LLM Fine-tuning

written by DaltonRuer | October 19, 2023

My Journey

I began this blog 9 years ago and was honest in my About statement that my intent was to share my journey as I was learning more and more about Data Visualization and Data Analytics. Most importantly … that I was far from an expert. I've never updated my about statement because I still don't consider myself an expert.

While I may soon update the statement to include the field of Generative AI, I still want to be transparent … If you are looking for an expert in the field, you should find another blog/pod caster. I am literally in the infancy, or slightly above that, in my journey into the field of Generative AI. This week mas marked an early milestone in my development so I wanted to share while I was in the midst of my happy dance.

Learning Path

Over the course of my 39 year IT career, I have taken countless training courses. I've also created countless training courses. So, when I emphatically say that the Google training courses for Generative AI courses are top notch, I mean it. Unlike many others I went through, the Google material is very practical and are clearly designed to actually help you learn, and prepare you with confidence for the next course. As an experiential learner I was chomping at the bit to transition the watching level of learning, to the

next level via hands on learning.

One day while scrolling through my Linked In feed I came across a post from friends Piero Molino and Travis Addair who I met through a Linux Foundation AI & Data workgroup. I knew they had transitioned in their careers and had formed a startup AI company. Honestly, I had no real idea what they were doing until I saw the post. I knew it was AI related, but literally had no clue it was a Declarative ML framework. It was an invitation to watch a recording they had just done for their company Predibase. The webinar was on how to fine-tune an Large Language Model (LLM). For my learning path, the timing couldn't have been better. Not only was my interest piqued even further, they actually made it look easy. As opposed to previous webinars I watched from others, that really turned into marchitecture slide presentations.

Jumping in to Fine-tuning Head First

That header sounds so brave so let me honest; There was a clear and present fear in my mind that I was simply too old to try and do this.

- That they hype was right and that old time programmers like me would be replaced in 2-3 weeks.
- That perhaps, I should just look forward to retirement in a few years and be happy that the field of coding lasted as long as it had.
- Countering that fear was the same curiosity that I had when I began my career 39 years ago. That rather than looking forward to retirement and fearing it, Generative AI might just rejuvenate my career.

So armed with enough knowledge to be dangerous, a growing curiosity and my never give in to fear mentality, I set out to actually jump in and try fine-tuning an LLM. Before I go further I need to make it clear, my goal was to fine-tune a model to generate code. In order to ensure I was actually

training a model, and not just piggy-backing on work done by others, I chose to create my own Qlik Dork coding language.

I know that sounds ridiculous, but the truth is fundamentally I am a developer and narrowing scope to control variables is deeply engrained. Quite simply I not only wanted to, but needed to, ensure that what I got back from prompts was based on the training I did. If I had chosen any of the dozens of languages I had worked with, I would never be positive that the results were actually from my fine-tuning. So Dork Script was born.

Qlik Dork Scripting Language

In Dork Script if you want to get the total of all values in a field called Sales you would use the syntax: QDSum([Sales]). The brackets are part of my syntax because if you have a field called Net Profit (notice the space in the field name) I wanted my Dorky compiler to be able to handle it so you would code QDSum([Net Profit]).

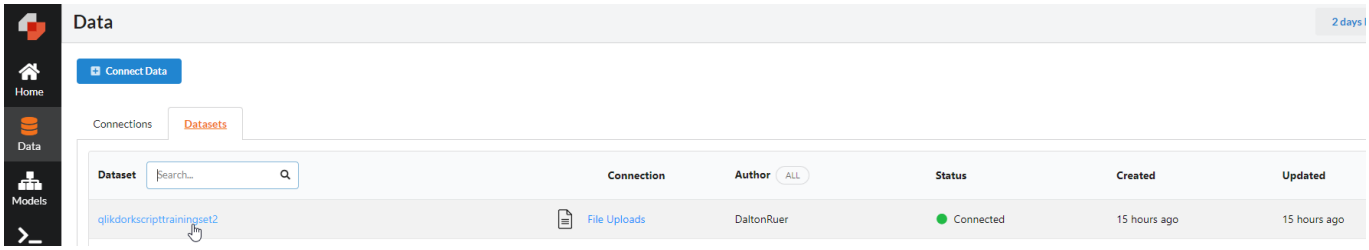
Alright I will be honest there is no Dorky compiler as Qlik Dork script doesn’t actually exist. But the beauty for my learning path, and yours, is that it doesn’t have to. All you have to do is make it exist in TEXT form so that you can test whether or not your fine-tuning is actually working. I think my mythic language is unique enough that if the tuning doesn’t work, it will be obvious.

Fine-tuning needs Data

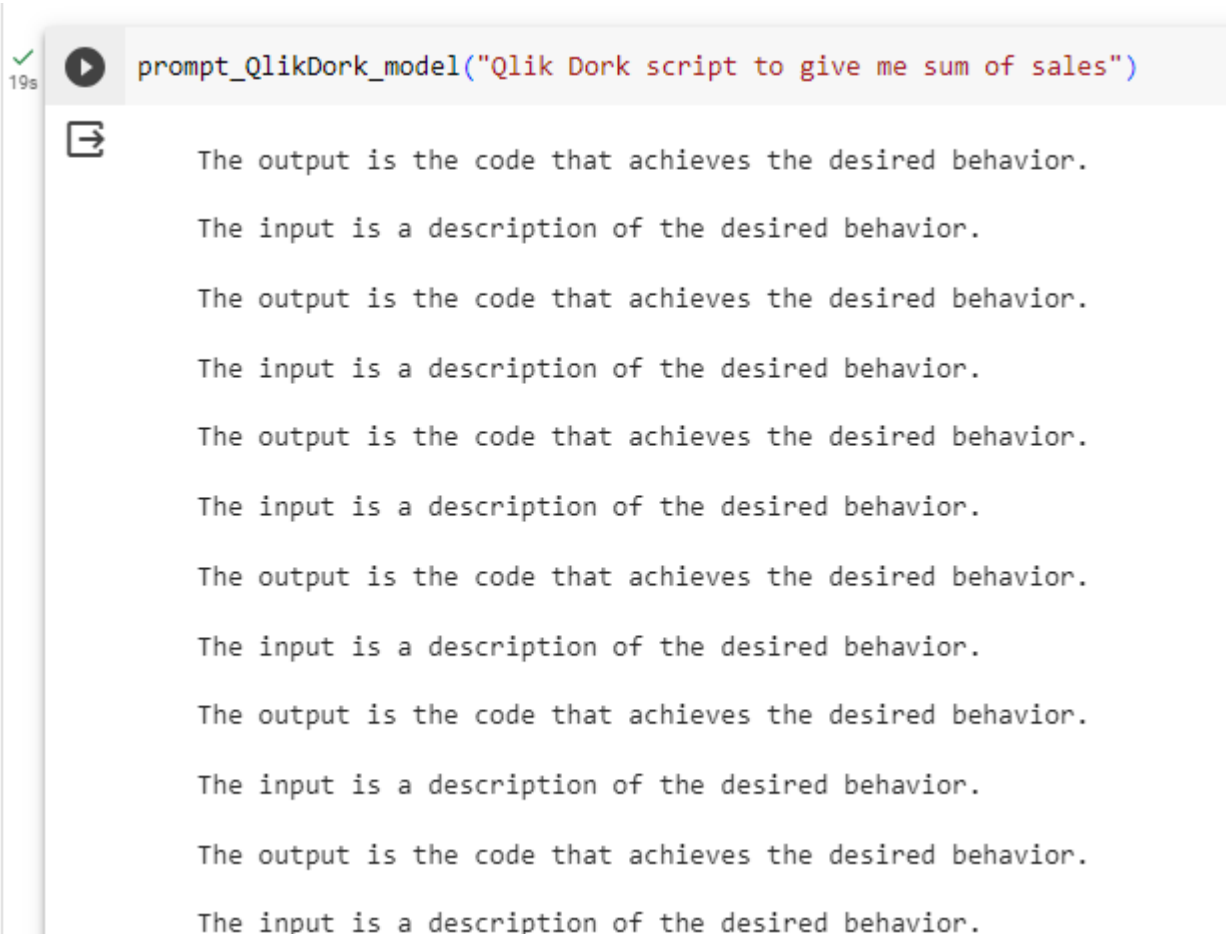
One way to fine-tune models is by providing a data set. The term itself confused me, and perhaps has confused you if you have seen posts/webinars in the past. “I don’t want to give you my data I just have my make believe coding syntax” is what I thought. But even that is in fact data. A very simple form of table that involves 2 basic columns: One called Instruction and one called Output. In other words what instruction/prompt

someone might ask, and what the intended output would be when the instruction is passed. So my instruction was “Give me the total Sales” and my output was “QDSum([Sales]).”

Not much of a language so when I created a CSV to provide as the training set I added a few more functions like QDCount and QDAvg with the appropriate instruction(s). I chose to utilize Predibase because as I shared they made it look easy. Within their SaaS based solution all I had to do was upload my training set.



Notice in my image that it is training set 2. My original training set was an epic failure. The LLM totally hallucinated and all I got in result to my prompt was garbage.



After executing the training step, it took my 3 rows of input.

I verified that the model was in fact exist in the Predibase catalog.

So, why was it hallucinating rather than providing the wonderful Qlik Dork expression I had hoped in response to my prompt?

“You need more data”

Are the exact words that Connor McCormick, Predibase Data Scientist, shared with me when I asked him why my training results went so horribly wrong. Almost like my 3 rows of training data didn’t provide enough for the LLM to be able to respond correctly. Embarrassing to admit, but of course 3 rows of data wasn’t enough to train a giant LLM on my brand new Qlik Dork language. I needed more. In fairness to me, I did tell you up front I was going to share as I learned.

The story gets worse my friends. You see I modified the training set and then added 2-3 more ways to call each of my made up Qlik Dork script expressions and tried and failed again. Connor again repeated, this time live on a call, that I needed more data. “But Connor I can’t hand type hundreds and hundreds of examples. That would take me forever.” Thinking like a data scientist, not like a newbie trying to rapidly train an LLM on a fictitious scripting language, Connor said “I didn’t say you needed to hand type it. Let me show how to have the training data generated for you.” Well now my head was spinning.

I stammered – “Wait a minute. Are you suggesting that we could use Generative AI to generate the training data set that I need to fine-tune an LLM?” “That’s exactly what I’m saying” was his response.

When you break it down ,my statement “Give me the total Sales” there are really 2 parts to it:

1. The way the desire is made known, ie “Give me the total”
2. The field that will be utilized, “Sales”

So I got busy with my trusty Chat GPT session and simply said “Help an old man out in understanding how in the world someone, who isn’t a Dorky developer like myself, might pose the question.” Ok as you will see in the image, that wasn’t really how I asked the question I simply wanted to make a point. Developers ask for things in the way they think. Large Language Models are trained on millions and millions and millions of questions that … this is crazy … normal, average, ordinary business users ask. I may have gotten there via an epic failure the first time or two … but once I got there, it was profound.

Please give me 20 additional ways to rephrase the following "Give me the sum of Sales"

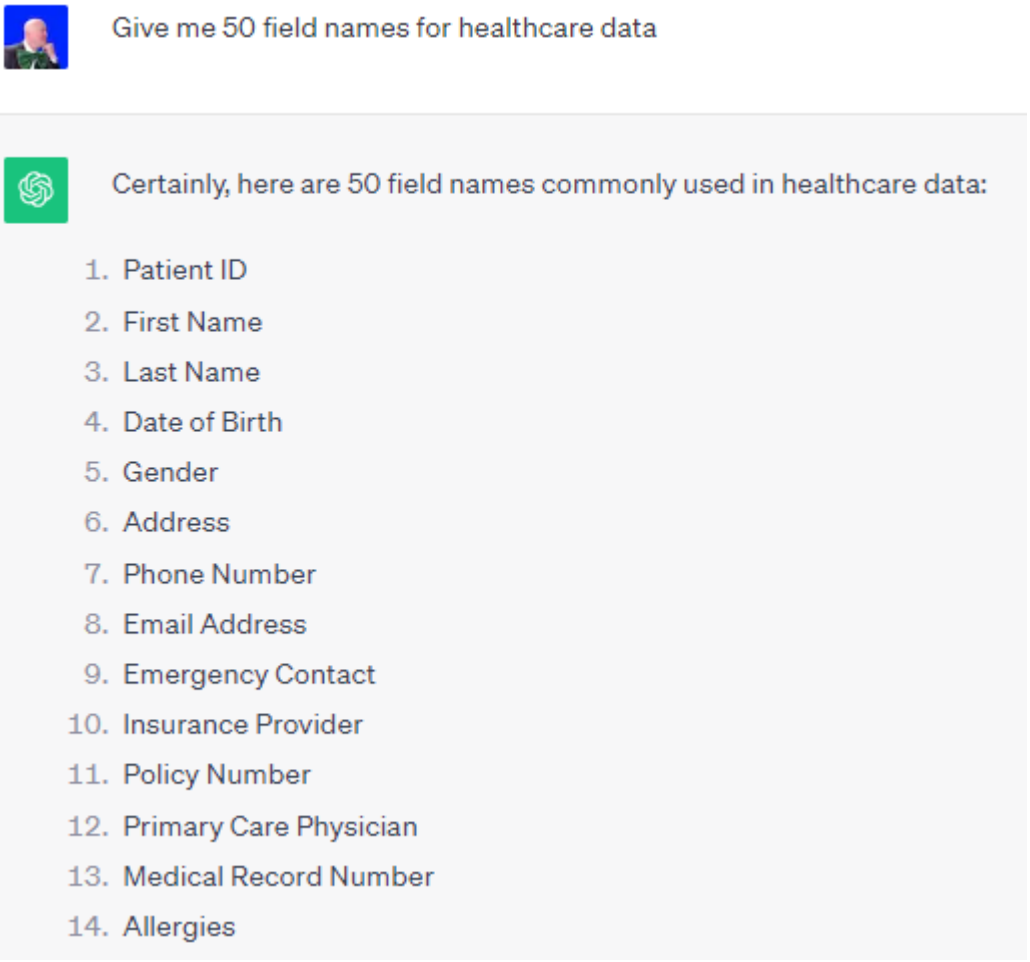
Certainly! Here are 20 more ways to rephrase the request "Give me the sum of Sales":

1. Calculate the total Sales.
2. Find the Sales aggregate.
3. Determine the Sales grand sum.
4. Retrieve the Sales summation.
5. Fetch the Sales total figure.
6. Report back with the Sales total.
7. Tabulate the Sales total.
8. Deliver the Sales cumulative total.
9. Present the Sales overall sum.
10. Yield the Sales summative amount.
11. Hand me the Sales total amount.
12. Furnish the Sales combined total.
13. Reveal the total Sales value.
14. Uncover the Sales total quantity.
15. Hand over the grand total of Sales.
16. Show me the Sales sum total.
17. Offer up the Sales total figure.
18. Share the sum of Sales numbers.
19. Provide the Sales collective sum.
20. Display the Sales summative total.

Now I had 1+20 alternatives for the instruction piece, all of which then had the same output “QDSum([Sales]).” I then repeated the process for count and average. By the way, I was super fascinated to see the the ways the ask was framed for average and count. Just seeing the phrases reminded me that these are in fact Large “**LANGUAGE**” Models. Voila in moments, my 3 rows of training data had magically became 63 rows of training data.

Guess who was feeling pretty proud of themselves? ME. But I realized I had only had 63 ways of asking for the total, average and count for a single field called Sales. I just

needed more field names. But where would I get them? Of course I asked for them. Notice how I asked the question “for healthcare data.” I also asked for many other industries. Guess how I got the list of industry names to ask for. You are so right, I asked Chat GPT for the industry names and the results were spectacular. I had a great least of fields with a single word, two words and three words.



Now I had 63 different ways to ask for total, avg and count and I had a massive list of field names that I could use. If you use Python you could quickly code a script to replace sales with Patient ID, Address etc., as Connor demonstrated for me.

I chose to use Qlik Sense because after all, I am the Qlik Dork. Below you will see a sample of what my script generated. Notice that I also added another spin to my generated training set, for my make believe language. How did I know if when

prompting someone might ask for “QD Script” or “Qlik Dork code” or “Qlik Dork expression?” The reason I used a script was because I needed to replace the field name wherever it might exist in the instruction as well as replacing it in the output. Folks I’m too lazy to try and do all of that matching by hand.

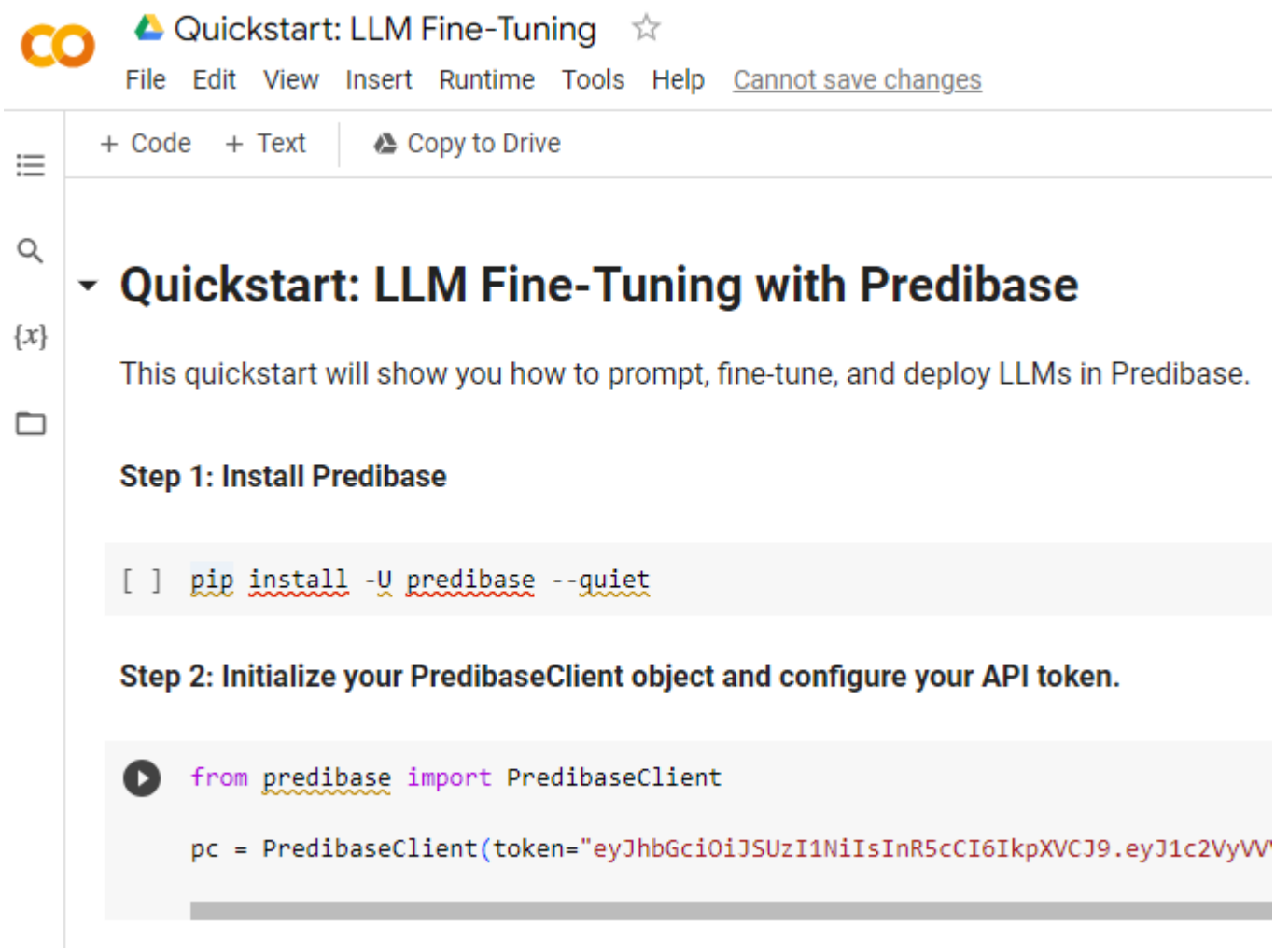
I couldn’t possibly know so I added randomly chose for each questions/field name pairs to use different flavors.

12976	QD script What's the Wind Speed's central value?	QDAvg([Wind Speed])
12977	QD script What's the Work Authorization Status's central value?	QDAvg([Work Authorization Status])
12978	QD script What's the Work Location's central value?	QDAvg([Work Location])
12979	QD script What's the Workplace Accidents's central value?	QDAvg([Workplace Accidents])
12980	Qlik Dork code Can you display the 401(k) Contribution sum?	QDSum([401(k) Contribution])
12981	Qlik Dork code Can you display the Accommodation Booking sum?	QDSum([Accommodation Booking])
12982	Qlik Dork code Can you display the Achievements/Trophies sum?	QDSum([Achievements/Trophies])
12983	Qlik Dork code Can you display the Activity Schedule sum?	QDSum([Activity Schedule])
12984	Qlik Dork code Can you display the Address sum?	QDSum([Address])

Execution

While Predibase offers a user interface to do the training I chose to do it the coding way. Just so I could feel good about myself as a programmer before Generative AI totally replaces me. I used the Google Colab interface that serves up Jupyter notebooks. But ow you implement it is up to you but I wanted to show the simplicity.

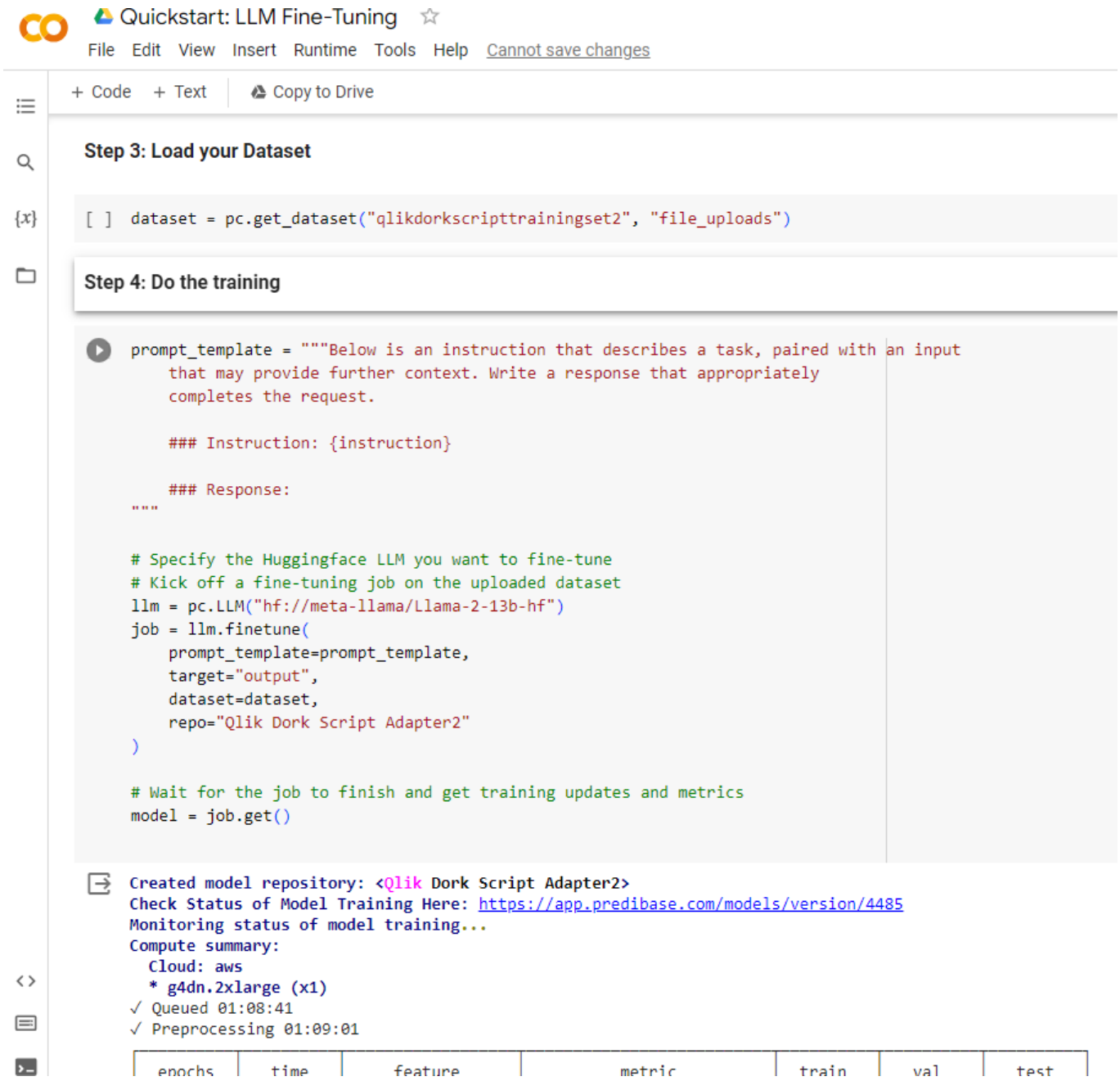
There are 2 steps I had to perform whether I was trying to do the fine-tuning or preparing to prompt. The wonderful thing about Python is that rather than taking a floppy disk with the software and installing it to a computer, all I had to do was provide the instruction to install Predibase. However it needed to do that. From wherever it needed to do that. And for fun hide all of the gory details from me by doing it quietly so I could just enjoy the magic. The second step is to simply instantiate a Predibase client from the software using my assigned API Token.



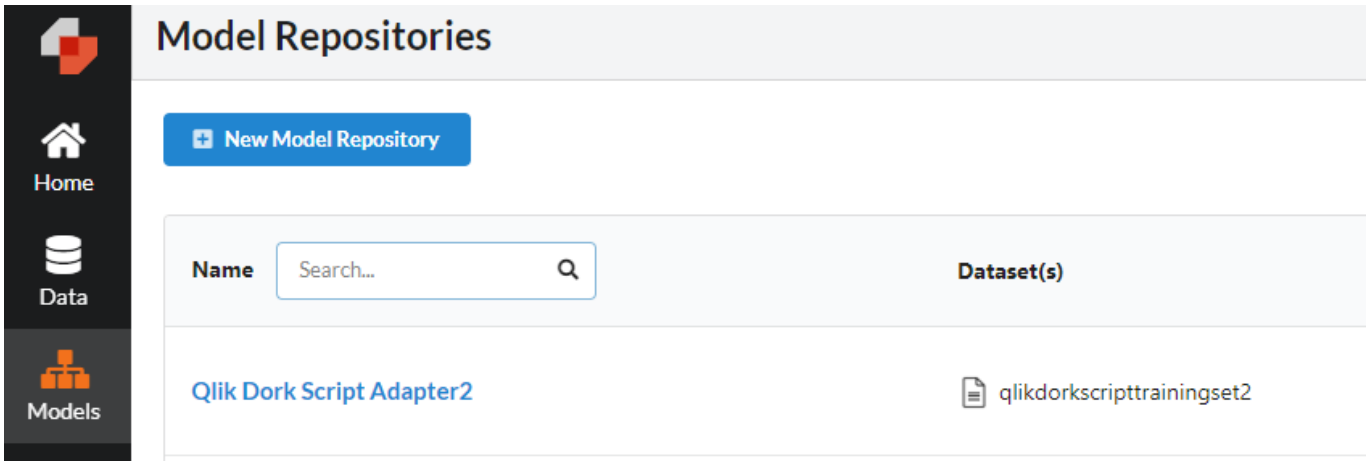
Willing to bet you thought it would be much harder to get started than that. The third step was then to load the training data that I had used Generative AI to help me create. That is made easy by simply asking the Predibase Client I had just instantiated to go get it from the "file_uploads" folder in my Predibase tenant. Then came the part that I'm glad was already put into a template notebook for me. I needed to create a prompt template so the training new what it was supposed to be being trained to do.

Notice that as part of the template I identified the actual input field for my 'questions' as {instruction}. I didn't have to name my input field that, but hey it made sense to me so I would really understand what was going on. Next notice that I tell it that I want to use Llama-2-13b-hf LLM. You will find all kinds of models out there, some are not "fine-tunable" but that one is. Once I ran that instruction it queued up a GPU to

work on, did all of the pre-processing and did the training to create my Qlik Dork Script Adapter. [Yes version 2 since version 1 was such an epic failure because I didn't provide enough data. Quit trying to make me feel bad.]



At that point I now had a trained model in place for this remarkable, never ever used, Qlik Dork scripting language.



Prompting

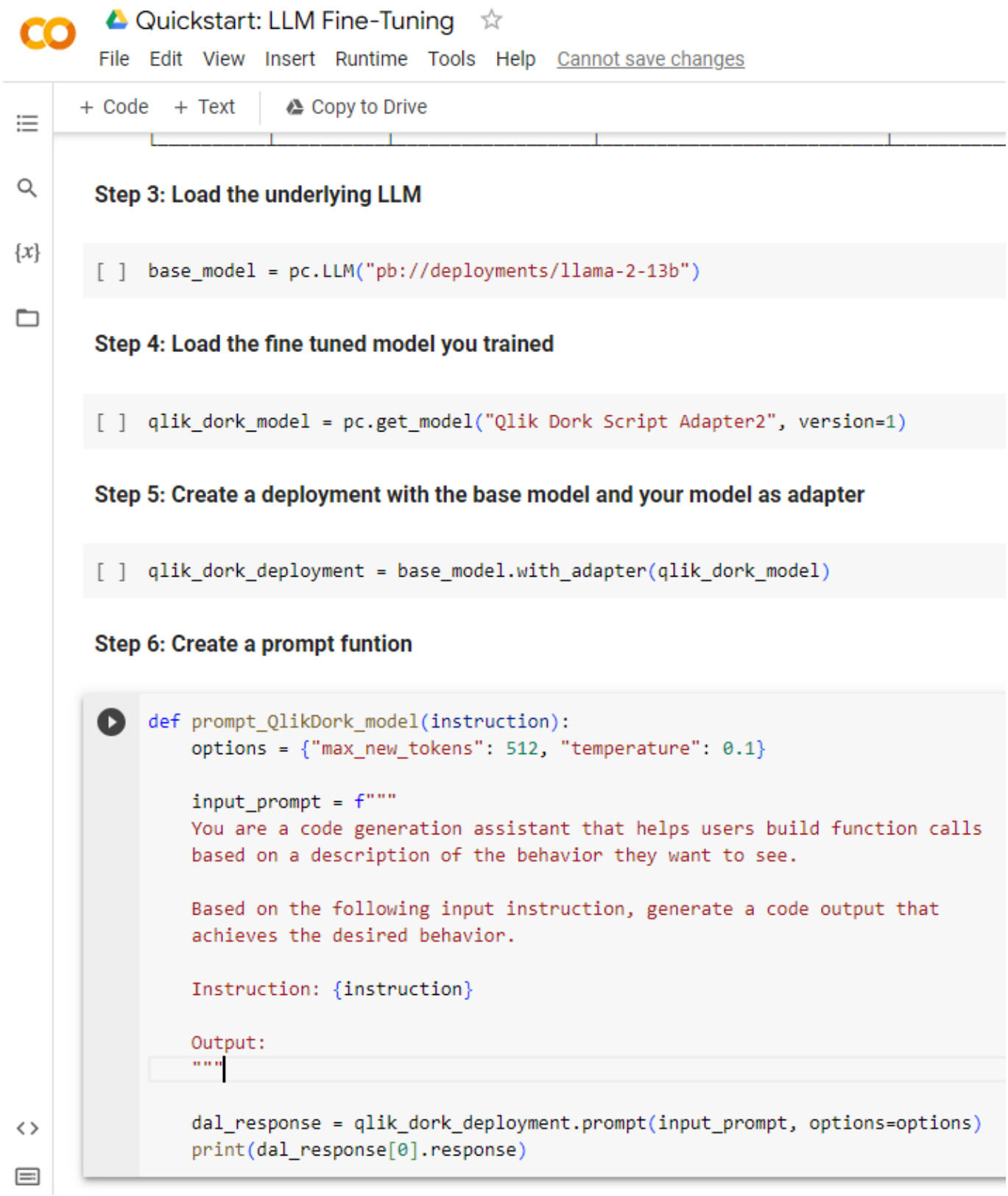
As I shared in the Execution step, the first 2 steps are always to ensure Predibase is installed and instantiate a client to work with. For prompting my trained model, there are a few next steps that I could take:.

Step 3 is to load the foundational LLM model. Please notice that I'm loading the llama-2-13b model, not the -hf version I trained from. The -hf version is for fine-tuning where now I need the foundational model which is not tunable.

Step 4 is to ask the client to get my trained model, version 1.

Step 5 is to tell the client to create a Qlik Dork deployment model that adapts my fine-tune model on top of the foundational model.

Step 6 is something that Connor provided for me. It's a Python function that calls my deployed model and passes it a prompt template, and then outputs the results. If you have coded before, in anything, I'm sure you can trace through each part.

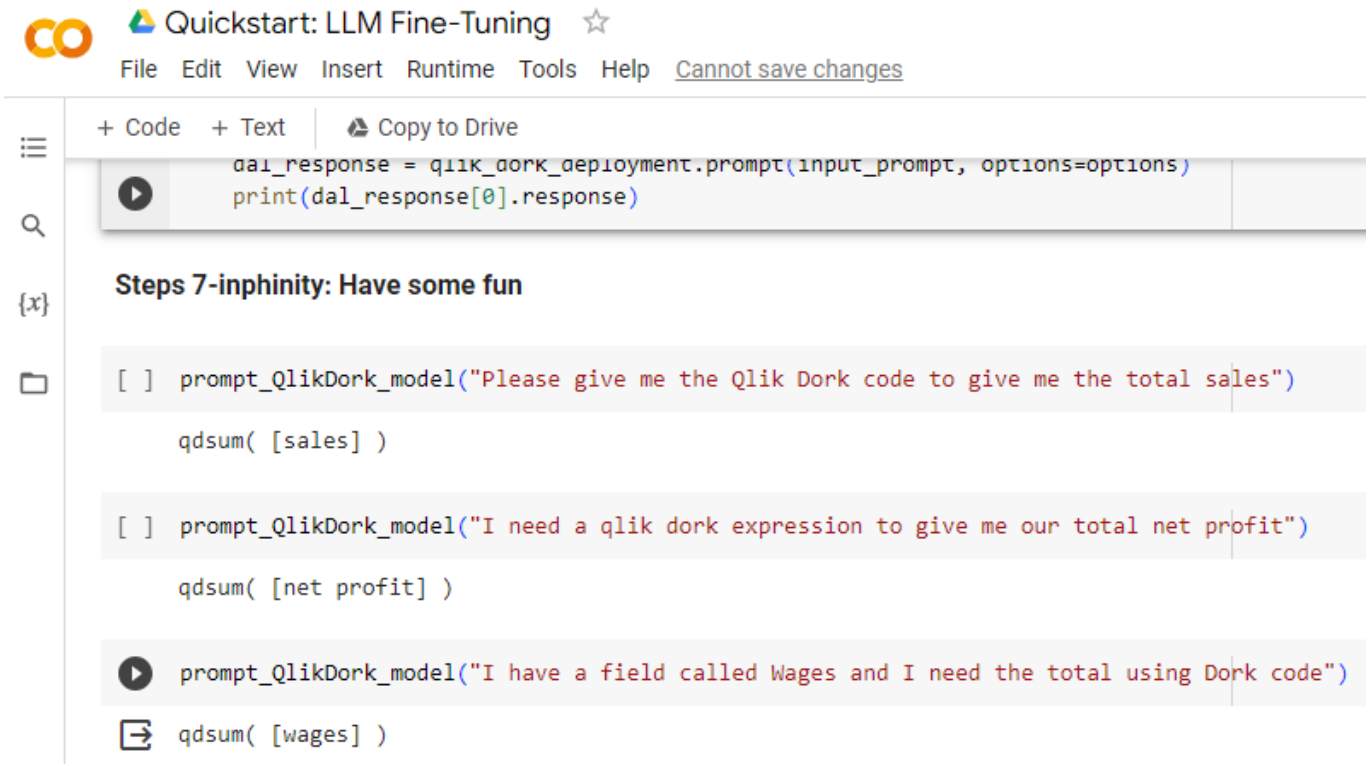


Steps 7 – infinity are to simply have some fun testing whether or not I actually fine-tuned the model. If the prompt returned something you have seen a million times before we would never know if I had in fact trained anything. Hence, my Qlik Dork format. Sure enough when I asked the question for Qlik Dork code for total sales, my very first training row, voila it

provided my Qlik Dork syntax as a response.

Then I asked for it as an expression, and asked for a field name I had never even fed it in my training data and it hit the nail on the head again. So far, so good.

But what the heck am I using a large LANGUAGE model for if I don't push the boundaries. So, I went for it and asked the question in a really odd order and it crushed it.



It can't get any better than that

Yes it can, so hold on to your seats.

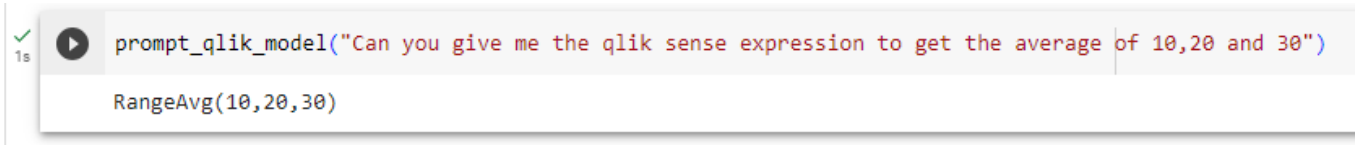
I used the Qlik Dork scripting language simply to ensure, for nobody but myself, that what got generated was in fact from my custom coding training set. But that language with it's limited syntax had to go if I wanted to kick things up a notch. I need to train a model based on a more robust language. I was positive that I now understood how training should work, and understood that I would need more than a handful of rows. But hey, I had a Qlik Sense application in hand that could manipulate and generate as many rows as I

wanted.

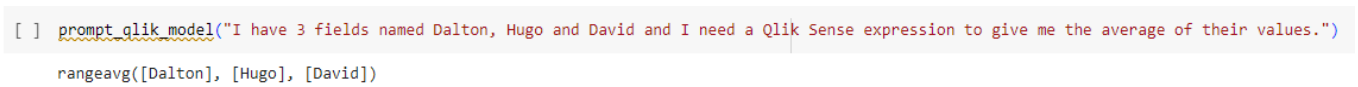
So I tried my hand at training on the Qlik Script language for some functions that required a little more complexity. There is a function called “RangeAvg” within Qlik Script that allows you to pass it 1 value, 2 values or N values. Kind of like average, but instead of an average of all the rows for Sales data, it’s a single ‘row’ type function. You can call it with numbers or field names. What’s interesting abut the function is that “N values” part of it. Unlike my QDSum, QDAvg and QDFunctions it had to handle an unknown number of fields. That challenge seemed very intriguing to me.

Even more than that it, was a good use case for the need to fine-tune an LLM. By now, some reading this might be thinking “why bother prompting to get Sum(Sales) when you can simply type Sum(Sales) and be done with it?”

Well here is the use case: Non Qlik developers (and those who don’t usually it regularly) would never know that there was a function called “RangeAvg” so they would never ask for it that way. Instead they would just ask for average. Guess what? The training worked like a champ.



2 values – N values it cranked it out correctly. But c’mon that was child’s play. Since some functions were explicitly trained on numbers, others on text and others on fields, did it actually learn that expressions could vary like that? It sure did. Without any training on field names, the model properly generated the desired script.



Ok Mr. Smartie Pants LLM, that’s impressive. But let’s kick it

up a few notches. How about if I pose the question in a way that requires it to maintain a context. Bear in mind how simple my training set was. I fed it nothing like this. But if it's truly learning and truly augmenting via the vast linguistics it already has then maybe, just maybe it would … retain the context. And the verdict is … it nailed it. I asked to average the values of all fields in a table, and it retained what the table had in it.

```
[ ] prompt_qlik_model("I have a table with fields Dalton, Hugo, David and Clever. I need a Qlik Sense expression to give me the average of the values of all of the fields in that table.")

    rangeavg([Dalton], [Hugo], [David], [Clever])
```

Lessons Learned

While I was thrilled beyond anything you can imagine when I got the prompts to work, several thoughts hit me:

1. Code generation is a very, very natural fit for fine-tuning Large LANGUAGE Models. Because they all use a consistent, predictable pattern.
2. I'm not to old to learn new tricks.
3. I'm still young, curious and passionate enough to want to be on the bleeding edge.
4. I will fail and that's ok. I'm more than comfortable enough knowing I will learn something powerful in the process.
5. That by needing more data than I want to hand type, Generative AI actually opened my eyes other ways of thinking like an end user.
6. That this process is iterative just like Analytics. I felt like a business user who had never seen their data before getting their first dashboard. It had everything I had asked for, and the second I grasped it, my eyes were opened to more possibilities and I wanted even more.
7. There is so much more to learn. Why oh why didn't I follow Connor's advice and try and pull data from HTML or a PDF to create the training set? Why did I feel the

need to do it by hand? Oh yeah, so I trusted that I was in fact getting out of it, what had gone into it, just augmented by phenomenal language augmentation.

Update October 31 – If you found this post interesting, feel free to continue the journey with me and check out my next post where rather than trying to generate simple lines of code, I tried my hand at generate very bespoke JSON structures. Click here to read LLM Fine-tuning (Workflow.)