GENERATIVE AI Nanodegree

> Large Language Models & Text generation

> NLP Fundamentals:

- Properties of a Tokenizer:

  model_max_length → Many pre-defined tokenizer can only take input upto a certain length of sequence.

  special_tokens → Unique tokens that are single entities that represent a class of tokens.

  ↳ Depends on the problem statement.

- Embeddings - An embedding is a way of projecting the high-dimensional text representation into lower dimensions.

  It ~~makes some~~ loses some information, but tries to retain the important information.

- Auto Regressive models:
  - Generate one token at a time, both to update context & generate new token.
  - Since newly generated token is highly dependant on the previous token, there is a high chance for the model to keep repeating the same token. That's why we need sampling rather than top-1 word.

- Temperature → the parameter that influences the sampling randomness.

-

> Transformers And Attention Mechanisms.

→ Vanishing gradient problem: The actual derivative → 0 ; making the weight update essentially useless.

In rnns, as the frequency of update is too high, it's far easier to reach the low gradient (plateau/peak) ; which essentially limits learning.

↳ Any update over RNN was tackling this problem (GRU, LSTM, Bi-LSTM etc.)

→ Information Bottleneck:

While the information is passed from one layer to another layer, ~~due to~~ there is a definite distortion of context. This is ultimately compounded by the time an input reaches the final encoding layer.

» Attention Mechanism

The query, key-value terminology is borrowed as-is from database retrieval.

»» Multiplicative Attention | $A(q,k,v) = \text{softmax}(qk^T) \cdot V$ | fast, representative.

»» Additive Attention | $A(q,k,v) = \text{softmax}(w_3 \tanh(qw_1) + (kw_2)^T)v$ | Flexible. Query, key dimensions..

»» General Attention | $A(q,k,v) = \text{softmax}(qw_1 g k^T)V$ | Flexible but more efficient. Additive Less efficient than Multiplicative)

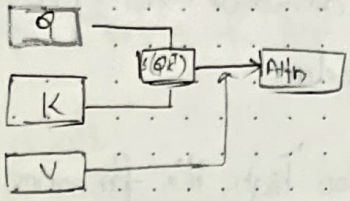
** Attention Scores → Mathematical definitions of Attention.

Attention Mechanisms → How these definitions are applied to different Q,K,V sets.

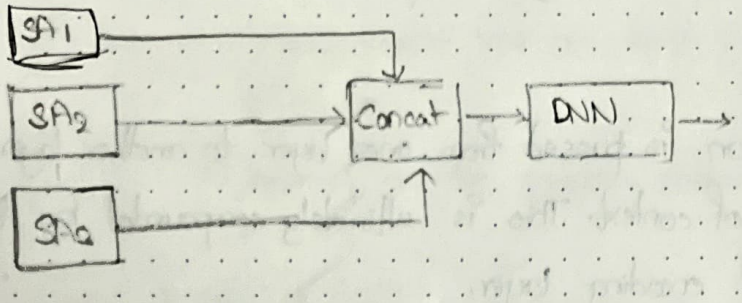
-Dot Product - Measures how similar two vectors are. [only focuses on $\vec{a}, \vec{b}$]

- We scale it by the order of their magnitudes to ensure general

>> Self-attention : Simple dot-product attention:



>> Multi-head attention: We run several instances of self-attention parallely and concatenated.



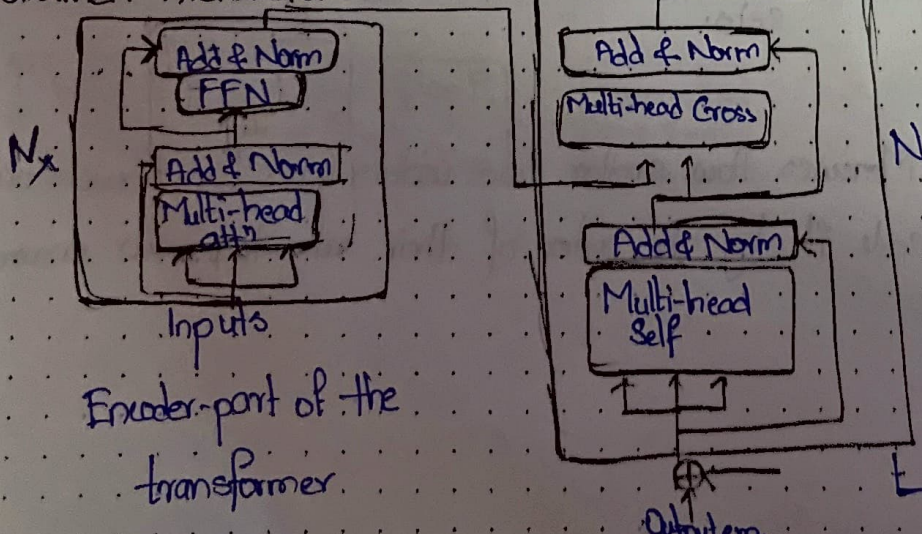
>>> Multi-Query Attention:

- Similar to multi-head, but instead of several {Q,K,V} sets, we have a single {K,V} set and multiple queries. The W_K, W_V are updated from the information of all these {Q} values.
- More efficient than Multi-head.

>>> Cross-attention:

↳ Query & {K-V} sets belong to different modalities

>> Transformer Architecture:



Transformers operate on ~~order~~ sets by nature, so we add positional encodings to add the positional information.

↳ Absolute Positional embeddings → Sinusoidal waves to parameterize the position

↳ Relative PEs → TransformerXL } → Active area of research.

↳ Rotary PEs → ?

>> Cons of Transformers:

→ High computational cost compared to LSTM & Bi-LSTM

→ Token length limit → 4000 ~ 200,000 token input length.

>> Current Research Trends

1. Optimisation:

↳ Hardware level Optimisation → Flash Attention
↳ 1-bit LLMs

↳ Architecture Variations (longer context sizes) → Reformer (locality-sensitive hashing)
→ Linformer (Low Rank Approx)
↳ Performer (Orthogonal Random Features)

↳ Alternatives to Transformers : Mamba.

> Retrieval Augmented Generation:

→ Prompt Engineering is a way to define the starting point of the models prediction

↳ Automating the prompt engineering, makes the model ~~deve~~produce more meaningful outputs compared to the manual refining.

↳ RAG is a step towards this "Automated Prompt Engineering".

>> Overview of RAG steps:

① Prepare the dataset.
↳ Vectorized text data of floating point numbers

② Find the data more similar to user input text using similarity metrics.

③ Design a custom text prompt that incorporates user's question & ~~ad above~~ relevant data

④ Query the model.

>> Common Data Representations: (for text)

→ Storage Representation - ASCII or Unicode

→ Binary Categorical representation - Onehot encodings

→ Embedding representation. *** (By far the best)

>>>> ASCII - each letter ⇒ a numeric code → Not a good representation as it leads to unnecessary math notations.

A-65 ; B-66 ; C-67

⇒ In data scraping, we can use 'SymSpell' library to correct spelling suggestions

Generative AI and Computer Vision <

> Introduction to Image Generation:

→ Discriminative models are approaches to learn Decision Boundaries.

↳ Generative models focus on the underlying distribution of each class in the dataset. (Can work for classification as a side-effect)

↳ Discriminative models learn $p(y|x)$; whereas generative models learn $p(x,y)$

→ Learning '$p(x,y)$' is much harder than $p(y|x)$.

→ Realistic images are an infinitesimally small part of the image space, which is mostly filled with noise.

→ Generative Image models → Conditional Generative Model [$p(\text{img}|\text{prompt})$]

↳ Unconditional [no starting point]

→ Video Generation is in infancy stages.

>Computer Vision Fundamentals

→ Image is a bunch of 2-D ~~image~~ array of pixels at it's core. At simplest, it's a single array (grayscale) at normal, its 3 sets of pixel ~~values~~ arrays (RGB)

→ Any analysis of images is interestingly layered & hierarchical:

Edges → Simple shapes using edges → Higher-level features from simple features.

→ Convolution is the process of extracting some information from the image with the help of a filter (Generally called Kernel).

→ Rather than manually defining these kernels, we let the backprop do it's magic.

» Common CV Tasks:

- Classification - What is ~~in~~ in the image?
- Localisation - Where in the image is the object present.
- Object Detection - Classification + Localisation
- Segmentation
  - Instance Segmentation - mapping each pixel to a class based on the contours and boundaries.

→ IoU is one of the best, simplest metrics to evaluate the predictions (bbox)

→ YOLO (You Only Look Once)

Image is divided into a grid → In each cell/grid-cell, YOLO tries to find any known-objects.

- Bounding boxes across detected objects
- Confidence-score for each grid.

→ Coverage - A measure of good diversity in the generations of the model.
- If a model has good coverage, it generates diverse images of a subject, i.e., a different breed of dog, a different angle, lighting etc.

→ Quality - Though this is intuitive, we need a quantifiable definition for quality. We do that by measuring the overlap between the probability distributions of the training/real-world data & the generated data.
- As images are of very high dimensions, we use a low-dimensional embeddings instead.
- Fréchet Inception Distance (FID)

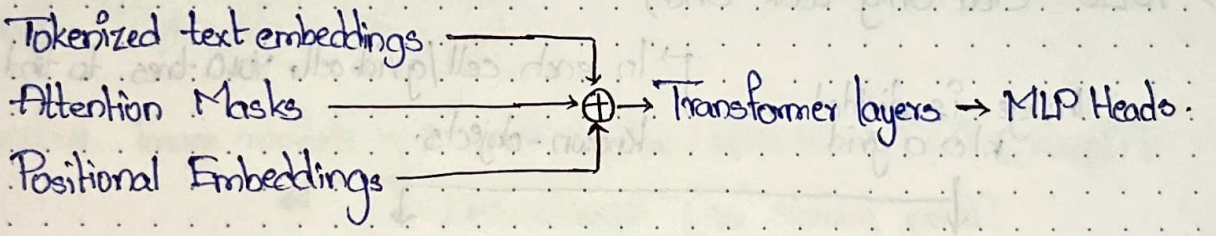
Real-world images → Pretrained Inception V3 Model → Distribution of Embeddings

Distance calculation (Wasserstein distance) ← FID

Generated Images → Pretrained Inception V3 model → Distribution of embeddings

> VISION TRANSFORMERS

This is a direct application of Text-based Transformer for images.
In text,



You can also consider an image to be a series of pixels.
(*) Raster Order - (Left to right) → Row after Row. ⇒ In a 64X64 img, top-left → 1st & bottom-right → 4096

The naive approach is to read all the pixels (in raster order) in place of tokenized text-embeddings.

- This ImageGPT was introduced by OpenAI in 2020

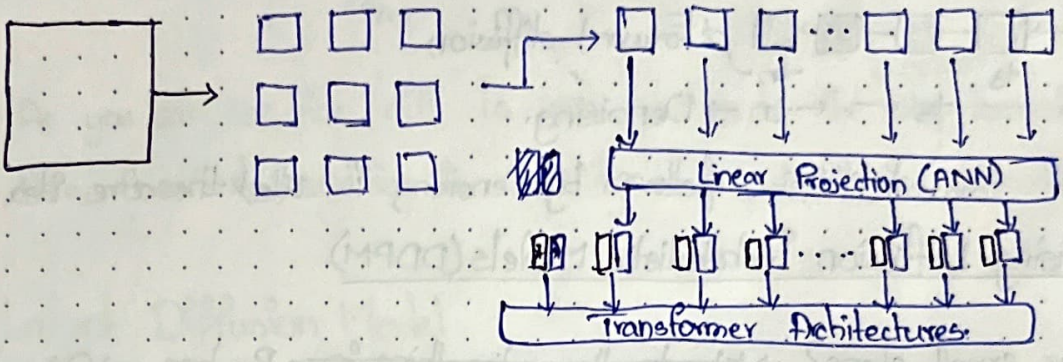
⇒ Vision Transformers (ViT)

Rather than the naive way of using all the pixel values, here, we use patches of the images as an input. These patches are linearly projected after they're flattened, and then are passed as embeddings.

→ Pros & Cons of ViT:

↳ Pros
↳ Compared to CNNs, they are not limited to local receptive field.
↳ Unlike CNNs, which need analysis for each task, ViTs are more flexible.

↳ Cons
↳ Like their NLP counterparts, ViTs are data hungry. Since images are way larger than text, the storage required explodes.



→ This "Linear projection" is nothing but a simple Conv2D or other NN that transforms 16x16 → 1x768 (?) vector.

⇒ A nxnx3 image → num_patches x3x16x16 → (num_patches +1) x768 (?)

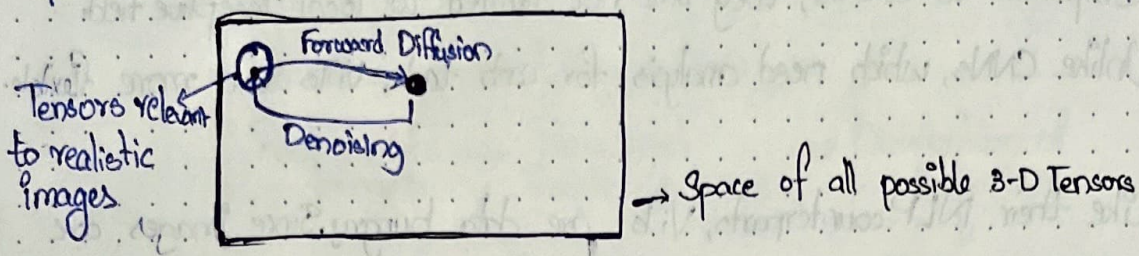
→ The extra patch is a learnable class encoding parameter. It is optised for a specific task, and is usually considered to "represent" the entire image.

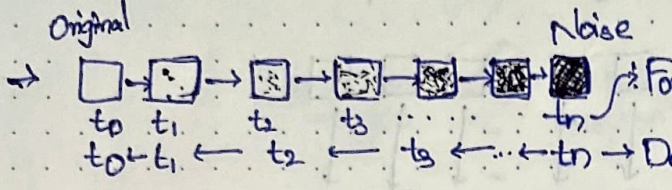
→ Models → Dall-E ; Dino V2 And SAM

> Diffusion Models

Why Diffusion? → Named after physical diffusion: high concentration → low
high pressure → low

The action of moving from a subspace of concentrated real world images to the larger, more vast space of set of pixels is, somehow, akin to diffusion process





This kind of forward diffusion followed by Denoising is ~~called~~ the core idea behind Denoising Diffusion Probabilistic Models (DDPM)

» Q. Why many small steps? → ~~We do the entire thing in~~ a Random → Original is hard.
high noise → low noise is far easier.

→ DDPM is "unconditional".

» Conditioning Diffusion Models

↳ We can condition the model on image, sketch, text, style etc. Most common? text.

↳ When Denoising, we just augment/add the noisy input with the text/image embeddings.

We use cross-attention to reinforce the conditions at different stages of the model architecture.

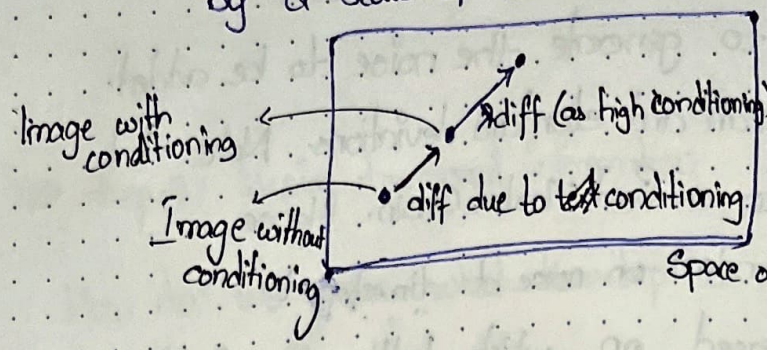
Of course, we sometimes remove ~~par~~ text/any conditioning during training for unconditional generation.

↳ During Inference, we use "Classifier-Free" Guidance Step.

Step-1: We first feed the random noise & text embeddings to the model. (Repeatedly)

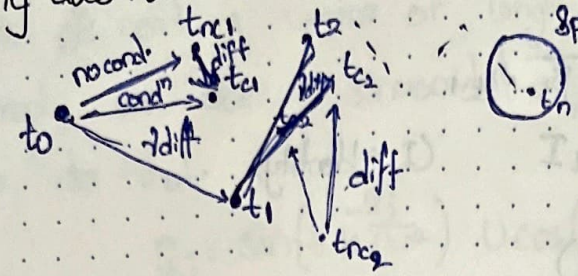
Step-2: We repeat the denoising without text embeddings. (same noise from s-1)

Step-3: We estimate the direction of difference from s-2 & s-1 and boost it by a scale of λ. [s-2 → s-1 is estimated to determine the effect of text]



λ → Classifier-free guidance scale.

Why does this work?



→ As you can see, the "diff" is used to estimate the direction towards the next time-step. Here, the text condition is implicitly moving the noise towards a final image.

» Latent Diffusion Model:

DDPs take too long and require huge computational hardware. So, what can we do? We embed into lower dimensions.

Image —Encoding→ embedding/latent vector —Diffusion→ Noisy latent $\vec{v}$

Noisy latent $\vec{v}$ —Denoising→ useful latent (or) new latent (or) generated embedding —Decoding→ New Image

⇒ This is just Autoencoders + DDPMs.

» DDPM Implementation (Theory):

»» Step-1 Diffusion

If you remember, the first step is to add noise at different timesteps. But this noise is very crucial and needs to be structured.

We use normal distributions to generate the noise to be added. For this, we need a specific mean and standard deviations. Note that the noise is added according to a fixed Markov Chain. Hence, noise at timestep (t+1) is dependent on noise at timestep t.

We, define a parameter β_t based on which both mean & std are determined. Specifically

$$\mu_t \text{ (mean at } t) = \sqrt{1-\beta_t} \cdot x_{t-1}$$

$$\sigma_t \text{ (std at } t) = \beta_t I \quad (I \Rightarrow \text{Identity})$$

$$\Rightarrow q(x_t|x_{t-1}) = N\left(\sqrt{1-\beta_t}\, x_{t-1}, \beta_t I\right)$$

But as intermediate (sequential) steps are unstable, we re-parametrise x_t's into terms of x_0.

We define $\bar{\alpha}_t$ to do this: $\bar{\alpha}_t = \prod_{s=1}^{t} (1-\beta_s)$

$$\Rightarrow q(x_t|x_0) = N\left(\sqrt{\bar{\alpha}_t}\, x_0, (1-\bar{\alpha}_t) I\right)$$

Here β_t is determined by a scheduler or can be learnt programatically. For now, we use a linear scheduler to increase β from $t=0$ to $t=n$.

»» Step-2 Denoising:

In the backward pass steps, we start with a noise at t_n as y_n and predict the noise to be subtracted from it from the model.

Say noise added at timestep t is ϵ_t and the noise predicted to be removed at backward timestep t is $\epsilon_{\theta t}$

The error then is a simple MSE of ϵ_t & $\epsilon_{\theta t}$

$$\text{error} = \|\epsilon_t - \epsilon_{\theta t}\|^2$$

Now, the loss function $\forall t$ becomes

$$L = \mathbb{E}_{t,x_0,\epsilon}\left[\|\epsilon - \epsilon_\theta\|^2\right]$$ (or) Minimising the expectation of the difference b/w $\epsilon, \epsilon_\theta$ at all 't'.

>>> Step-3: Passing temporal information.

How do we pass what timestep we are to the model?

We start with time 't'.

We need a vector of length 256 (arbitrary) unique to timestep 't'.

Simply, we use a sinusoidal embeddings (similar to positional embedding) to do that.

$$P_{tj} = \sin\left(t \cdot k^{\frac{-2j}{n_c - 2}}\right) \cup \cos\left(t \cdot k^{\frac{2j}{n_c - 2}}\right) \quad \forall\ j = 1 \cdots 128 \quad \text{(for 256 values)}$$

$$\vec{P_t} = [P_{t1}, P_{t2}, \cdots P_{tn_c}] \quad (n_c \text{ here is } 256)$$

Then we pass this $\vec{P_t}$ through a neural network to get the embeddings.

>>> Step-4: Paralleising the whole process to a batch.

Instead of training for all images at all timesteps, we consider a batch of images of size bs.

for each image(x_i) in bs, we select a random timestamp 't', between (0, T) where T ⇒ Total timesteps.

```
b = batch.shape[0]
t = torch.randint(start=0, end=T, shape=(b,)).long()
```

Based on this, and $\bar{\alpha}$ we create a noise tensor & add it to images.

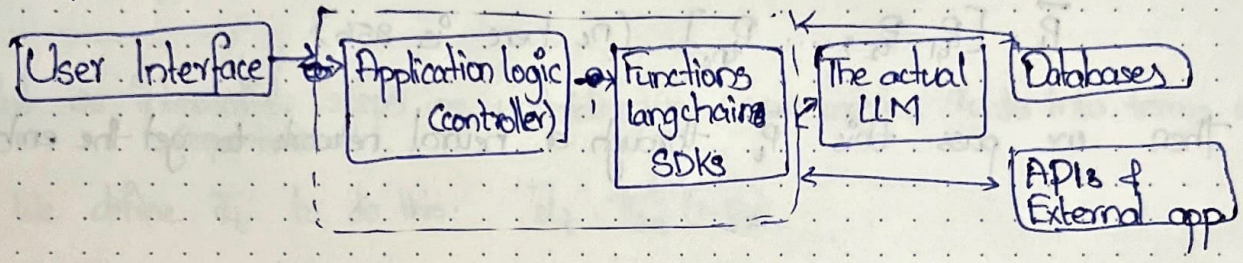
Generative AI Solutions:

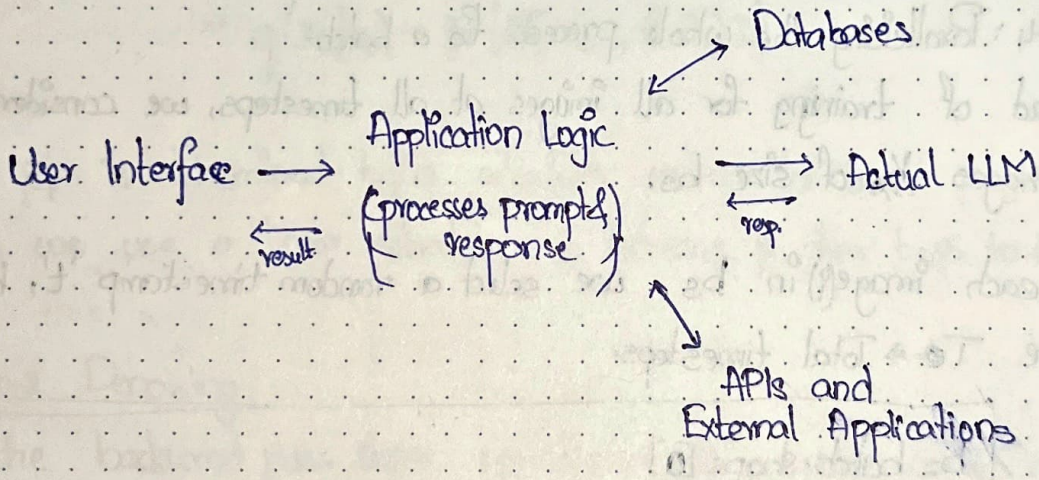
> Building Gen AI solutions:

→ Generative AI Techstack → Hardware layer → The GPU/TPU/ other compute providers.
→ Model/Platform layer → Models & Data understanding
→ Application layer → Enables user interaction with the models.

→ Generative AI's major problems - Unpredictable responses
- Statelessness
↳ The model's chat history you see in it's responses are fake.
A model can only respond to what's actually in the API call.

→ Components of Generative AI Solutions





→ AI agent - a program that performs automated tasks, often mimicking human behavior (or) decision making processes.

» Prompting [Basics]

↳ System prompt: Defines and sets the model's overall context & purpose.

↳ Contextual prompt: Provides immediate task-specific information to guide the response.

↳ Role prompt: Frames the model's output style and voice.

↳ Chain Of Thought: Asking the model to respond 'step-by-step'.

↳ chain of Thought - Self consistency - Repeat the same cot prompt multiple times, get the most common output.

↳ ReAct Prompting: Useful when using LLMs + external tools.

↳ Works based on a thought-action loop until the task is done.

↳ Typically, an ReAct prompt contains 3-sections in it's system prompt.

↳ (Thought step) Assess the provided information and think of next steps to take

↳ (Action step) - Contains a list of actions which are sometimes linked to external tools that perform actions.

↳ [Observe step] - Summarise and gauge the responses from the Action steps and decide whether to continue (or) finish the task.

↳ Third party tools like langchain offer predefined methodologies for these tasks (ReAct) - All you'd have to do is plug-in the tools from APIs and you're good to go. (in detail later)

→ Whatever "function Calls" (or) "External tools" we are calling happens before a prompt enters the model (or) after the prompt enters the model.

→ So, whatever "Generative AI" solution you're building, it's going to need a specific structure.

① Define the tasks that the solution can handle (both tools and other things)

② Craft a few user stories to → Identify user intent

↳ Map the user intent to the tasks and tools identified.

③ Define basic prompt templates.

→ Function calling

① Define a function

② Define a 'tool' list, whose elements are dictionaries describing different tools.

```
tool = [ {"type": "function",
          "function": {
              "name": "functionOriginalName",
              "description": "Function Description",
              "parameters": {
                  "type": "object", (? isn't everything an object?)
                  "properties": {
                      "input_string": {
                          "type": "string",
                          "description": "param_desc"
                      }
                  },
                  "required": ["input_string"]
              }
          }
      }
]
```

⑧ LLM uses this information and prompts given to get user intent & map it to the function calls. The response is parsed programatically and the function calls are made:

Prompt + tool description → LLM → Intent based, actionable steps → Parsing & Executing these steps → Result (or) Another LLM call

In OpenAI responses, we get a "toolcall" field to determine what external tool to be called.

> Vector Databases

→ A storage tool that's stored seperately to augment the model's knowledge gaps.

→ prompt → Vector Database querying → Prompt augmentation → LLM call.

→ Vector search is the process of querying a vector database and retrieving the closer/similar ones.

→ There are many Vector Search tools out in the wild.

↳ Vector Index libraries - These focuse on indexing vectors to support fast vector search.
- These don't have many data management features but have good vector search features.

↳ Traditional databases - ElasticSearch and similar tools.
- These aren't scaleable and are unoptimized.

↳ Vector database providers - Support unstructured data
↳ LanceDB, Milvus, Pine cone etc.
- More complete offerings.

→ Basic necessities of a vector database - Querying & Indexing

Additional management features - Filtering - caching & storage.
- Post-processing
- Versioning
- Updating

We also need the ability to store raw data for AI applications
Meta-data writing
Keyword search
SQL support