This is a script used for the YouTube video, which contains skipped parts to make video shorter, read this script if interested in depth explanations.

Hey everyone! You can call me Ryan; I am an AI voice brought to life using Clipchamp’s text-to-speech feature. and today, we’re diving into something that’s equal parts wild and groundbreaking. I was doing my usual scroll through the latest buzz in Artificial Intelligence, and stumbled on a problem that caught my eye –

Drug Discovery. Sounds intense, right?

But then, I thought… ‘Why not take the craziest approach and actually *build* a solution using AI?’ So, welcome to *PharmaGPT*!

We’re not just building any AI project here; we’re going to tackle one of the complex and impactful challenges – from scratch!

Let’s be real for a sec: I have no idea if this will work (the odds are not in my favor), but that’s part of the fun, isn’t it? We’re about to get into the nitty-gritty – the tech, the coding, the concepts, the legal stuff – all while keeping it *real* and *fun*.

Here’s the kicker: I’m just an average tech enthusiast with a computer using a browser and a free Google account. Sarcasm incoming.I already feel like we are about to challenge the status quo… or, you know, *barely scratch the surface* and hope nobody notices. No fancy tools, no pharma knowledge. Just me, some generative AI, and ChatGPT as my co-pilot.

So buckle up, because whether it’s success or failure, we’re here to learn, build, and hopefully break some rules along the way. Let’s do this!"

Alright, so here’s what we’re really diving into — the actual *grunt work* behind building AI projects. Yep, the stuff no one really talks about but is absolutely essential!

This video will break it all down into easy-to-follow sections:

1. **Dataset Building** – because AI is only as smart as the data we feed it.
2. **Generative AI Concepts** – we’ll get into the fundamentals while building our model.
3. **Inferencing** – turning raw models into actual predictions (the fun part).
4. **Hyperparameter Tuning** – where we fine-tune our AI like adjusting the dials on a radio (remember those?).
5. **Legal Stuff** – yep, even AI has to follow the rules, and we'll cover what that means.
6. **Future Applications** – what could this project turn into, and how could AI shape drug discovery?

* **Dataset Building**
* **Generative AI Concepts**
* **Inferencing**
* **Hyperparameter Tuning**
* **Legal Stuff**
* **Future Applications**

So stick around, because we’re going to break it all down, step by step!"

Alright, let’s talk about the foundation of any AI project — *data*. It’s the crux of everything we’re about to do. If data isn’t your thing, feel free to skip to [timestamp] where we dive straight into building the model, training, and seeing if we can actually make this work.

But for those of you still with me — here’s the plan. We need both an input and an output for our AI. The output is easy: we’re generating medicine. The input? Diseases. Simple, right? Well, not exactly.

So, how do we represent diseases and medicine in a way our AI can understand? That’s where the fun starts.

For diseases, we’re going to use *gene protein sequences* — it’s like giving our AI the DNA of diseases. And for medicines, we’ll represent them as *chemical compounds*. These compounds can be described in two main ways: **SMILES** (great for simple molecules) and **InChI** (more accurate for complex ones). It’s like shorthand for how chemicals are structured.

Now, you’d think finding this data would be easy, right? Nope. I scoured the internet and hit a paywall — literally. The good news? There are student-accessible databases, but they’re usually locked behind academic verification. So, what’s our Zero budget solution? *We build our own database.*

Here’s how I did it:

**Step One**: Get the list of diseases. I pulled data from UniProt, which gave me disease names and their protein sequences. This dataset is huge (about 280 MB), according to zero budget and includes everything from human diseases to plant and microbial diseases, even the *Penicillium notatum*, which gave us penicillin.

**Step Two**: Now we need to find the corresponding chemical compounds for those diseases. In an ideal world, we’d hire a team of doctors, researchers, and chemists, but... we have zero budget. So, I turned to ChatGPT, but even then, limitations with free and premium models meant we didn’t always get accurate SMILES or InChI codes.

**Step Three**: We would be querying every disease's protein sequence name and asking their respective medicines to any GPT models that are trained on medicinal research papers. Top contenders are ChatGPT, copilot, Hugging face, Gemini, claude AI and the lit goes on. Pick your poison.

**Step Four**: Now, we need the actual SMILES and InChI codes for each compound. So, I wrote a web scraper to pull this data from Wikipedia. Even then, I hit some discrepancies and had to double-check certain compounds using PubMed.

**In short**: Data collection took 5 days — 3 for querying GPT models, 1 for web scraping, and 1 for piecing everything together. And now we’ve got ourselves a basic dataset to train PharmaGPT.

And that’s how we built a foundation for this project. Next up, we dive into building our model!

Now comes the fun part — *designing the AI model* itself. This is where we get to dive deep into the tech that powers AI like ChatGPT.

So, what model are we using? We’re building our very own transformer from scratch, the same core architecture used in chat-GPT models for predicting the next word or token in a sentence.

Let’s break it down. Transformers were originally designed for language translation — think English to Hindi — using an encoder-decoder setup to capture two different languages and relate them through a cross-attention mechanism.

But here’s where things get interesting for models like ChatGPT. Instead of translating between languages, they predict the next word in the same sequence, making them **decoder-only models**. These models use *self-attention*, which means they focus on different parts of a sentence to understand the context. For us, we could treat drug discovery like a translation problem, or we could go for a next-word prediction approach.

And I’m choosing self-attention only approach. Why? Because it simplifies things and gives us a peek into how GPT models predict text.

**The twist?** We’re not using a pre-trained GPT model here. Nope, we’re building everything from scratch. This is what’s known as a **domain-specific model**, tailored for a specific field like pharma, law, or even coding.

**So what’s the catch?** Let’s talk advantages and disadvantages, *specific to us*:

**Advantages**: If you know exactly what you’re looking for, custom models take *less time*, *fewer resources*, and *lower costs* to achieve specific goals. Generic models like ChatGPT are versatile, but when you need something hyper-specific, you have to add more layers, increase the context window, and pour in resources to boost accuracy and performance.

**Disadvantages**: Expertise. We need someone who *really* knows the field. In this case, that means having a doctor — or better yet, a research scientist or pharmacologist — to understand what’s happening in the domain. But, since that’s not happening for us, we’ll discuss this challenge toward the end of the video.

**So what’s our final model?** We’re going with a decoder-only, ChatGPT-style model — a generative AI that uses self-attention to grasp the context in a single sequence and generates text one token at a time. The model will rely on techniques like input embedding, positional encoding, multi-head self-attention, feed-forward neural networks, and a softmax layer for output.

Now, here’s the kicker… I’m *not* going to code this transformer from scratch. Why? Because it's way easier to *source* it from ChatGPT itself! But here’s where things get tricky: you’ve got to be good at **prompt engineering** to make it work.

I started with a simple prompt:  
*'I want the code for a ChatGPT-style transformer, including every concept used in GPT-2 or GPT-3, in Python using TensorFlow.'*Sounds easy, right? Not so much. It took me multiple prompts, debugging errors, and learning each concept — from embeddings, to multi-head self-attention, all the way to the final logits layer. By the time you’re compiling the model with a softmax loss function, you’ve gone through *a lot* of back and forth with the AI.

**So, how long did it take?** About 3–4 days of tweaking, re-prompting, and slowly piecing it all together. Easier said than done, right?

But that’s the beauty of it — with patience and the right prompts, you can get ChatGPT to do most of the heavy lifting. And trust me, it’s a wild ride."

Okay, let me be real with you — I’m not here to deep dive into every theoretical concept of AI. I’m more of a *hands-on* guy. But if you're craving a solid breakdown of transformers and attention mechanisms, I highly recommend checking out **3Blue1Brown** on YouTube. Their visuals are mind-blowing, and trust me, I don’t have the skills to recreate anything that great.

But, since you're here, I’ll give you the gist: Transformers use something called **self-attention** to understand context. In simple terms, when ChatGPT reads a sentence, it looks at every word and tries to figure out how they relate to each other to predict what comes next. That’s it.

Let’s break down the magic behind ChatGPT: the **decoder-only transformer** model.

Picture this: ChatGPT reads a sentence one word at a time and tries to predict the *next* word. To do that, it uses a technique called **self-attention**, which means it’s constantly looking back at the previous words in the sentence to understand the context. This helps it figure out what should come next.

Now, unlike the original transformer models — which use both an **encoder** and a **decoder** (like when you translate languages) — ChatGPT skips the encoder part. Why? Because it’s not translating between two different things; it’s just focusing on generating the next word in the sequence. That’s why it’s called **decoder-only**.

The process involves several key steps:

1. **Input Embedding**: First, it turns words into numbers that the model can understand.
2. **Positional Encoding**: It keeps track of word order, so it knows which word came first.
3. **Self-Attention**: This is the secret sauce. The model checks all the previous words to figure out which ones are important for predicting the next word.
4. **Feed-Forward Layers**: After self-attention, it processes everything through some regular neural networks to make the final decision.
5. **Output**: Finally, it spits out the most likely next word, thanks to a softmax function that picks the best option from all possibilities.

And just like that, ChatGPT generates text that feels like a real conversation, one word at a time!

No crazy deep dive, just a high-level overview. For the cool visuals and detailed explanations, go watch those 3Blue1Brown videos!

Alright, we’ve got our model, and now it’s time for the fun part: **data handling and vocabulary building**!

First up, we need to create the **input and output** for our model — this is the crux of data handling. To keep it straightforward, here’s the plan:

1. **Sequence Design**: Each sequence will look like this: *‘Disease protein sequence’ + a separator token + ‘medicine SMILES or InChI’ ending with END token*. The Separator token separates the input from the output, and the END token signifies the end of the sequence.
2. **Next is Tokenization**: Next, we’ll convert each character in the protein sequence and SMILES/InChI into unique tokens. This helps the model understand what it's working with.
3. **Our output would be Right-Shifting the actual Output**: To prepare our output for training, we shift it to the right. This way, the model learns to predict the next token based on the previous ones.
4. Bundling the data in a numpy array with the respective shape.
5. **Numpy Arrays**: Finally, we’ll bundle everything into a Numpy array. We’ll end up with two arrays:
   * One for input data shaped as (number of sequences, max-padded length of tokens).
   * Another for output data shaped as (number of sequences, max-padded length of output tokens, unique tokens in the output).

The max padding is crucial since sequences can vary in length, and for TensorFlow to train effectively, all input and output shapes must match.

Now that we've got our input and output data ready, it’s time to train the model!

For this step, we need to declare and compile the model. To keep it simple for our demonstration, we’ll use a minimal number of layers and set our batch size to 1. Why? Because our Colab notebook has limited resources, and hence we will be dealing with one sequence at a time.

**Pro tip**: The industry standard for loss functions in next word prediction models is **Sparse Categorical Cross-Entropy**. This helps us measure how well our model is predicting the next word. So there would be a change in Shape of our output.

Let’s get this model trained!"

Alright, let’s dive into training our model! For illustration purposes, we’re going to feed it just one sequence of disease-drug pairs and see how it performs.

After running the training for 10 epochs, we can see some impressive metrics: the model achieves an accuracy of about **0.9** and a loss of around **0.1**. That’s pretty remarkable for such a short training time!

Now, if we ask ChatGPT what these numbers mean, it gives us some insightful feedback. A **0.9 accuracy** with a **0.1 loss** suggests that our model is performing well and is a good fit for the task and the dataset we’re using.

However, there’s a catch: it raises potential concerns about **overfitting**. This means our model might be doing a fantastic job on the training data but could struggle when faced with new, unseen data.

So, while the initial results look promising, it’s essential to keep an eye on how it performs in real-world scenarios. Let’s see how we can tweak it for better generalization!

Now, while we’ve achieved a **0.9 accuracy** and a **0.1 loss**, it’s important to remember that high accuracy with low loss doesn’t necessarily mean our generative AI model is performing well. We need to look at other metrics, like **BLEU score**, to truly assess its effectiveness.

The real test, however, comes when we start **inferencing**. This is where we can see if our **PharmaGPT** is truly capable of its task. Since it’s a task-specific AI model, the best way to verify its performance is to infer using the same sequence we trained it on.

So, let’s put it to the test! If our model generates the exact same SMILES representation we provided during training, we can confidently say it works. If not, we’ll have to explore other options and improvements.

Let’s see what happens during inferencing!"

Well, it turns out our model is spitting out gibberish instead of valid SMILES! Despite achieving impressive evaluation metrics, it’s clear that our PharmaGPT isn’t performing well in practice. We even switched from **Sparse Categorical Cross-Entropy** to **Categorical Cross-Entropy**, but the results remained unchanged.

We’ve followed every step in the book—technically and theoretically, our model makes sense. Yet, during inferencing, the reality is quite the opposite.

So, where did we go wrong? Most issues in AI projects stem from the data itself, and a closer look at our output reveals two major red flag: it seems the tokens from the gene protein sequences are creeping into the output.

The culprit? **Extreme data imbalance.** The sequence structure we’re using, which combines diseases and medicines, skews heavily toward gene sequences—**95% gene sequence to just 5% SMILES**.

To dig deeper, I asked ChatGPT and Gemini whether this situation could occur, even with our training metrics showing **0.98 accuracy** and a **loss below 0.1**. Could input sequences influence the output in next-word self-attention models due to this imbalance? The answer was enlightening.

Here’s the summary: high accuracy does not guarantee generalization. There’s a significant chance this problem won’t vanish even if we increase our data points or implement a teacher forcing approach, or even if we add more layers to the model.

This was a surprise for me! I didn’t expect self-attention mechanisms to have such limitations. They’re powerful, but they can struggle with long-range dependencies in input sequences—especially when data is imbalanced. This means our gene sequences are receiving more importance and weight in the model we’ve built.

In short, we need to rethink our approach. Let’s explore how we can balance our data and improve the model's performance!

Now, while data rebalancing is one of the common solutions to tackle our issue, it’s unfortunately not feasible in our case. So, what’s next? How can we still achieve our goal?

All we need for our problem statement is to maintain separate vocabulary for input and target sequence. Let’s break it down. First, we need to stop letting the input sequence hog the spotlight during the inferencing of the output sequence. This can be achieved by converting our next word prediction approach to the language translation problem, the one with a cross-attention **mechanism** . However, this means we’ll have to completely revamp our model structure from scratch again. Adding an encoder to our current model would be one of the major changes we need to implement.

This shift will allow us to maintain a separate vocabulary for the input and output, ensuring that the input tokens won’t interfere with our output tokens.

But hold on! This means we’re starting from scratch—ground zero in our model-building process.

To implement a cross-attention mechanism, we need to introduce an **encoder** to capture the context of our input within this new language translation framework. We’ll also need to adjust how we handle data inputs and outputs. Instead of just two variables, we’ll manage three:**Source Sequence, Target Sequence, Right Shifted Target Sequence**

1. **Source Sequence:** Our protein gene sequence.
2. **Target Sequence:** The SMILES representation.
3. **Shifted Target Sequence:** This will help with the output during training.

Alright, let’s dive into a key concept in AI: the difference between *next word prediction* models (like ChatGPT) and *language translation* models. And to make this fun, let's break it down with a simple question:

If you're working on a language translation problem with *just one layer*, how many attention mechanisms would you need for the model to actually work? Sounds like a riddle, right?

Well, the short answer is—you’ll need *three* attention mechanisms. And, yes, we’re talking about two types of attention: *self-attention* and *cross-attention*. Each plays a different role depending on where it’s used in the model.

Let me explain:

1. **Self-Attention in the Encoder BLOCK:** This is where the model tries to understand the input language. The encoder takes the sentence from the source language (say English), and the self-attention mechanism helps it figure out which parts of the input matter the most.
2. **Self-Attention in the Decoder BLOCK:** Now, on the target side (let’s say Hindi), the decoder also needs to understand what’s going on within the target language itself, and that’s where another self-attention mechanism kicks in.
3. **Cross-Attention in the Decoder BLOCK:** Finally, the magic happens here—this attention mechanism links the two languages. It makes sense of the relationship between the input (English) and the target (Hindi) at the start of the decoder. This is where the actual 'translation' happens, connecting the two worlds.

So, to recap, you need:

* *Self-attention* in the encoder to focus on the input language.
* *Self-attention* in the decoder to focus on the target language.
* And *cross-attention* in the decoder to connect the two languages.

This is how we go from "next word prediction" to "language translation," with each attention mechanism doing its part. Simple, right?"

And each encoder and decoder block is repeated n times where n is the number of layers.

To get this rolling, we’ll prompt ChatGPT with something like this:

*“Give me code for a language translation model with encoder-decoder architecture consisting of a self and cross attention mechanism and every other component that is required for generative language translation transformers, just like in the research paper ‘Attention is All You Need’ paper.”*

*You might not get exactly what you need right away when building the language translation model. In fact, you might need a solid 2-3 more days of prompting ChatGPT. That's because getting everything right with the transformer model takes time—especially when you're trying to clear up every concept used in the code. It’s a process of trial, error, and understanding.*

Now there is one of the most important concepts in generative AI. That is Masking.

Masking is a key concept in AI models, like transformers, used for controlling,considering and ignoring which parts of input data the model can "see" during training or inference. It’s like setting up blinders, so the model can focus on the right things at the right time. The definition I narrated was for custom models. The true definition used for the generic transformers is displayed on the screen pause if interested.

Masking is a part of parameters of multihead attention where the other parameters are query, key and value. Concept arrives from a dictionary where one has a query and the answer to that query is found using a key aka an index that’s pointing to a value. Masking prevents access to some entries, specifically the future tokens in the query.

**Types of masking**

There are many types of masking, some of them are displayed on the screen. We will be using the basic version of masking which are a padding mask and look ahead mask also know as casual mask.

First, *Padding Masking*. This is used when input sequences have different lengths. To process them efficiently, we pad them to the same length. But we don’t want the model to learn from these artificial padding tokens, so padding masking is used to ignore them.

Then there's the *Look-Ahead Mask* also known as casual masking – this is super important in language translation tasks. When generating text, the model shouldn’t "cheat" by looking ahead at future words of the targeted language or data it hasn’t predicted yet. The look-ahead mask ensures the model only considers the words it has already processed.

In AI models like transformers, **embedding** and **positional encoding** work together to turn words into meaningful inputs. Embeddings represent each word with a unique vector in a "map" of relationships, where similar words (like "king" and "queen") are close to each other, helping the model understand meaning. Since transformers process all words in parallel, they lack a natural sense of order; positional encoding adds this by assigning each word a position-based value, helping the model know the sequence of words. Together, embedding captures the meaning of words, and positional encoding provides their order, enabling the model to process the entire sentence efficiently and contextually.

**So far, we've covered the basics of language translation transformations, and we've even sourced our code using ChatGPT. Now, we’re getting to the *real* exciting part—training the model. But before you jump in, it’s crucial to know exactly how much resource allocation your model needs. That means getting a clear understanding of factors like the number of layers in the transformer, the size of your vocabulary, and much more."**

**"This is where *hyperparameter tuning* comes into play. Think of it as adjusting the dials on a machine to get it running at maximum efficiency. Hyperparameter tuning involves tweaking critical settings like learning rate, batch size, and the number of layers to find the optimal combination that boosts your model’s performance."**

**"Here’s an analogy: it’s like cooking the perfect dish. Too much salt? It’s ruined. Too little heat? It’s undercooked. In the same way, finding the right balance between these hyperparameters ensures your model neither overfits nor underfits."**

**"Let me break down two common strategies used for hyperparameter tuning: *grid search* and *random search*. Grid search explores all possible combinations, but it can be very resource-intensive—it’s exhaustive but effective. *Warning number one*: grid search can quickly eat up your computational resources, especially for large models. On the other hand, random search picks different combinations randomly, which might sound less precise, but in reality, it’s often much more efficient and gets you to a solution faster."**

**"Now, *warning number two*: hyperparameter tuning is a trial-and-error process. Even with techniques like random search, it can take time, and there’s no guarantee that every combination you try will improve the model. But here’s the upside: the right hyperparameters can turn a mediocre AI model into a top performer. It can be the difference between a model that barely functions and one that excels at its task."**

**"But here’s where we hit a snag. We’re working with limited resources and even less time. *Warning number three*: when you’re short on both budget and time, hyperparameter tuning becomes a lot trickier. You need to prioritize the most impactful parameters first—like learning rate or batch size—because those will give you the biggest gains for the least cost."**

**"In short, hyperparameter tuning is all about striking the right balance—fine-tuning those settings until your model is learning as efficiently as possible. But remember, it’s not a one-size-fits-all process. Be ready to experiment, fail fast, and learn quickly. And if you're short on resources, make sure to focus on the hyperparameters that matter most to avoid getting stuck in endless tuning cycles."**

**So we would be using custom techniques suitable for our problem statement and our model.**

**It involves simple steps: Overfit the model with one sequence, and that infer the model on the same data, technically it should give out close enough output. If it does, increase resources. If it does not, we’ll see.**

**The structure of the Inputs/output sequence for the language translation approach is simple. They are displayed on the screen.**

**"We’re exploring the *customized bottom-up approach* for training models. We start with minimal parameters and intentionally overfit the data. Overfitting means the model memorizes the data, so when it makes predictions on that same data, it should hit the target perfectly. But, *warning*: overfitting isn't good for generalization—it’s just a step to understand the model's behavior."**

**"After observing performance with low parameters, we gradually increase them, noting any improvements. But, *warning number two*: more parameters don’t always mean better results. You can hit diminishing returns, where adding more resources doesn’t improve performance or can even worsen overfitting."**

**"This approach helps find the *minimum resources* needed for optimal results, which is ideal if you're low on budget or time. If you’ve got the resources, you can scale faster, but this method ensures efficient problem-solving."**

Let's see this model in action.

After training and inferring for the first time , we are at least getting the first 2 tokens right. If we are conceptually correct, by increasing the number of layers and dimensions of embedding, the correct number of tokens should increase on overfitting the data or doing the industry standards.

By looking at the output of the model, we were able to solve the problem of input sequence that enters the output sequence. But even after increasing the number of layer. There was no improvement while inferencing. We have now encountered another huge issue during inference. We are having the following issues.

* Alignment Challenges
* Information compression
* Training complexity
* Potential inability model Generalization

These issues are caused by the huge difference in sequence length between 2 languages that we are trying to translate from one to another for any language translation problem. For example English (longer word count) to Chinese(short word count) which deliver the same meaning.

Now potential solution for this issue are

* Length penalty
* Enhanced attention mechanism
* Preprocessing

But this video is meant to cover the basis of generative AI, this first two is complex to achieve. The preprocessing part can be include in basics of generative AI.

The preprocessing part include the following

* Sentence segmentation
* Tokenization
* Paraphrasing

In short, a form of data augmentation. Which we could do. Sequence structuring. This requires innovative thinking when it comes to paraphrasing. But we are going to keep things as simple as possible.

Our thinking is if the language translation model performs very well if the sequences are of same length. Then why phrase our target sequence of the same length. One quick fix can be repeating tokens in a structured faction. For example if our input sequence length is 363 and out output sequence length is 33. We will divide 363 / 33 = 11.so we will repeat each token in the target sequence 11 times consecutively. For example the smile mentioned above

Clc1cc2nccc(c2cc1)NC(C)CCCN(CC)CC

Would look like displayed in the video.

CCCCCCCCCCClllllllllllccccccccccc11111111111ccccccccccc…….

In case if the remainder is there, suppose 365 / 33 = int(11) , we would get the remainder as 365 % 11 in python as 2. In the end we will fill those places with ‘[END]’ tokens. Even in cases where there is no remainder , we would still add an end token.

And then we repeat the process of hyperparameter tuning and see if we were able to solve the issue. Inference code will be different, it will consist of removing the repeated token to get the fine output.

We will start with the same base case. Remember our output sequence is “Clc1cc2nccc(c2cc1)NC(C)CCCN(CC)CC” and with one layer and 32 dimensions we got “Clc)C(C” we got the first three tokens correct. Lets see what happens if we increase our resources slowly. We got “Clc1c2nc1)NC”. Yup, our model is now working by the book.we got 5 tokens correct with just 2 layers and 48 dimensions. Our model follows what a language translation model follows. As we increase resources, our performance has increased.

Now, let’s train our model again with the output structure change and notice if there is an increase in performance. And we were able to infer the first 2 tokens correctly. Let us see if increasing resources, will there be increase in performance and if yes, we are on right track. And surprisingly, we went from 2 tokens to 3 tokens by increasing 1 layer. Increasing more resources slowly, we got four tokens correct.

Since this video aims to provide only the basic concept of Generative AI and proof of concept for PharmaGPT. We will not be training the model that is production ready. Simple reason, time and money. The model that we ran was on a simple cpu provided by Colab, probably intel i5-7-9, pick your poison. It would take years or decades if I proceed with CPU. This also taught me the importance of high end gaming laptops and GPUs. I asked ChatGPT the initial valuation of a startup that works on a discovery model and was astonished that the amount it gave me was 5-15 million dollars. Just Wow!!.

Moving on with our discussion. Remember while discussing advantages and disadvantages of custom domain specific models. The model would better understand and converge fast, taking less training time if we add details about the targeted variable. And that’s why A doctor is needed for our problem statement. A doctor can add more variables on the input side explaining how one can arrive at the output side. This would require extreme domain specific knowledge which would reduce training time, generalization, and less resources while training.

For example doctors can help start including information about protein structure’s function, target binding sites, or target mutation etc. Physicochemical properties, affinity, selectivity, pharma-cokinetics and pharma-codynamics. One can include information about description of diseases, like pathogenesis, prevalence, severity, and its progression. The list goes on. Basically Domain-Specific Knowledge and variables.

We do not have pharma specific domain knowledge. So our hands are tied in this case. But we do have expertise in Business analysis and administration. This gives us the ability to analyze financial markets and create advanced custom AI models that predict Next day Daily sentiment forecasting for 20+ stock exchanges for 200+ indices with an average accuracy ranging from 75-80% forward tested on live market data for more than 268+ days. But we soon realize, people are less interested in sentiments and more in what would be the exact entry and exit trades for the day readymade. So from a technical standpoint, this model seems amazing but from a business point of view, there is no demand for it. We could have gone for exact trade providers but for that we would require a license for each of these stock exchanges separately. Then you might have this question, why go for sentiment analysis for every other stock market. Well simple argument, stock markets influence each other. The saying “ the closer you look, the less you see” is true to some degree, mostly one of the factors being Foreign direct investment. Moral of the story, make sure your AI problem statement makes Business sense and has a demand for it and is feasible to achieve.

So we have almost completed our checklist for the video. The remaining parts are legal exposure and future applications of AI.

Every domain has legal exposure. When it comes to AI models, there are legal implications . In our case, regulations would be mostly based on many factors, some of the important being ethics, morality, and human rights that ensure public Safety and welfare. One cannot guarantee the accuracy of the model. The reason is simple, one can look at the output variables. It is a value that varies from 0 - one but never exactly 1.0 being least confident and 1 being the most. And while inference, the model almost gives out values less than 1 . AI models are nothing but probability machines. There is no pure surety even by the model itself.

The second exposure is how this tech is used and built ethically and morally.

Perfect analogy would be a pencil, doctors use to write prescriptions and research that save lives and then there is John wick aka baba yaga, the boogie man.

The reverse of our problem statement is also possible. One can generate protein sequences for diseases as target sequence and on the input side might be symptoms. One can create a model to generate output that consists of protein sequences of deadly diseases and the input to the model may be a customized set of symptoms the user desires that protein achieves. I did not believe this but There is something called protein engineering for generating custom protein sequences which is one of the core parts of drug discovery.

Authorities incharge might put restrictions on such models as we have experienced the wrath of SARS-CoV-2 aka covid 19 protein sequence which caused pandemic. Models like this can be used for bio-warfare. And there is a scary saying “It has to work only once.”

But for custom models to work, one needs extreme expertise and huge capital to build the entire project. In the case of pharma, the model needs to be of a certain size. This kind of capital is only available to the upper class echelon of the society. For references, let's ask Gemini about the average capital required to build such a model and the amount came out to be 5 to 15 million dollars which is basically huge.

Restrictions on the models by authorities in charge may vary based on the resources used to build it which is evident and easily detectable by taking a look at a balance sheet of any firm building AI models.

That’s it for the legal part.

If more research is done on the custom model and its architecture. It has a potential to achieve greater accuracy in domain specific problem statements.In fact, research is underway. During internet search for ideas, I came across a term called informers which are transformers for time series data analysis which can be used to predict weathers etc. Research is underway for different applications in different domains that use core concepts of “attention is all you need” generative AI.

For the future applications of generative AI.

Other applications can be risk assessment, fraud detection, supply chain optimization, content generation, image processing like converting words to videos, manga images to anime, and customized music creations, the list goes on..   
  
If you ask me, AI is just getting started and resource requirements will increase more than ever. Which includes, gpu, electricity, infrastructure to support it and so on.

That’s it for this video. We were able to cover many concepts from different domains, innovative ideas and AI regulation with legal exposure. And covered everything from our checklist. People with a 15 GB dedicated GPU ram gaming laptop could build and train a production level AI model in months that could be used to pitch VC’s, final year projects with real life applications or hackathons as proof of concept. Best of Luck!

The entirety of the project is posted on github. From dataset collected, code for data handling, cleaning, scraping, to model building, everything is posted there. Feel free to play with it, take it as homework and reuse the code to build AI models for different problem statements. Comment below. Any positive feedback, mistakes or suggestions are much appreciated that instigates curiosity.

This problem statement was inspired by a concept of mythical medicinal herb like Sanjevani which is believed to have life restoring properties reviving individuals from near death which has the ability to heal any injuries or illness.

I had a lot of fun with this problem statement.

#HaveFun

#MazeKaro