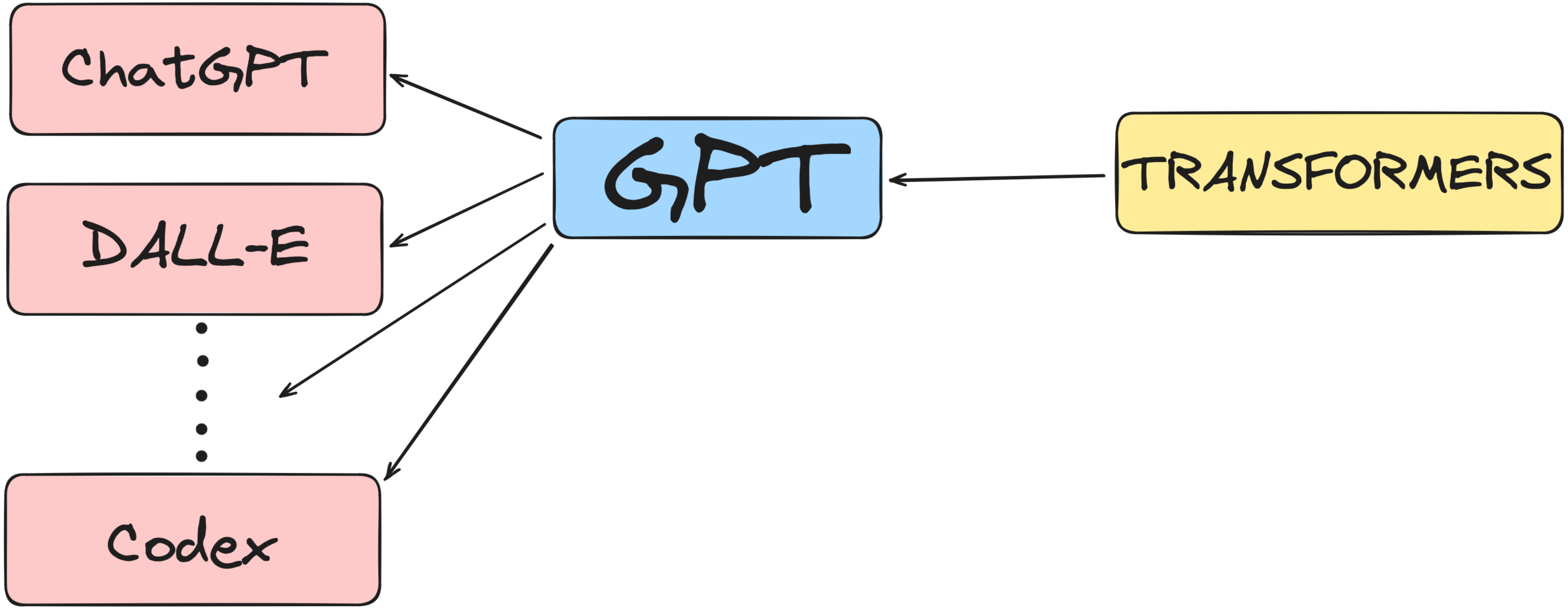
Opening up the black-box that is GPT

Preprocessing inputs for fine-tuning

Diving into models that generate text, code, images, and voice.

Recognizing and avoiding common pitfalls

Making <material> generation better through RAGs and Vector DBs


ChatGPT
DALL-E
Codex
GPT
TRANSFORMERS

What is "GPT"?

Generative

Produces output like text (including code), image, and voice

Pre-trained

Already trained on large corpuses including millions of webpages, books etc

GPT3 was trained on all of Wikipedia amongst other things

Leaves room for additional training (fine-tuning) for specific tasks like classifying news articles into categories

Transformer

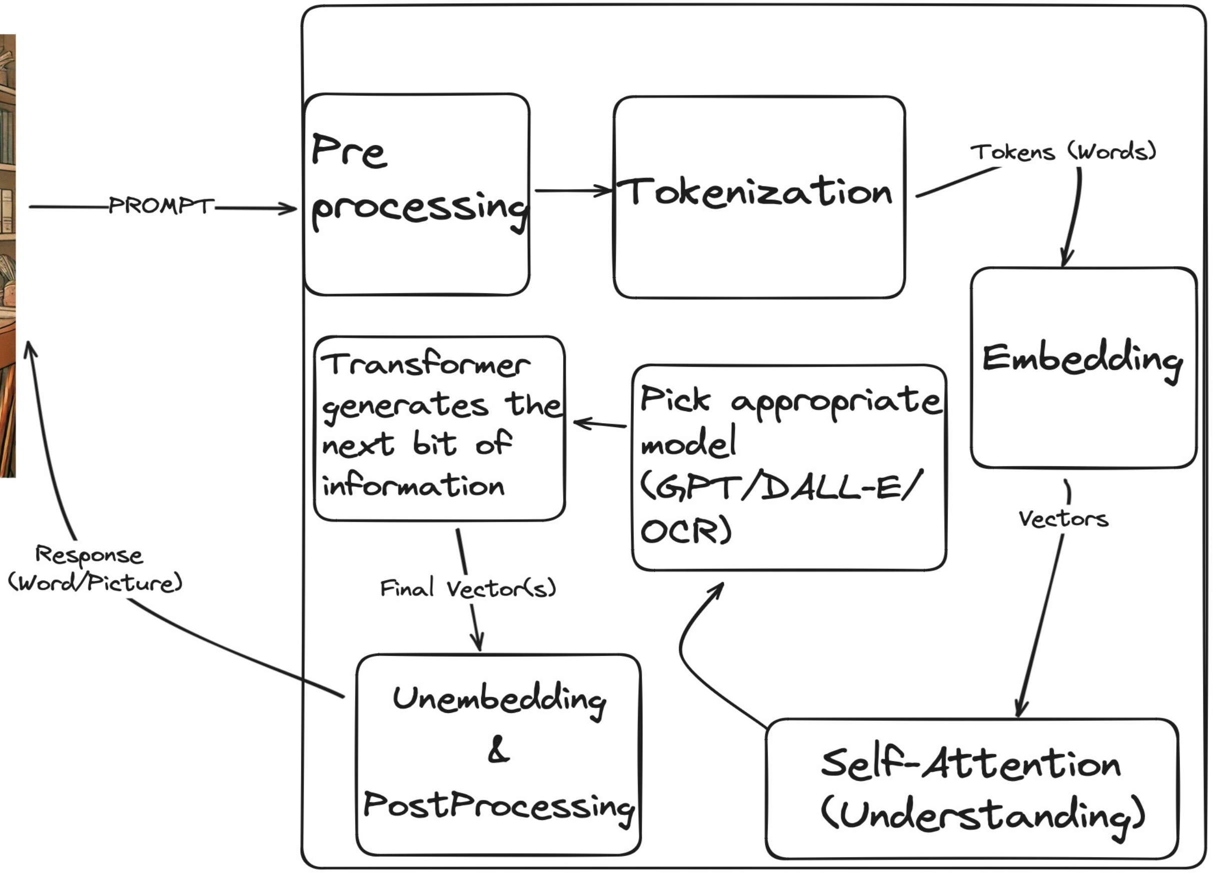
(attention is all you need)

The core neural network architecture that every generative model is built on

Iteratively calculates the probability of what the next piece of information might be based on current inputs like the next note in Beethoven's 5th Symphony or the next word in Fee Fi Fo ___

CHATGPT

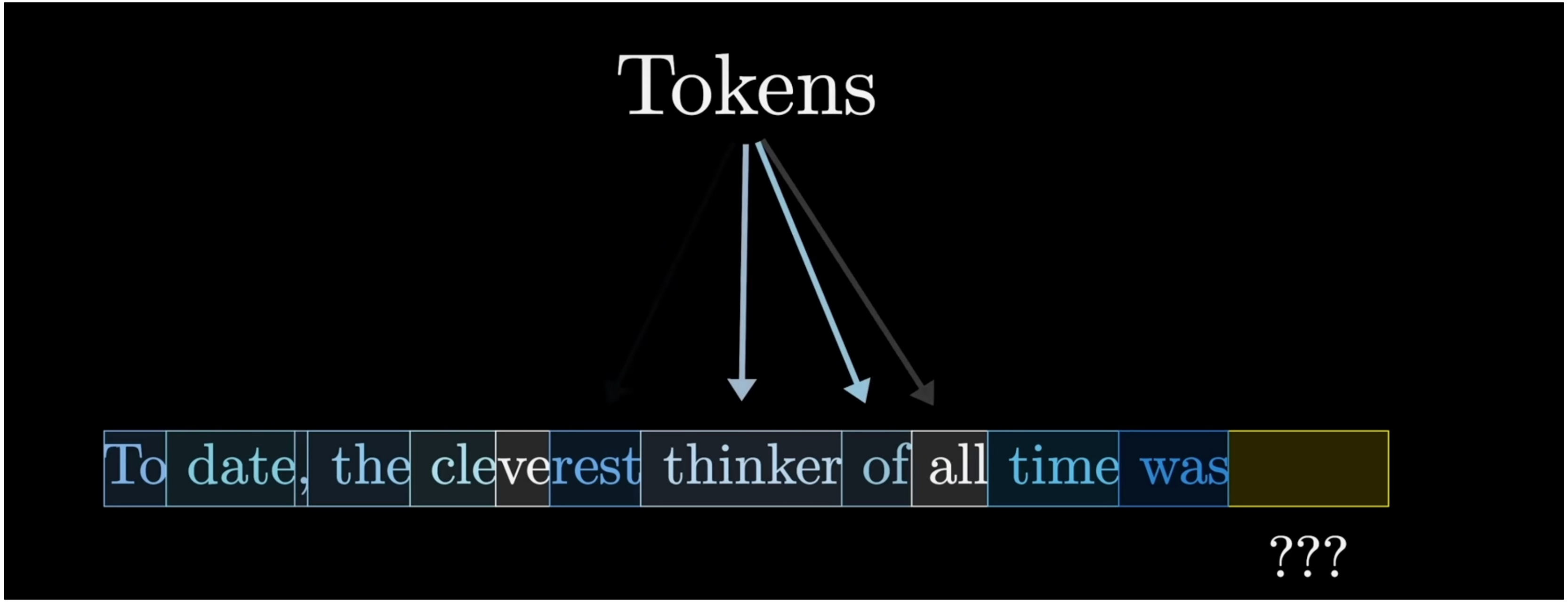
ChatGPT


PROMPT
Pre processing
Tokenization
Tokens (Words)
Embedding
Vectors
Self-Attention (Understanding)
Pick appropriate model (GPT/DALL-E/ OCR)
Transformer generates the next bit of information
Final Vector(s)
Unembedding & PostProcessing
Response (Word/Picture)

What happens to a “cleaned” Input?

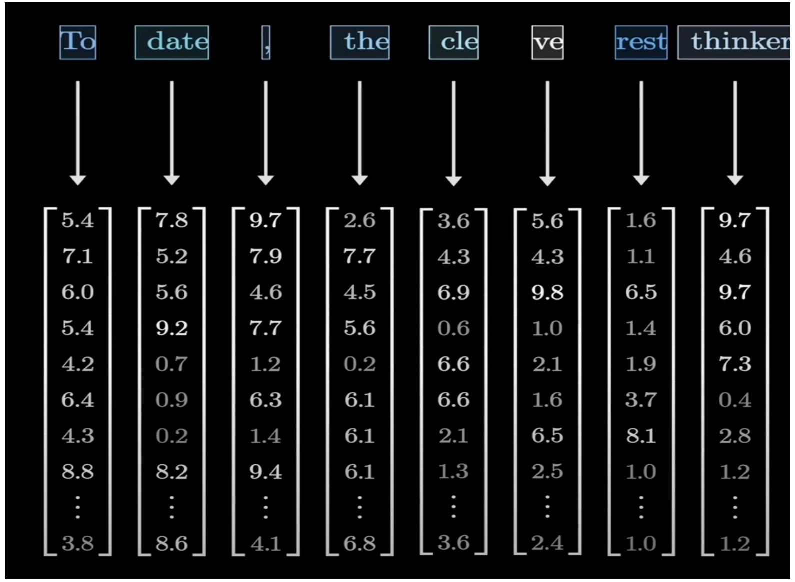
Tokenizing

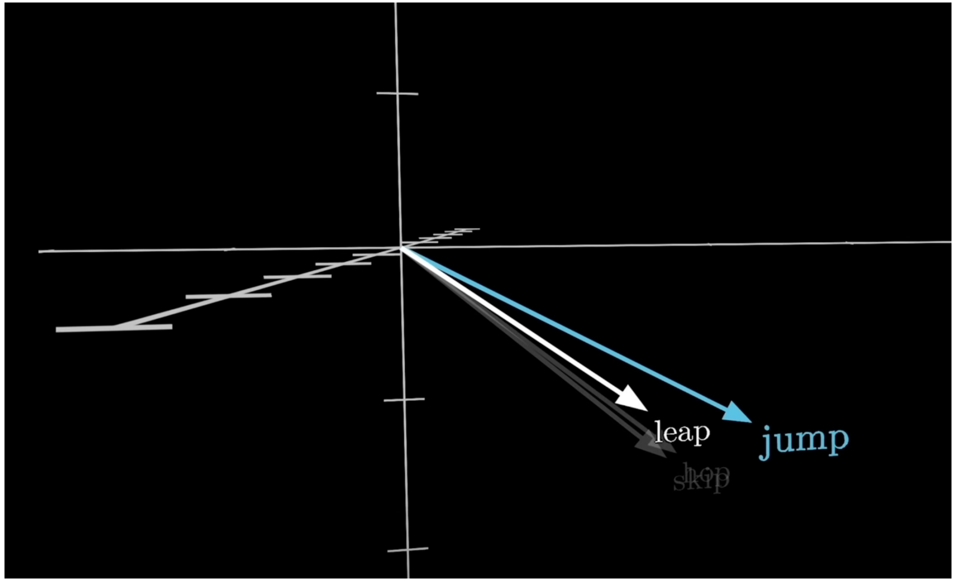
- Breaking inputs into chunks
- Words/characters for text, sound snippets for voice, pixels groups for image



Embedding

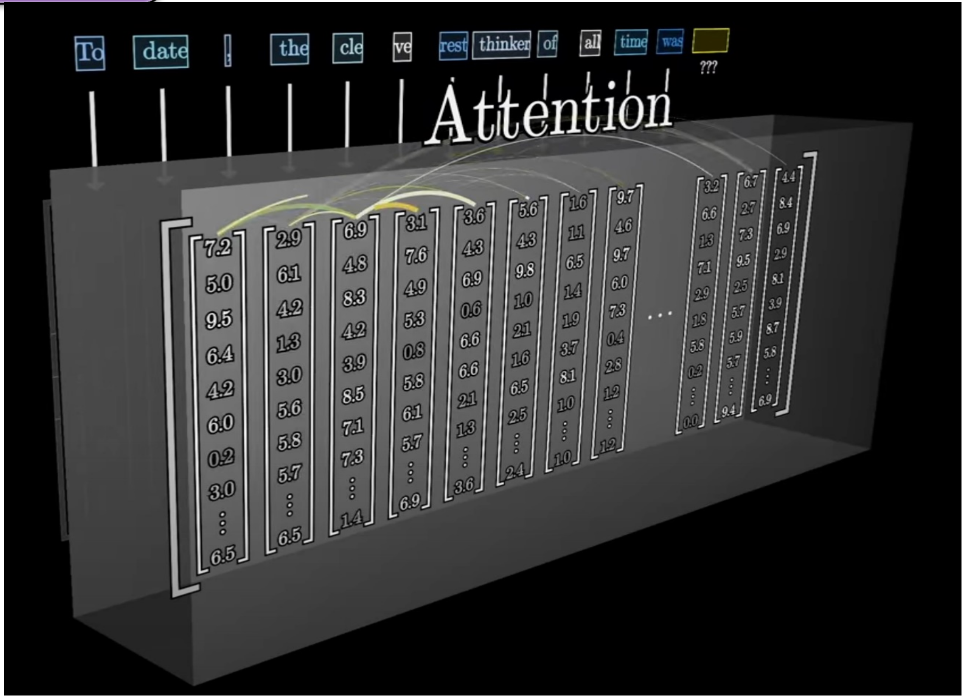
- Encoding chunks into vectors in an N-dimensional vector space
- Similar chunks end up as vectors close to each other (words with similar meaning, pixels that have similar RGB values)
- Can do vector operations like add, subtract, dot, and cross products
- [Vector for “King”] – [Vector for “Man”] + [Vector for “Woman”] = [Vector for “Queen”]



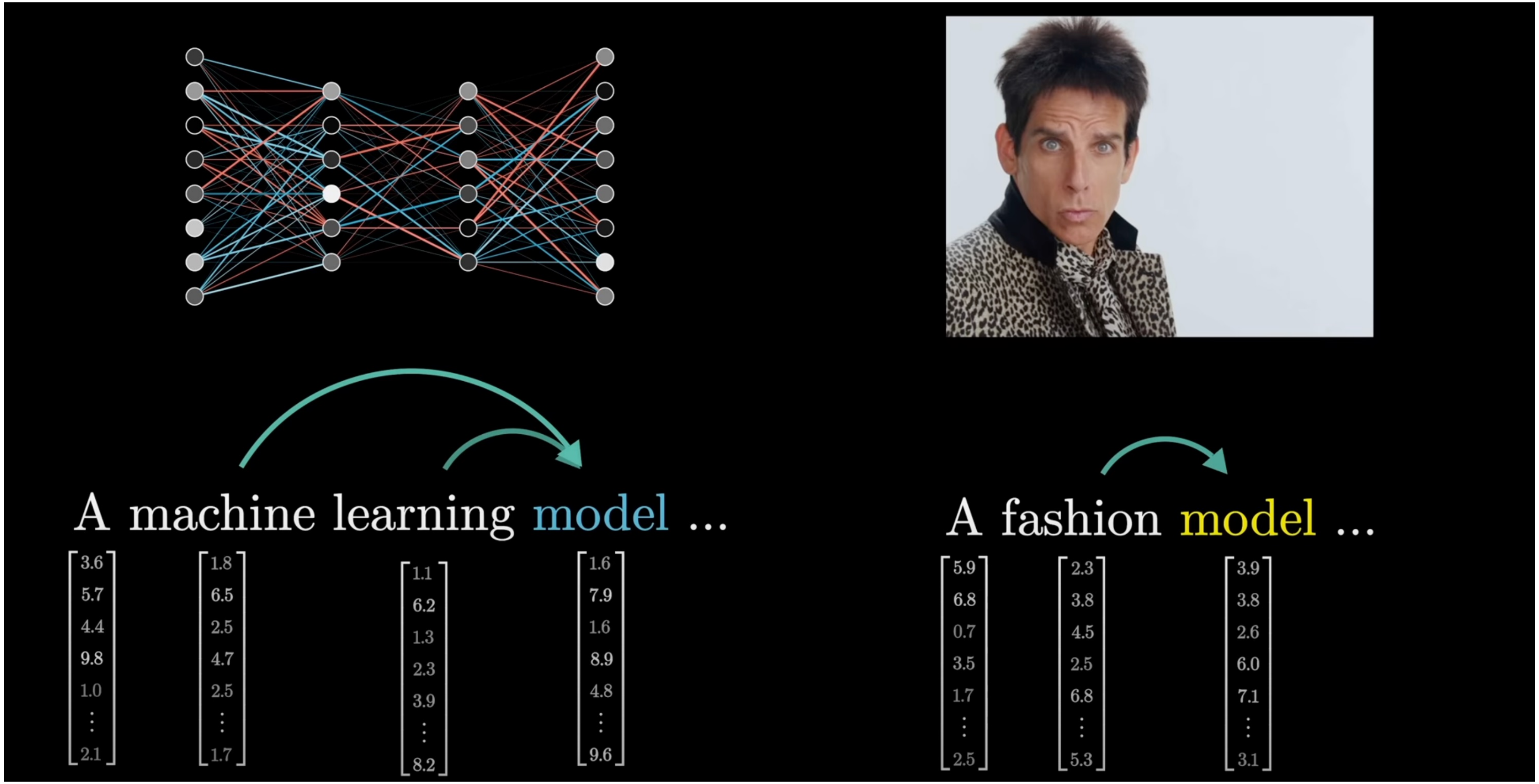


Attention

- The vectors update each others' values based on information from each other
- E.g. a ”bank” vector might get updated to represent the side of a river or a piggy bank based on what information comes before and after the word “bank”



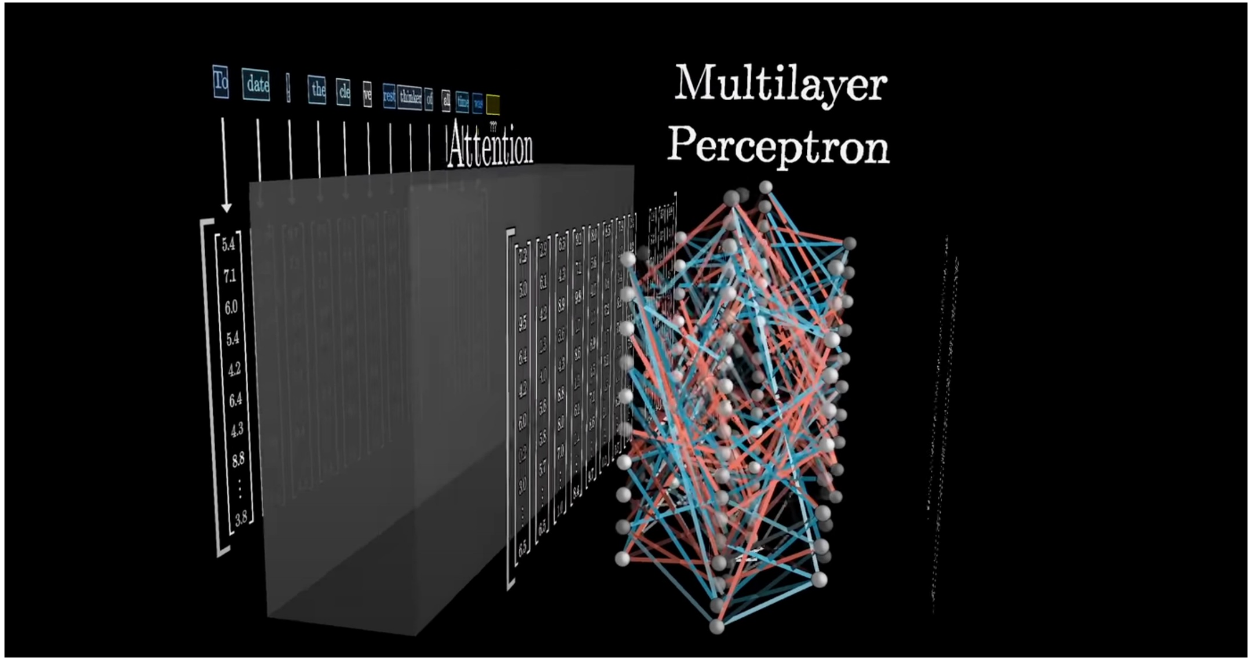
Attention is all you need



MLPs

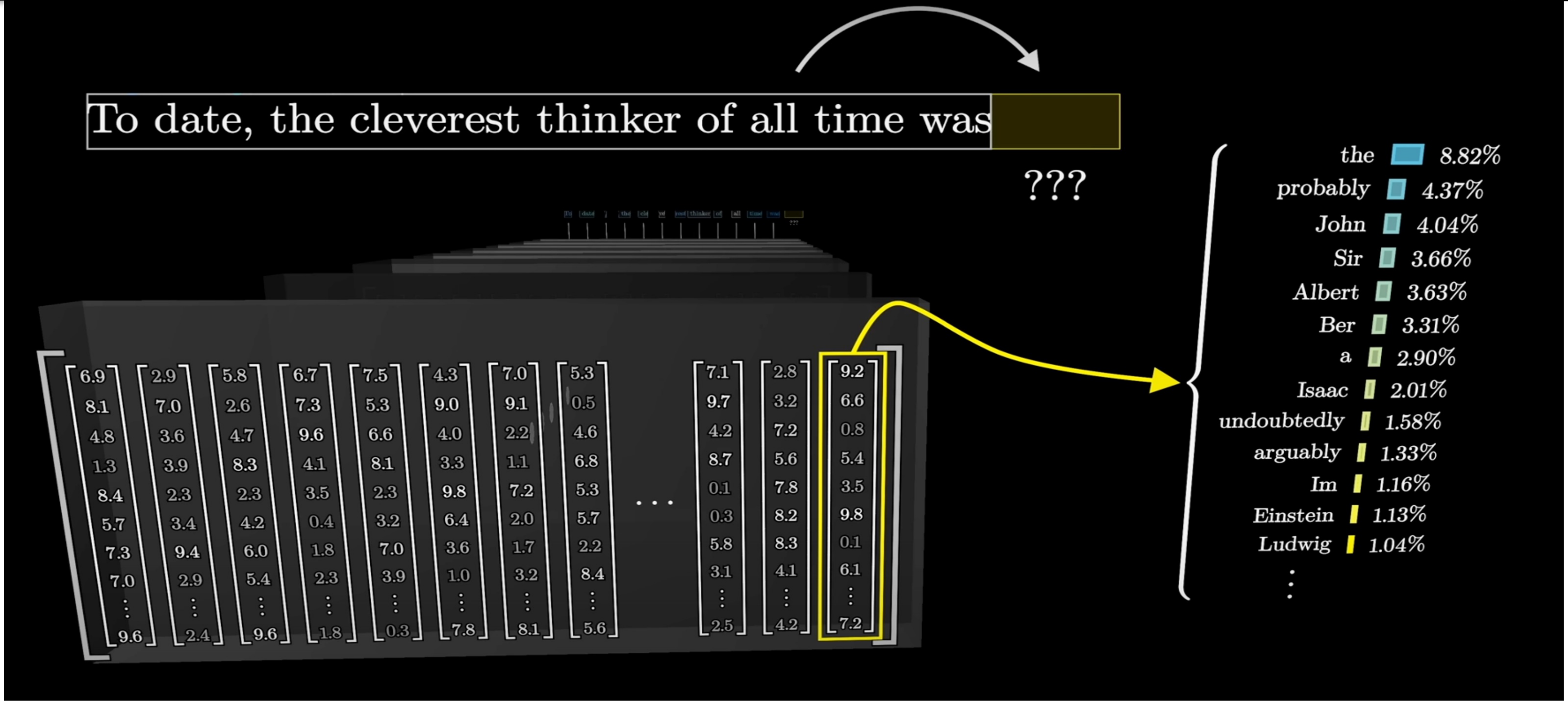
(Multi-layer Perceptrons)

- No more talking between vectors
- Just a many-layered neural network that updates it's weights and biases based on the input vectors and previous layers
- Answers questions like "is it English", "is this part of the pixel fur from an animal" etc



Unembedding

- After many rounds of Attention and MLPs, a final vector is produced which is the official next bit of information
- Now we reverse the embedding into a useful token that is the same as the input tokens like words or pixels



?

"Clean Inputs" – TEXT - Stop words

- Human languages, like English, have a lot of filler words that are grammatic but don't add any meaning to the context, these are called stop words
- Some examples in English are "a", "the", "an"
- Removing these stop words will lower tokens used and will help you get a more bang for your buck when fine-tuning a Large Language Model
- Utilities to remove them: NLTK, a python package, is the most used way to remove them, the full list of stop words can be found here.

STOPWORDS DEMO

“Clean Inputs” – TEXT - Stemming

- A way to remove prefixes and suffixes from words to distill meaning and “stem” each word to its basic forms
- Doing this standardizes words and allows models to be sure of each word’s meaning
- Simple examples are: “walks” becomes “walk”, “retrieval” becomes “retrieve”
- Done using algorithms that, thankfully, we don’t have to develop
- The most common one is PorterStemmer, which uses rules such as removing common suffixes like “ed”, “ing”, converting plurals to singulars (cars to car), and reducing adjectives to their base forms (happier to happy)

“Clean Inputs” – TEXT – Stemming CONTD.

- Stemming is useful in information retrieval, search, and data mining
- Use it wisely! Be cautious of ”over stemming” – All stemmers have false positives where a word might be reduced to a form that is meaningless, like “university” might be reduced to “univers”
- Stemming words can make them lose their contextual meaning e.g. agreement -> agree
- Compound words are poorly handled (e.g “Whiteboard”)
- Stemmers usually don’t handle proper nouns like names well.

“Clean inputs” – Text - Lemmatization

- More sophisticated than stemming: better contextual awareness
- Reduces words to their dictionary basic form (lemma) so ”university” will remain as such
- “Running” and “ran” will both be reduced to “run”
- This helps models group certain words (and sentences) together
- Helps reduce overhead and reduce dimensionality of vectors that words produce
- Helps info retrieval: “Best coffee” will also retrieve results for “good coffee”
- Done using NLTK as well, (WordNetLemmatizer) (code demo)

“Clean inputs” – Text – Lemmatization CONTD.

- Beware! Lemmatization is not king.
- Context can still be lost. E.g. “running" and "run" have different nuances in certain contexts, which might be lost after lemmatization.
- It is computationally expensive and slow because of the complex linguistic analysis
- Rules are specific to each language, requiring different approaches and tools for different languages.
- Ambiguous words can lead to incorrect lemmatization. For example, "bats" can be the plural of "bat" (the animal) or a form of the verb "to bat".
- Different lemmatization tools might produce inconsistent results for the same text. (so use the same tool please)

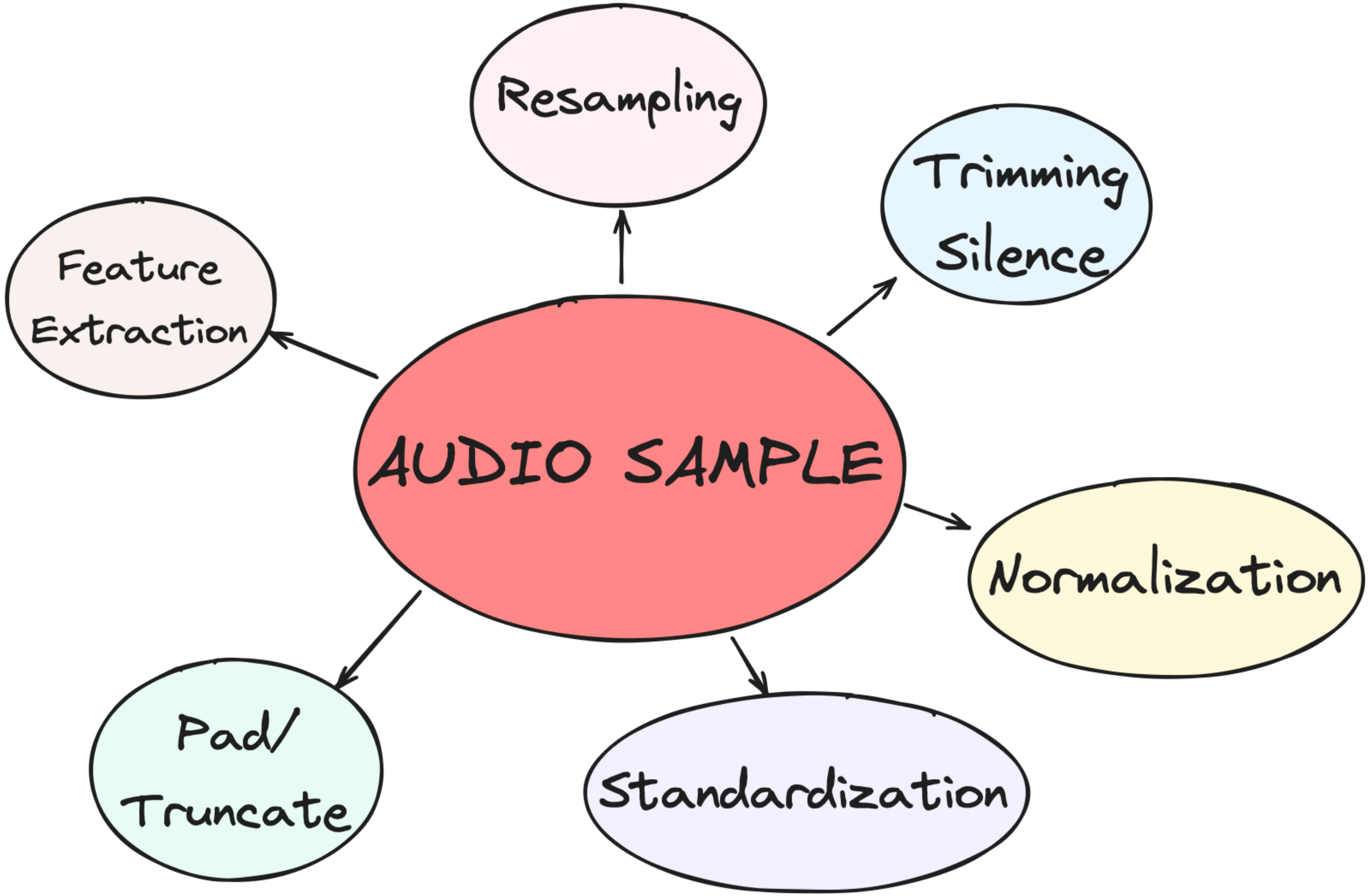
STEMMING & LEMMATIZATION DEMO

"Clean inputs" – TEXT other techniques

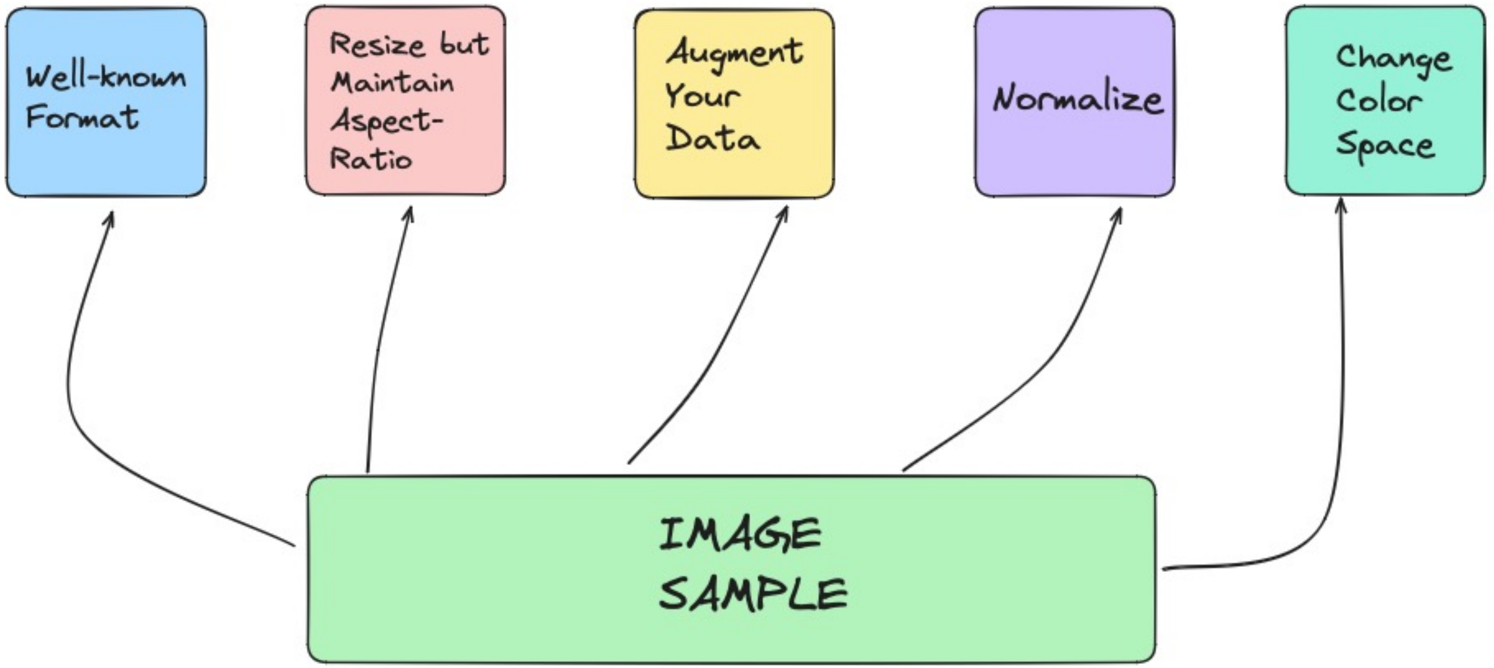
- Regex (regular expressions) are your best friend!
- Pure Text models don't usually do well with numbers and special characters {!,@,#,$,%,...} so it's wise to remove them
- Regex helps you search (and remove) certain patterns in strings
- "[^a-zA-Z\s]" will match all the English characters in a string
- Handle contractions ("don't" -> "do not" use py contractions)

REGEX DEMO

Cleaning inputs – Voice



Cleaning inputs - images



Why pre-process audio & Images?

- **Consistency**: Ensures that the data fed into the model is consistent in terms of format, scale, and length.
- **Efficiency**: Reduces the size of the dataset by removing unnecessary parts (like silence), which speeds up training.
- **Feature Quality**: Extracts meaningful features from the raw audio that are more useful for the model to learn from.
- **Improved Model Performance**: By normalizing and standardizing the data, the model can learn more effectively, leading to better performance.

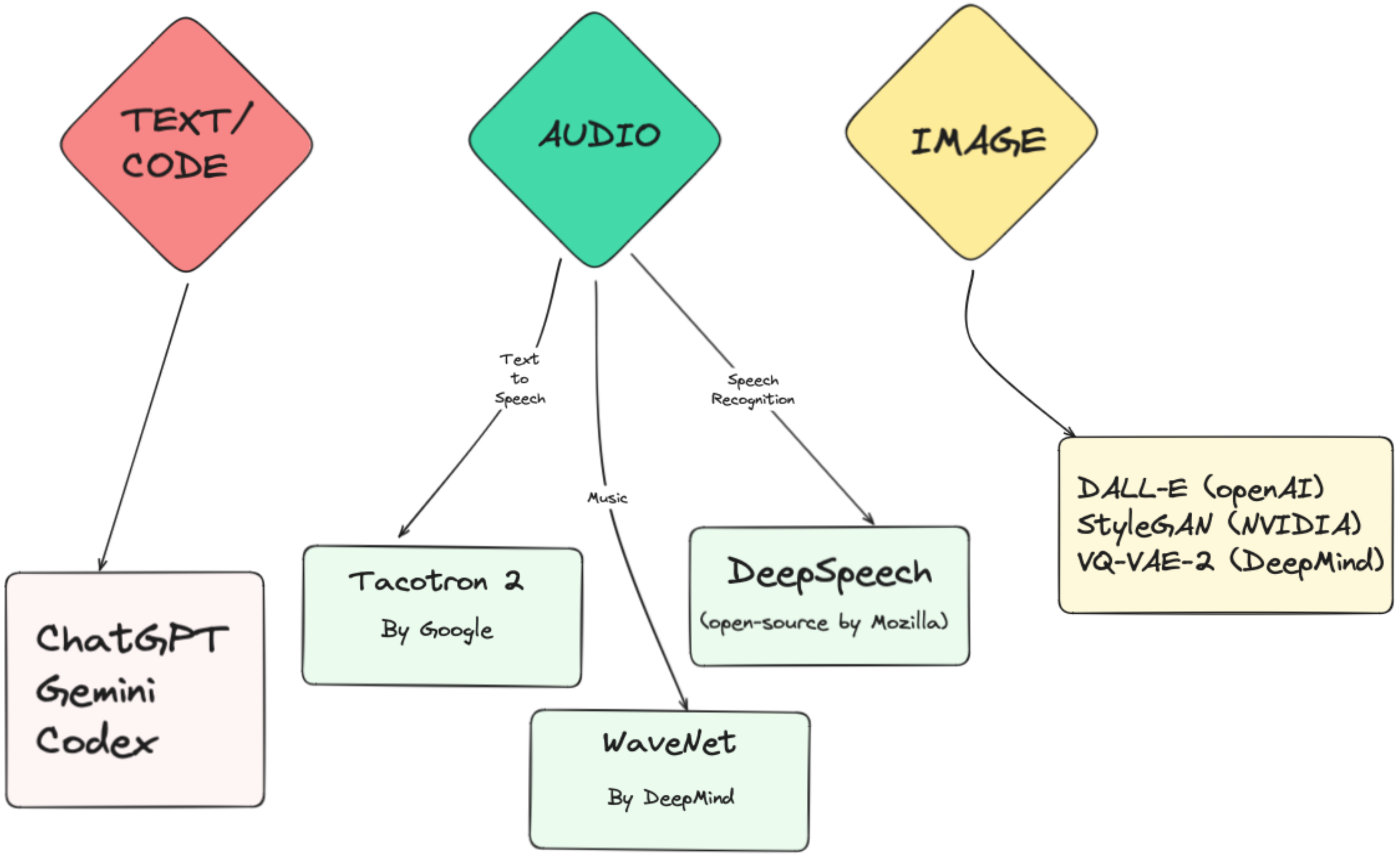
- **Normalizing** prevents bias and helps stabilize the training process as large input values cause large parameter updates on the model. E.g. in audio, normalization prevents the model from recognizing “loudness” as a feature (obviously if that is something we are aiming for, then the audio shouldn’t be normalized).

- **Standardization** produces mean-centered data which is great for the model as it prevents large mean values from dominating the dataset and show the true patterns in the data. It also creates uniform variance which increases performance of the model by making sure each feature contributes equally to the training.

- Most models use some form of gradient descent where normalization & standardization help immensely in letting the model converge faster by avoiding uneven gradients

?

What model for what purpose ?



Let’s talk models – TEXT/CODE

- OpenAI’s Generative Text Models and LLMs took the world by storm but what’s going on underneath? Hint: it’s just matrix math & probability
- ChatGPT is relatively new, but it’s built on top of Google’s Transformer architecture where each word (or token) is “understood” by assigning it to a vector space
- Pre-training is paramount for these models where they are fed huge corpuses of text data with no obvious goal (unsupervised learning), books, webpages, open-source-code
- Self-attention is the backbone of transformers, and weighs the importance of different words in the context of a sentence by deciding similarity between one word and others in the sentence (if they exist close to each other in the vector space)
- Transformers don’t keep track of the order of words so LLMs assign positional encoding to each word

Let’s talk models – TEXT/CODE - GENERATION

- After the input is “understood”, the model needs to be able to predict the next word
- Most early models used “**greedy sampling**” which just picks the next highest-probability word. This works and is super fast but leads to repeated text
- **Beam search** improves upon this by keeping track of multiple (beam width) possible sequences at each step, selecting the top sequences based on their cumulative probability. This produces better quality of text, but is computationally more expensive and can still lead to text generation
- **Top-k sampling** improves this further by only considering ”k” number of words and choosing the higher probability one to sample, this includes randomness and diversity which reduces repetitive text but it needs more work on the developer’s side to tune what the optimal “k” is
- **Top-p sampling**, unlike top-k, considers the smallest set of words whose cumulative probability > p. This is more flexible than top-k and allows for better coherence while maintaining diversity.

Text-gen algos DEMO

?

Let's talk models – Audio
Speech Recognition

- DeepSpeech (Mozilla)
  - An open-source speech-to-text engine, using an RNN based model trained on large datasets.
  - **Pros:** High accuracy, open-source, pre-trained models available.
  - **Resource Intensive:** requires significant computation to infer
  - **High latency**: real-time speech recognition may experience latency depending on the hardware.
  - **Language Limitation**: performance may vary significantly across different languages and accents, requiring language-specific fine-tuning.

Let’s talk models – Audio

Text-To-Speech

- Tacotron 2 (Google)
  - A neural network architecture for speech synthesis directly from text.
  - Produces natural-sounding speech
  - End-to-end system, simplifies our job by letting us just enter normal text
  - Highly customizable for different languages and voices
  - Similar to DeepSpeech, requires significant computation resources
  - Needs powerful hardware for near-real-time inference
  - High-quality and large datasets are required for optimal performance

Let's talk models – Audio
Music Generation

- **WaveNet** (DeepMind)
  - A generative model for raw audio waveforms.
  - Generates highly realistic audio, including music and speech.
  - Super versatile
  - High performing in the realm of clarity and naturalness
  - Similar drawbacks as other new age audio models

Let’s talk models – Images – DALL-E

- DALL-E uses the Transformers architecture that is adapted for consuming text and producing images
- Pre-trained on tons of images and corresponding text so it understands the relationship between the two
- It uses the same positional word encoder as other GPT models, but the difference lies in how it uses these as a condition to generate corresponding image tokens, where each token is a part of the image
- The image tokens are then decoded and pieced together in a way that forms a coherent image (e.g. pixels with similar values are grouped together)
- DALL-E is now integrated into ChatGPT+

Let’s talk models – Images – GANs (StyleGAN)

- GANs are made up of 2 neural networks that work against each other, hence ”adversarial”
- One, called the generator, creates fake data (like images) from random noise
- The other, called the discriminator, evaluates and tries to distinguish between the fake and real data
- Both are trained to do better in their own purposes and one getting better feeds the other one to get better as well
- GANs are used to generate realistic synthetic data for data augmentation, making art, or realistic photographs

Let’s talk models – Images – VAE (VQ-VAE-2)

- VAEs (Variation AutoEncoders) generate new data by understanding the structure of the input data
- There are 3 key components:
  - An Encoder which , like CNNs, compresses an image into a lower dimensional representation and produces a mean and variance of the data in a latent space
  - A Latent Space is where this compressed data lives and is operated on
  - A Decoder that then “uncompresses” this data into its original form

- The model is trained to reconstruct the data with minimal loss (using reconstruction loss and KL divergence)
- After that, new data can be generated by sampling from the latent space and passing it through the decoder.
- Handy for creating images, compressing data into a lower dimension, and detecting anomalies by comparing the input to the learned distribution

?

Common Pitfalls – Text models

- Overfitting, the model learns to generate text that is too specific on the training data and fails to generalize on new prompts. It is wise to augment your data and use early stopping
- Lack of diversity in the input data can lead to repetitive text. Use top-k/top-p sampling to avoid this
- Fine tune the model to domain-specific text data and use context windowing to maintain relevance
- Train on content-filtered, diverse, and balanced data to avoid generating biased/offensive text. Also regularly monitor output

Common pitfalls – Audio models

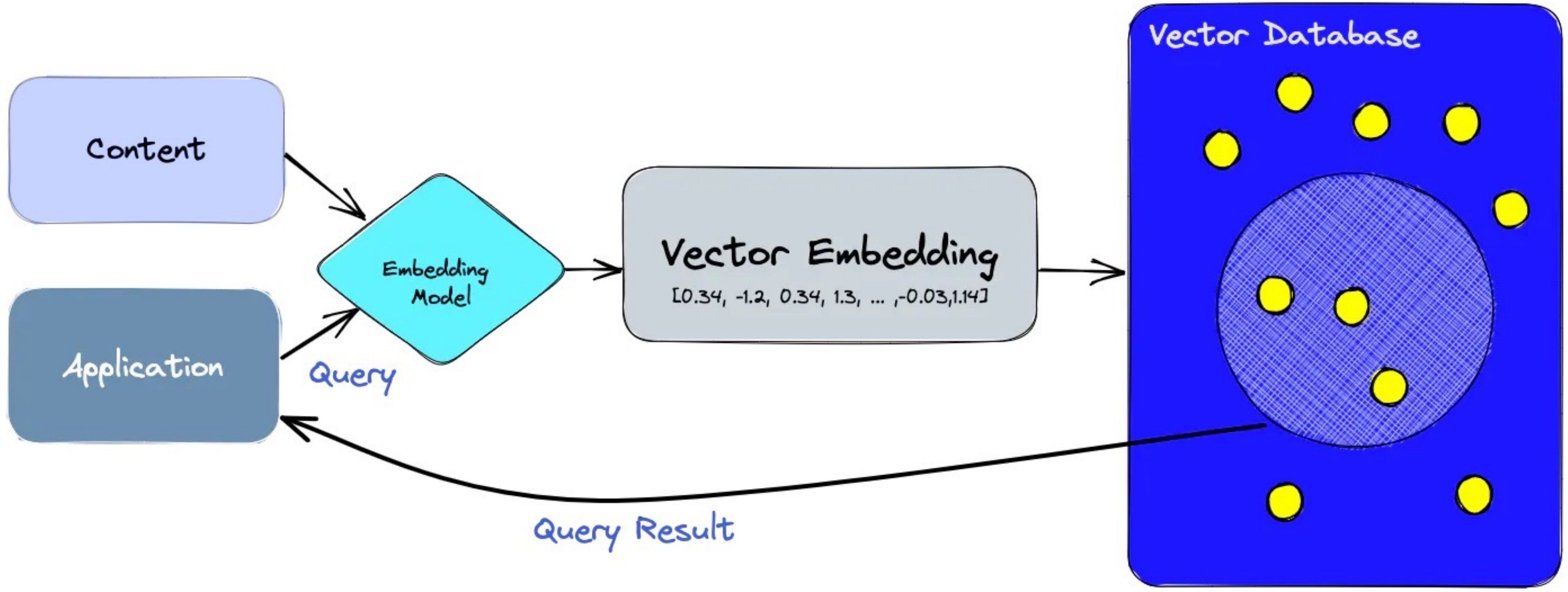
- Use high quality audio data, and post-process the output audio to make it coherent and natural
- Use varied accents and intonations to avoid mispronunciation and increase coherency
- Incorporate emotional speech datasets, and use models that can modulate pitch, tone, and pace to convey emotions effectively.
- Ensure datasets are anonymized and obtain necessary permissions. Implement privacy-preserving techniques in model training.

Common pitfalls – Image models

- Avoid blurriness by using advanced GAN architectures like StyleGAN, increase training data quality, and apply image post-processing techniques.
- These models can also suffer from a lack of variety in generation so implement techniques like minibatch discrimination, unrolled GANs, or improve the training process with regularization.
- Data bias is a problem for these models as well, so use diverse datasets (e.g. people with different ethnic backgrounds, age, features etc). Use data augmentation (flipping, rotating etc).
- Carefully tune hyperparameters, apply techniques like gradient penalty, and use robust optimization algorithms. This will prevent the model from generating poor quality images.

?

Make retrieval better - Vector DBs

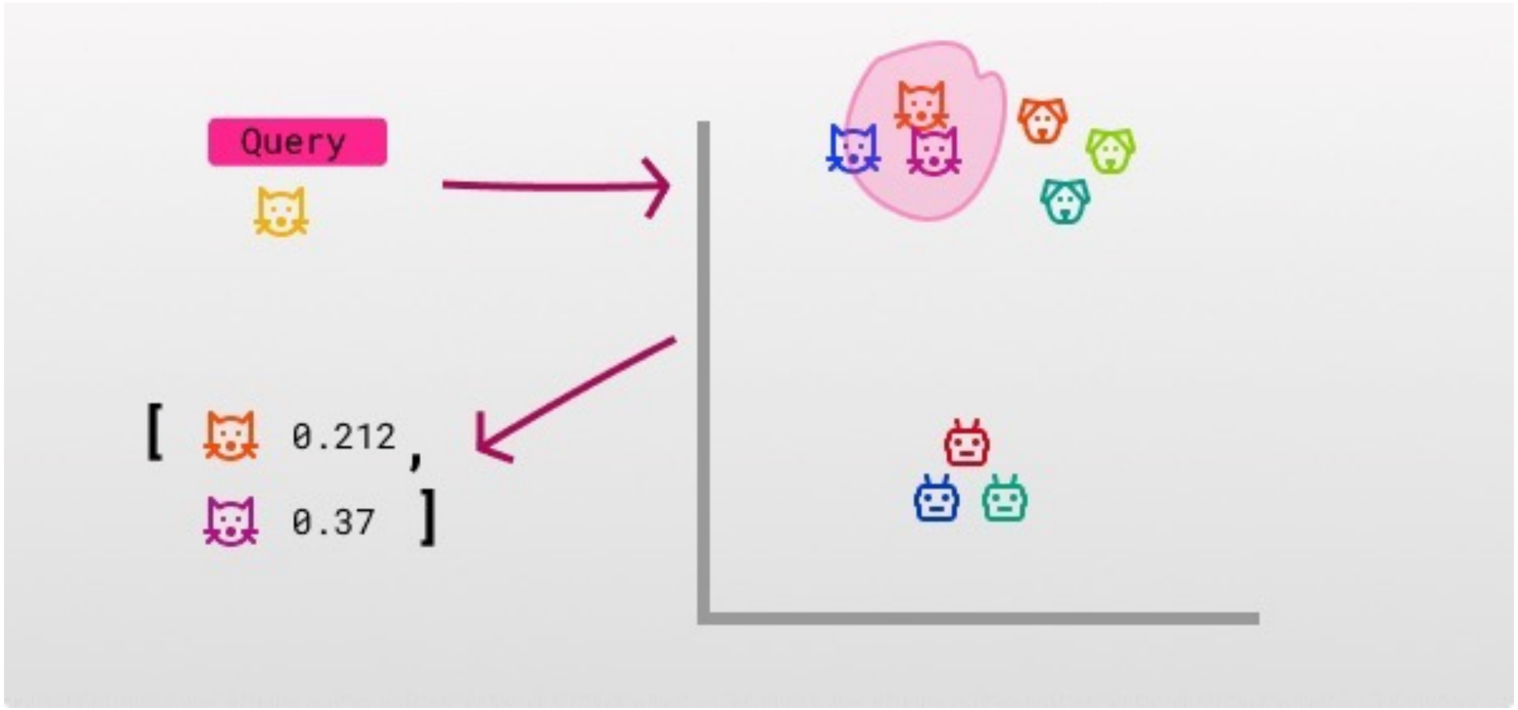


Make retrieval better - Vector DBs

- Vector databases store and retrieve high-dimensional vectors, these represent various types of data (e.g., text, images, audio).
- They are optimized for efficient similarity searches.
- First, we transform data into vector embeddings using models like BERT for text, ResNet for images etc

- Each type of data has different dimensions with more complex data like images taking more dimensions
- Highly scalable while maintaining speed of retrieval using NSW (Hierarchical Navigable Small World) or ANNOY (Approximate Nearest Neighbors Oh Yeah)
- Uses cosine similarity, dot products, or Euclidean distance for retrieving similar vectors
- This makes them perfect for recommender engines, and text search

Make retrieval better - Vector DBs



Make generation better with retrieval - RAG

- RAG (Retrieval Augmented Generation) is the perfect way to generate your data based on other data that is stored in a vector DB
- 2 main modules:
  - Retriever: Gets relevant documents from the Vector DB (using techniques discussed before) that are most similar to the input query
  - Generator: Generates relevant data based on the retrieved documents in context of the input query using GPT3 or any other generator model
- Main advantage is that the generated data is highly relevant to your use case
- Most suited for customer service bots, Q & A bots, and creating content that is specific

?

Thank you
Kamsahamnida
Terima kasih
Dziekuje
Mahalo
Toda
Spasibo
Dhanyavad
Merci
Shukran
Gracias
Tak
Asante
Dankе
khun
Khawp
Hvala
Efharisto
Xie xie
Obrigado
Shukriya
Arigato
Grazie
Salamat