Webinar: How (AI) Stuff Works – Understanding the key parts of AI simply and quickly
Date: June 11, 2026

WEBVTT

00:00:00.000 --> 00:00:03.000

Cool. Ontario, Ontario. Great. A lot of people from Canada. That's cool

00:00:03.000 --> 00:00:06.000

Yeah

00:00:06.000 --> 00:00:13.000

All right, I just heard recording in progress. I'm going to assume, Laura, that was you.

00:00:13.000 --> 00:00:14.000

Okay

00:00:14.000 --> 00:00:24.000

Yes, we are kicking things off officially. Welcome, everyone. Thank you so much for joining us for another SoftEd webinar. We're so pleased to have Brent Laster here joining us with the topic how AI stuff works

00:00:24.000 --> 00:00:50.000

Understanding the key parts of AI simply and quickly. And this is being recorded, and it will also be sent out afterwards, so if you would like the slides, you'll see that in an email follow-up afterwards. And we welcome your questions. Brent is willing to save some time at the end, so please feel free to chat those in or use the Q&A box to put your questions in. I'll keep an eye out for those

00:00:50.000 --> 00:01:09.000

So that we can really tailor this presentation to what you are most curious about and wanting clarity about. And so please feel free to, throughout, use

the chat box, let us know where you're from, let us know your LinkedIn profile if you'd like to network, please feel free to do so.

00:01:09.000 --> 00:01:27.000

We'd love to just make this what serves you best in terms of where you're at in understanding AI. And this is a very good starting place if you feel very new. So thank you all for making the time to be here. And with that said, I'll turn it over to you, Brent

00:01:27.000 --> 00:01:28.000

Yes, they are

00:01:28.000 --> 00:01:44.000

Well, thank you everyone for taking time out of your busy day to join us from all over the world. It's so great to see everyone or to virtually at least get to know everyone is here. We appreciate that. My name is Brent Laster

00:01:44.000 --> 00:02:00.000

And I am joining you from Cary, C-A-R-Y, North Carolina, just outside of Raleigh. I live around here with my, my wife and my kids are in the area, and my two dogs today are doggy daycare, so hopefully we… so we won't hear them, at least. So

00:02:00.000 --> 00:02:16.000

Hopefully we'll have a nice quiet presentation for you. But again, welcome everyone. This is being recorded, as Laura said, and we'll also have a PDF slides available to you afterwards. So this presentation is called How AI Stuff Works. It's one that I put together

00:02:16.000 --> 00:02:33.000

Some months ago, and did an initial version of it at a local AI conference, and Laura was kind enough to engage with me afterwards and say, hey, we would like to share this with a wider set of people there. And so I've evolved it some, but the goal of this is to try within the next

00:02:33.000 --> 00:02:50.000

50 minutes or so to help you understand some of these AI terms. You know, AI is becoming almost like utility these days. It's something that we all expect to be able to access, to have access to, but there's lots of terms that are tossed around, lots of these things. MCP, RAG, agents

00:02:50.000 --> 00:03:05.000

And it's not really clear necessarily what they mean. And so this presentation is designed to give you some analogies, some examples, some animations, ways to think about these things. As I like to say, to remove the mystery, to actually

00:03:05.000 --> 00:03:27.000

help us to conceptualize some of these ideas. So let's go ahead and get into it. Really quickly, just a little bit about me. There's information on connecting with me with a QR code and other links there on social media. Always happy to make new connections on social media if you're interested in reaching out, please feel free to. I have a long career in corporate as a developer, manager, director

00:03:27.000 --> 00:03:44.000

In those areas, doing lots of different things, and these days I have a training company, Tech Skills Transformation, and we work with organizations, companies like SoftEd to be able to bring quality training forward, and these, especially on AI topics these days

00:03:44.000 --> 00:04:00.000

you know, a soft ed is a fantastic company. From everything I've seen and talked with them about and done with them in terms of providing these kinds of training there and providing quality training programs. So we're happy to partner with them and be a part of working with them to bring these kinds of webinars

00:04:00.000 --> 00:04:17.000

and educational events to you. Also, have written a number of different books over time. If you're interested in any of these, they are available, most of them on Amazon, or also if you happen to have a subscription, like on the O'Reilly platform. O'Reilly has available to read on there as well.

00:04:17.000 --> 00:04:28.000

I want to take just a moment to call out that we are doing some exciting trainings coming up on the SoftEd platform. We have an Apply Claude Code course. This is going to be a 3-day

00:04:28.000 --> 00:04:46.000

Basically, four and a half hours a day that we're doing this when a couple of options for you in July and August. This is going to go all the way from the basics of Claude code going through to managed agents. We'll talk a little bit about co-work and give you just a good understanding and grounding in that every course that I do

00:04:46.000 --> 00:05:02.000

is hands-on labs as well as lecture, meaning that you'll get some practice in an environment using actual Claude Code, so you can internalize the skills and come away feeling more confident using the technology and understanding it. We also have one coming up for Applied GitHub Copilot

00:05:02.000 --> 00:05:20.000

and understanding MCP. The MCP one is just a short one session one with four and a half hours there, but the other two are going to be those three session things. I know that I think those registration links will be available later today or soon on the SoftEd site, so check those out if you're interested

00:05:20.000 --> 00:05:41.000

in exploring any of those topics. Now, let's get into what you came for. Let's start off by talking about large language models. Large language models are the way we call them LLMs, are the foundational pieces that make AI possible. Ever since ChatGPT, or rather OpenAI, I should say, put a chat interface in front of their OpenAI models back in 2022,

00:05:41.000 --> 00:06:00.000

We've been excited about these because we've been able to have conversations with the AI. We've been able to talk to it and get answers back. But we want to understand just for a moment kind of how these things actually learn. How does a model come together? What makes it able to converse with us and to have the knowledge that it does

00:06:00.000 --> 00:06:19.000

Let's do an analogy here. Let's think about from a standpoint in human learning, how we learn, and from an academic sense. You know, we think about, we all probably go to the school, come to college, something that we go through, a formal training process, and we are reading massive amounts of information, digesting it over our time

00:06:19.000 --> 00:06:34.000

While we're learning, we're making connections. Connections, some of them are stronger, some not as strong, but we're learning how things go together. For those of you who are programmers, or are in the IT field or project management, any of those kinds of areas there

00:06:34.000 --> 00:06:52.000

You know that you learn about what sort of concepts go together. How do we build code out? What kinds of keywords come together? Syntax, semantics, those kinds of things over time. We learn the patterns, we learn what fits together and what makes sense. And at some point, we are done with that formal training

00:06:52.000 --> 00:06:55.000

We have a cutoff from that, and we are actually

00:06:55.000 --> 00:07:11.000

Then, done with that training, and we, for that point in our lives. We have a certain amount, and we are graduated, and we're ready to go out and pursue other things. Now, when we talk about how large language models learn and predict

00:07:11.000 --> 00:07:26.000

How do these models learn about these concepts, the terms, the technologies, the words that we use, vocabulary and such? Turns out that they approach this, the approach is called training in a similar way that we go to school and we are trained on things

00:07:26.000 --> 00:07:43.000

These models are exposed to lots and lots of content, lots of things, the collection of books, collection of coding repositories, even things like

Wikipedia, all those kinds of things out there. And so they digest all this information, and they learn from the patterns.

00:07:43.000 --> 00:08:07.000

They learn from reading, essentially, from being exposed to the information, what concepts are related, and to be able to make sense of those concepts, they put them together or decide how similar they are, and kind of place them into this multidimensional space. Multidimensional space, for purposes of ours things here, we can think of it like those graphs we used to do in school, the X and Y axis

00:08:07.000 --> 00:08:23.000

You know, we have two dimensions, which ones and how things are close together there, but they have hundreds or even thousands of dimensions. They're mapping these things over. But to keep it simple, as an analogy, take a look over on the right-hand side. Let's suppose that as you're training, we have some very simple sentences here

00:08:23.000 --> 00:08:40.000

As we're training through this, the model training process is digesting these sentences, and as it's doing so, we can think of it as capturing these concepts and then placing them closer together. The more often it sees these concepts together

00:08:40.000 --> 00:08:55.000

The more it identifies these as, oh, these are things that are closely related and should be placed closer together in my space. So again, as it progresses over all of this training material, it's going through and deciding what things are related

00:08:55.000 --> 00:09:11.000

and how they are. Now, internally, this is done with numeric values, and this is done with things called vectors and embeddings that we'll talk about in a few minutes, but for our purposes here, we're just representing it as simple words to make it easier to kind of see the ideas between… behind that

00:09:11.000 --> 00:09:18.000

Now, beyond that, beyond just learning about those concepts and learning what things are similar in there

00:09:18.000 --> 00:09:36.000

And how they're related. It also starts to establish what we call weights. Weights are just the strength of the connection. You know, as we are going through and as we are learning things in school, we are also developing stronger connections, thicker connections

00:09:36.000 --> 00:09:56.000

between the neurons in our brains, between things in there, and understanding more about things that we see repeatedly, the patterns in there, and the models are doing that, too, as they're trained. As they are looking over these different sets of text, they are moving things around, and they are learning how strongly things are connected

00:09:56.000 --> 00:10:13.000

Those strength of those connections is what we call the weights in the model, the values that are persisted through that. And so when we are moving on then from taking those models from there, taking those words, we can then start to actually look at

00:10:13.000 --> 00:10:30.000

Having those connections, you see the numbers increasing if you look closely on here, and the lines getting thicker, representing that in this space, as we… as the model learns that things are routinely similar and concept there and closer together in the space

00:10:30.000 --> 00:10:48.000

the strength of the connection becomes stronger between that, and we develop those numbers to represent that. Now, having that sort of a model of the world, having that idea, the model has gotten about where things are, how similar they are, placing them in this multidimensional space

00:10:48.000 --> 00:11:05.000

there gives that model a framework, something to draw on. It understands how strong the connections are, and that becomes then the basis for prediction.

On this slide, if you look down towards the bottom right, you'll see we've got a little example there where we're going to run a prediction on this

00:11:05.000 --> 00:11:24.000

As we are reading, we can run, as the model is digesting information, we can ask it to make a prediction about what five words come next. Mouse hit, mouse shed garden. Now, this isn't very intuitive, but it's a very small sample set. But the point is that as it digests more information.

00:11:24.000 --> 00:11:43.000

You can see the prediction changes a little bit. Mouse hit, mouse cat chased. Again, not really kind of a little bit of nonsense in there, but you've got the different terms that it's running over time. The point of this is that as it digests more information, as it learns how things are connected, as it strengthens those weights between things

00:11:43.000 --> 00:11:55.000

then it actually learns about what concepts are connected over time, and it actually uses that to be able to predict the next word. It learns that

00:11:55.000 --> 00:12:11.000

Mouse and cat are more similar than maybe mouse and shed, for example. So it can figure out how to predict the next most likely token. Beyond that, though, when we actually work with this, when we actually do these kinds of things.

00:12:11.000 --> 00:12:27.000

We can think of what the model produces, what comes out of the training as a set of weights. What we mean by that is this idea that as these models are learning and being trained over these data sets there, they're making these connections

00:12:27.000 --> 00:12:45.000

They're having, they're reading huge amounts of text, and we can think of it almost as the weights being dials. Kind of we dial things up a little bit to mean stronger connection between concepts, or dial them down a little bit to have less connection between them. So as we are moving, as we are digesting these things, and as we are

00:12:45.000 --> 00:12:53.000

doing those weights, we're also adjusting those kinds of the kinds of the sliders here

00:12:53.000 --> 00:13:09.000

Let me see if I can get that going again, the animation. So ultimately, then, those sets of weights, how those dials are adjusted, how the model has learned the different kinds of connections between things, those weights become what gets stored

00:13:09.000 --> 00:13:27.000

And what is unique to the model itself. Now, again, this is a very simplified version of things, but ultimately, those weights, those strengths of the connections that the model has learned, that's what we end up with in model files, and the files that actually

00:13:27.000 --> 00:13:44.000

have the data from the training. That's what's unique about it, how the all the learning that it did in understanding how things are connected, in being able to position things in that space so it can represent them, in being able to draw strong connections between things

00:13:44.000 --> 00:14:00.000

or weak connections. Those weights are the key. And your input then goes through those sets of weights for the model. Every model sets of weights are different, and then produces an answer. Your input runs through that, and it actually then produces an output.

00:14:00.000 --> 00:14:16.000

And so, we might have, for example, a 7 billion parameter model. You'll hear these models talked about in terms of the number of parameters they have. One of the key parameters is the amount of weights, the amount of connections it has made over time

00:14:16.000 --> 00:14:32.000

usually expressed in billions of parameters, and that translates to certain amount of storage per weight out there. Just numbers, no words or sentences, but those weights are the transformations. So really, when you are… when you're asking a model to

00:14:32.000 --> 00:14:49.000

take input and produce a word. It's taking your word, and we talk about it in terms of tokens. A token is more like 3 quarters of a word, really. But it's a token that goes through and goes through… is multiplied by those various weights, doing lots of math on there

00:14:49.000 --> 00:15:04.000

That's why we have to have very powerful processors like GPUs and such graphical processing units, and producing a set of most likely tokens from that. We can think about it then, for example, if we were to say

00:15:04.000 --> 00:15:20.000

The cat sat on something, right? Then taking those words, those initial values, and putting them through the wait process, transforming all the weights that the model has learned, then might say the most likely term is Matt, the most likely thing

00:15:20.000 --> 00:15:30.000

The most likely token there, the most likely prediction is Matt. Next one might be Rug, or Sky, or the cat sat on the dog, in some cases. But you see

00:15:30.000 --> 00:15:46.000

These have different probabilities. So, inputs then flow through the weights of the model, and they get manipulated. This is a single layer. These models have lots and lots of layers in there, and even billions of weights. So again, lots of numeric processing going on

00:15:46.000 --> 00:16:06.000

And lots of information that the model has learned. But by taking those weights that that particular model has learned, we have the input, it goes through, and it produces a most likely token, a most likely token to come out of there. And this is a prediction based off of what the model has learned, what it's been exposed to

00:16:06.000 --> 00:16:24.000

Now, a key point about this is that the model is not going out there and looking up things on the fly, checking a source, knowing where the answer is really true or not. In some cases, the chat interfaces that we get through there, in some cases, the chat interfaces

00:16:24.000 --> 00:16:39.000

That we see on top of these models will have ways to go through and verify information on the fly, but the models themselves don't. That's why sometimes you'll see models hallucinating, making up plausible sounding next words

00:16:39.000 --> 00:16:54.000

They are producing something, predicting something, based off of what they've been exposed to, based off those weights and what they've been trained on, not based off of any kind of fact-checking they're doing on the fly. Models want to be helpful. They've been trained to be helpful and provide an answer

00:16:54.000 --> 00:17:12.000

But they're not going back and checking it, so we do have it sometimes making things up and doing what we call a hallucination. And all models do that. We do have tools on the front of them now, a lot of cases that help prevent that. But now training has limits. So how do we augment that training?

00:17:12.000 --> 00:17:29.000

Let's talk about that. So we know that we have our training when we go to school, college, we have a cutoff. We have a cutoff, models are the same way. If you were to take just a model without any kind of other interface on top of it, talking to it, for example, about a programming language.

00:17:29.000 --> 00:17:44.000

A model would have a cutoff at a point in time, and it would not know that things had been deprecated in a programming language after the fact, or new features had been added. It would only know about what was current or previous point in time it had the cutoff

00:17:44.000 --> 00:18:02.000

So, if we want to go back and actually update a model's knowledge, or actually get additional knowledge for the AI to work with, we have a couple of options. We have things called retraining. Retraining is essentially like you going back and doing another degree, going back

00:18:02.000 --> 00:18:20.000

And going through another round of formal education to get a whole new set of knowledge and kind of re-apply that. We retrain the model from scratch. This becomes a very expensive, very time-consuming proposition, which is why you see usually these models coming from these bigger companies that have the resources

00:18:20.000 --> 00:18:38.000

to actually go and train them. But we can also do things like fine-tuning, which is kind of like the equivalent of us taking an extension class, or even maybe learning from a webinar like this. Tweaking certain small amounts of knowledge in the model there, tweaking our knowledge to add in some additional information there.

00:18:38.000 --> 00:18:55.000

not retraining, not going back and doing the whole education from scratch, but going back and just adding on to it, adding on to the training there. The other way to do it, and both of those, by the way, essentially reset the clock. What I mean by that is that they actually just say

00:18:55.000 --> 00:19:12.000

We're going to add training, but our training still has a cutoff that point. The other approach is what we call using RAG, and RAG, as we'll talk about in a moment, is akin to the idea of being able to go out and search for the knowledge ourselves. Think about in your role

00:19:12.000 --> 00:19:31.000

in your job, or even doing, like, a search out on the web or something, finding more information yourself and adding that onto your knowledge. You have some knowledge based off of your formal training, and then you have additional knowledge that you've gone out and researched yourself and found the most relevant information

00:19:31.000 --> 00:19:47.000

and then made decisions on that. So we can find and include relevant data and make it… and it's less costly, more doable on demand. Now, when we want to use that data to determine what data is relevant for RAG, we first have to get our data into a form

00:19:47.000 --> 00:20:03.000

that is searchable for similarity. What does that mean? It means that we're not just doing the Google sort of keyword search, we're actually using some mathematics to figure out, based off of this entire collection of data that we have. Maybe it's our policies and PDFs

00:20:03.000 --> 00:20:15.000

Maybe it's our code bases based off of that collection, those collections of data, what things are most relevant, most similar mathematically? Let me show you what I mean by that.

00:20:15.000 --> 00:20:34.000

Now, we'll just do, you know sort of an imaginary raising of hands here, but how many of you have used a color picker before my color picker, I mean an application in some environment where you got to choose what your color was. You wanted to pick a certain color, and you would have, yeah, Laura says she has

00:20:34.000 --> 00:20:49.000

I think most of us have at some point, and you have the green to blue and the red. So a couple of characteristics of this. I've seen the hands raised. Great. Thanks, folks. Appreciate the engagement. Yeah, so the idea here is that we've got this, this

00:20:49.000 --> 00:21:05.000

multidimensional cube, right? A cube here. And we've got blue, red, green values in there, and we know a couple of things about this. We know that the colors have multiple coordinates, right? An amount of red, an amount of blue, an amount of green in there, three different coordinates here

00:21:05.000 --> 00:21:22.000

And we also know that colors that are more similar have similar coordinates, like light blue and blue will be closer together in the space and have

similar coordinates than blue and red. Blue and red are further away. So that's essentially kind of what we do with embeddings

00:21:22.000 --> 00:21:38.000

Embeddings turn your text into a long list of numbers, a vector. Our vectors here have three dimensions, but vectors in AI can have hundreds or even thousands of dimensions. We're going to keep it simple and just have a very simple way to think about it

00:21:38.000 --> 00:21:58.000

Text with similar meaning ends up closer together, just like colors here, blue and light blue are similar, and so they are closer together in the space. Similar idea. Embedding AI models are models that have been trained to learn how this closeness is, and to put things closer together in that space

00:21:58.000 --> 00:22:16.000

And so we can take these concepts that the AI learns, or we can take the concepts even from our own documents, even from our own code bases, and we can say, we're going to put these concepts into this space so that the AI can understand, or we can understand

00:22:16.000 --> 00:22:32.000

Which things are similar based on their how close together they are in the space. And we can use these types of embeddings. Think of embeddings as just turning your text into this series of numbers, these vectors, these coordinates in space. If we had the text blue

00:22:32.000 --> 00:22:52.000

We might represent it here as 00255, and the text read as 25500. Taking that concept and putting it into a space in multiple dimensions. So let me give you kind of a simpler picture of this. Let's say that we have over on the right-hand side, our X and Y, right? These things we've done in schools

00:22:52.000 --> 00:23:09.000

We position points on them, and we see how close the points are, and we have coordinates there. Let's say we take some basic words, king, queen, and lunch. You know, as the old saying goes, I forget if it was Sesame Street or

Electric Company. Sorry, I'm dating myself here, but the idea, you know, one of these things is not like the others

00:23:09.000 --> 00:23:27.000

in there. So let's say we're going to just, say, have some coordinates for king and queen, representing them as numbers, or a two-dimensional vector in this multi-coordinate space. And lunch is there as well. So maybe king and queen are over here, and maybe lunch is somewhere down here.

00:23:27.000 --> 00:23:45.000

So then we've got a couple of words that are close together, and one that's not so close together. And realize I'm just making up these coordinates, but again, you have hundreds or thousands of these in place. Now, notice at the very top up here, we've got a little, we've got a little thing that says cosine similarity.

00:23:45.000 --> 00:24:03.000

Fancy-sounding term simply means that we've got a mathematical formula that's going to measure the angle between these. If you think of these things having multiple dimensions kind of shooting off in all directions in that, you know, like a three dimensions there, you can measure how close they are by how the angle between them.

00:24:03.000 --> 00:24:21.000

So, let's just keep it simple here, though. Notice that king and queen, when we just start out, have a 0.96 cosine similarity. All that means is that the math has said, hey, the angle between them, they're pretty similar. They're pretty close together in the space. Now, notice what happens if I start measuring the angle between

00:24:21.000 --> 00:24:38.000

at them in other words. As we get further away, like down to lunch, we get down to, like, a negative 1, basically, cosine similarity. Not very similar at all, much further apart. So, using these formulas, these math formulas, like cosine similarity.

00:24:38.000 --> 00:24:54.000

allows the allows us to be able to tell how similar two things are when they're placed in this kind of a space like that. And that's one of the key things that we work with when we start working with RAG, and I'll explain how that works as well

00:24:54.000 --> 00:25:11.000

Now, as I said, in reality, there would be hundreds or thousands of dimensions that we're working with instead of just two dimensions or three dimensions. It's hard to visualize, but I wanted to give you a little bit of an idea. Maybe you can think of this as the Death Star of words, or the giant

00:25:11.000 --> 00:25:31.000

parable of words in there. You can kind of see on here the idea that these terms, these would be actual vectors or numbers, but if we think about them in terms, they're out in these multiple dimensions there. Again, how similar they are is… means how close they are in the space, or how far apart they are from other things.

00:25:31.000 --> 00:25:46.000

So by representing these terms in this kind of multidimensional space, in these kind of vectors there, we can gauge through math how similar things are or not similar. All right, so how does all of this then play into RAG?

00:25:46.000 --> 00:25:50.000

For RAG, what we do

00:25:50.000 --> 00:26:12.000

is we take our data and split it into chunks. Let me tell you what RAG is at a high level. RAG simply means that we are going to take our own data, our PDFs, our code bases, whatever it is, and we're going to put it in this space so that we can figure out what things are most similar to our query.

00:26:12.000 --> 00:26:28.000

Think about the model from the AI model, as we said, the model has a set of kind of formal training and a cutoff, right? It doesn't know about our policies. It doesn't know about our code base directly. It knows what it was trained on

00:26:28.000 --> 00:26:46.000

But it doesn't know about our internal, our enterprise data. It hasn't been trained on that unless we went back and retrained or fine-tuned, which, as we talked about, aren't always practical with that. But the models that we use do not know about our data. They know about the public data they're trained on with that

00:26:46.000 --> 00:26:48.000

So what RAG provides us with

00:26:48.000 --> 00:27:04.000

It's a way to take our data and put it into a searchable database so that we can find the most relevant things from our data, and then we can take those hits, those most relevant pieces, and we can tell the AI

00:27:04.000 --> 00:27:20.000

Please consider our data in addition to your training data. It's like us going out and researching something on Google, right? We go out and find information that maybe we didn't know about before, but we go out and find the top couple of hits out there

00:27:20.000 --> 00:27:37.000

of relevant information, and then we say, oh, taking that into account with what we already know, we can get a better quality of answer. So the RAG process. RAG stands for Retrieval Augmented Generation, but the setup step for RAG is taking our data

00:27:37.000 --> 00:27:55.000

converting it to vectors and storing it into a searchable database, which is called a vector database. Here's kind of what that looks like. We take our PDFs, for example, and we chunk them, we divide them into chunks. Chunks simply means a section of the text that would make sense to us

00:27:55.000 --> 00:28:15.000

If we got it back from a search, think about it like a sentence. Think about it like a paragraph, maybe a table in a PDF. We chunk it into pieces. We take

each of those chunks, and we use an AI model that understands how to turn it into these vectors. So it understands how to turn it into

00:28:15.000 --> 00:28:31.000

Vectors of information. Again, vectors are just numbers that represent that concept in that big space, like we had the king, queen lunch, and we had X and Y coordinates. But think about, instead of two coordinates, hundreds or thousands of coordinates there

00:28:31.000 --> 00:28:48.000

And so then we store the vectors in the vector database, and that is then searchable by saying, take a question, turn it into a vector, and find the things that are most similar, just like if you were searching for a color, like, let's say, green.

00:28:48.000 --> 00:29:05.000

in that color picker there, you might say the most light, the most similar colors are light green, maybe dark green, those kinds of things in there. So it's a similar idea. We have all of our terms, all of our chunks, again, a sentence, a paragraph, a table

00:29:05.000 --> 00:29:22.000

that are stored as numbers in the vector database, and we can go and say, given some kind of a query, what are the most similar things? Now, this isn't really using AI other than just to transform things into numbers. Vector databases are just a searchable data store

00:29:22.000 --> 00:29:39.000

that the AI helps us to produce the vectors on. We could do the same thing for source code. Source code might be a line of code, a function of code, it might be even a whole file. We can take that and turn it into a chunk, chunks of code. Again, think of a chunk as something that if we got that back

00:29:39.000 --> 00:30:00.000

it would make sense to us. If we ask, for example, from the vector search, from looking up things in there, where do we do authentication? And we got back a chunk of code that was an authorization function in there. Or if we

ask about a policy, and we got back a section, a paragraph on what our HR policy on PTO was.

00:30:00.000 --> 00:30:15.000

Those could be chunks there, right? Sections of the documents. Now, we can take that again, we put it through an AI model that understands how to make it into these vectors, and it gets stored in the vector database. This is just the setup for RAG. This is the setup part

00:30:15.000 --> 00:30:31.000

Putting our data into one of these vector databases. There are a number of them out there. There are dedicated ones called ChromaDB, Weviate and stuff. Even though things like Postgres, Postgres has a PG vector extension that allows it to manage vectors and search for things.

00:30:31.000 --> 00:30:49.000

All right, that's our setup for RAG. So what actually happens with RAG? The easiest way to think about RAG is like an open book exam. Instead of answering from memory, or instead of the LLM, the AI just answering from what it's already been trained on, we go and look up information. We decide

00:30:49.000 --> 00:31:06.000

What stuff is relevant in there? And it's like an open book. We can find the relevant pages first, we can hand them to the model along with the question. So our question comes in, we search our documents in the vector database, find the most relevant sections, the most relevant chunks.

00:31:06.000 --> 00:31:22.000

And we add that to it. Again, kind of like the idea of searching over the pages of a book that's separate from our training, finding the relevant pages, and then going and adding that to our collection of knowledge, reading from our actual documents

00:31:22.000 --> 00:31:38.000

So, to give you a better sense, to give you kind of an idea of getting your data ready, think of it this way. Think of it as we are reading over our documents, splitting them into chunks on the right-hand side, we have policy, HR policy, engineering design, and financials

00:31:38.000 --> 00:31:58.000

And again, for purposes of simplicity, we're representing kind of grouping these together in that multidimensional space. As it is reading these things, it is taking them, running them through a model that then puts them into that space. Sorry, I went through that a little quickly there, but you get the idea. We split up our data into chunks

00:31:58.000 --> 00:32:15.000

We store it in these vector databases so we can search it there. The question was, and there was a question that says, do we chunk it using some kind of tool? Yeah, we use tools that actually can turn in… there's all sorts of chunking strategies and all sorts of libraries,

00:32:15.000 --> 00:32:33.000

programmable toolkits that you can use to chunk it into different ones, and then to split it up that way. And there's lots of different chunking strategies on there. Yeah, Patricia, but we do have… we do have tools available to do that. So we go through and do that. So, now, here's how RAG all plays together.

00:32:33.000 --> 00:32:49.000

We have our data. We have our data that's internal to our enterprise, and we actually have then stored it in this vector database. We've stored it as a series of numbers. We split it up into reasonable pieces, and then we've stored those pieces as numbers or vectors

00:32:49.000 --> 00:33:05.000

in there, and the vector database. That's what this, this thing with the three circles here represents. And then we've got our model. Now, remember that our model does not know about our data. Our model has training, but it doesn't know about our data in there

00:33:05.000 --> 00:33:20.000

So, let's suppose that the… so the retrieval augmented process goes like this. The first thing is, let's suppose somebody comes in, we built a chat bot with RAG, and so somebody comes in and says, I want to know about our overtime policy for contractors

00:33:20.000 --> 00:33:38.000

What happens is we take that query, we turn it into its own vector, we turn it into a series of numbers. Using that same kind of AI model that can do that, and then we say, find the top 3, usually 3 is a good number. Find the top 3 pieces that are related

00:33:38.000 --> 00:33:49.000

And tell us how they're related. Remember that cosine similarity? Like, we're between king and queen at lunch we saw in there, too? So there's math processing that goes on, and it comes back and says.

00:33:49.000 --> 00:34:05.000

It says, I looked in your searchable vector database, and I found these are the items that were most related to this. There was a thing from HR Policy on page 2 about time tracking, a thing about leave requests, a thing about benefits enrollment

00:34:05.000 --> 00:34:20.000

It is doing a similarity search. This is the hardest thing, or one of the hardest things about RAG. It's not a keyword search. It's a mathematical similarity. We use the math formulas to find the most similar things. That's where you see these cosine values there

00:34:20.000 --> 00:34:35.000

You can see some of them are, like, 80.83, 0.79, 0.74. Those are mathematical values that say how similar was this chunk of data that we had from our PDF documents

00:34:35.000 --> 00:34:37.000

In there

00:34:37.000 --> 00:34:57.000

We have from that, that we have from our PDF documents to the query that we had. So somebody asked about overtime policy for contractors. Here's the chunks we found in our searchable database that were most related. That's the retrieval part. Retrieved data from our searchable vector database.

00:34:57.000 --> 00:35:12.000

The next part, the A in RAG is augmenting. What does augmenting mean? It simply means we add those chunks to the prompt. The prompt is our key, just like you're talking to ChatGPT, and you prompt it and ask it something

00:35:12.000 --> 00:35:27.000

We're going to ask it about overtime policy for contractors, but we're also going to take the information we found from our vector database search. We're going to add that into the prompt. We're going to say, in addition to your own training, please consider this information

00:35:27.000 --> 00:35:39.000

Please consider these chunks of data. So then what happens is we add that data, we add that data from we found directly into the prompt. We inject it into the prompt, and then we send it over to the model

00:35:39.000 --> 00:35:50.000

And we can tell the model either answer the question based on just the data we found from the vector database, or in addition to

00:35:50.000 --> 00:36:06.000

So in this case, we have some data that's specific to our documents that we retrieved out of our database. So it might say, hey, here's some information on our overtime policy for contractors, and it might fill it in with some of the pieces of information that we got from our database

00:36:06.000 --> 00:36:21.000

Models are very good at phrasing words, doing natural language, but it also might say, here's how over time works in general. So the idea is that some of the answer came from those chunks of data that we found, and we supplied to the model

00:36:21.000 --> 00:36:38.000

Some of it was specific to our documents, and some of it came from the models training itself. So that's the generation part. So RAG is retrieve

information from our searchable database, from our documents, or whatever we put in their code bases, find relevant chunks

00:36:38.000 --> 00:36:56.000

We don't want to have to put everything in there, because that would be just every time having to parse everything again and going through that. So we find the most relevant pieces. That's the retrieval. We add those to the prompt, and then we generate a response on there. And that gives us that grounding in our information.

00:36:56.000 --> 00:37:14.000

Let's move on then and talk about AI agents. Are the chunks permanently added to the LLM once it's loaded? No, the chunks don't actually get loaded to that, Aaron. What happens is the chunks get just, are just in the vector database

00:37:14.000 --> 00:37:30.000

They can get added to the prompt, but they only exist in the prompt unless we go back and retrain the model with that data or fine tune it. Otherwise, they only live in the vector database and they just get added dynamically to the prompt when we do the RAG operation

00:37:30.000 --> 00:37:48.000

Let's talk about what makes up an agent here really quickly. An agent is a collection of things. It's an AI model that we use as the brain. The AI model can actually do things like planning how to approach things, planning the steps, being able to cause things to happen, to execute in there

00:37:48.000 --> 00:38:05.000

And having code around that and having tools that it can call. So with an agent, then, what we have is a model that serves as the brain. It's good at text interpretation, it's good at planning, figuring out what order to do things in. It's good at asking that tools be called on its behalf

00:38:05.000 --> 00:38:22.000

AI models do not have the ability to cause tools or to actually invoke APIs. By tools, think about an API. Think about searching the web, think about looking something up. They don't have that ability to do that themselves.

Now, they may have a chat interface on the front of it that can get the information

00:38:22.000 --> 00:38:41.000

But out of the box, the models can't do that themselves. So we use an agent to say, we're going to have code that works with the model to call tools on its behalf. So what happens is the workflow code can take input, for example, from things. It can say the model can say, I would like to have this tool called

00:38:41.000 --> 00:38:56.000

And then it can go off and cause things to happen. The key part of an agent is that it has agency. We give the AI hands, so to speak. We give it the ability to cause things to happen and to interact with the real world. How does that happen?

00:38:56.000 --> 00:39:16.000

Let's do a quick analogy here of how we do tool calling in humans. Let's suppose that you're a manager, and you've got a boss that's a director, and you have an employee named Joe. And Joe has told you at one point, hey boss, I've got a tool here, if you ever need it, I can get some raw numbers for you from the meetings. So the director might come back and say, hey, we need a report

00:39:16.000 --> 00:39:18.000

with numbers from your departments for the second

00:39:18.000 --> 00:39:34.000

order for the all-hands meeting. I don't know if this ever happened to you, but when I was in corporate, we would sometimes get requests, you know, to have numbers gathered about stats that we could present in the all-hands meeting for that. So that then, when you're working on that, if you're the manager.

00:39:34.000 --> 00:39:50.000

Your thought might be, well, I can't generate the report myself, but I know that Joe has a report generator tool that can get the raw data for me. He

told me he has, so I'm going to send a message over to Joe, and I'm going to say, please run your tool to get the raw data. Here's… I know you have a tool

00:39:50.000 --> 00:40:07.000

Here's the specs I would like you to run with on it. And Joe says, cool, boss needs some numbers, I'll go and run my tool, I'll get the answer for the boss, and I'll reply back to the email. I'll reply back to the message to the team's message, whatever. Here's the raw numbers you wanted, boss

00:40:07.000 --> 00:40:23.000

Here you go. Now, as the boss, you are as the manager, you might say, that's great, but I can't just hand these back to the director. I need to put them in a presentable form. Remember that AI models are very good at doing natural language processing

00:40:23.000 --> 00:40:41.000

So, we could then say, we take that, and we take that value, and we give it back to the director. So think of… keep that human sort of example in mind. Let's think about how tool calling happens in agents. I said model, let's just say with agents, really. If the agent instead is… do we have a user instead of the director

00:40:41.000 --> 00:40:57.000

And instead of the manager, we have an AI model, and instead of Joe, we have our code. And instead of a tool, we have a weather API. Let's say we're going to have a weather agent here. We might have a we have a system prompt, just as the Joe said to the manager, I have a tool that can do X

00:40:57.000 --> 00:41:14.000

We talk about agents, we have a system prompt. The system prompt is the definition of how the AI model should behave. We can do things like say you're an AI assistant designed to help users find weather conditions. Your primary goal provide precise, helpful, and clear responses.

00:41:14.000 --> 00:41:31.000

That helps the AI model zero in on the part of its training about weather and how handling weather. We can also, though, tell it it has a tool. You have a tool called FindWeather. Now, that doesn't mean the model can invoke the tool

itself. Rather, where you have told the model, if we tell you you have a tool definition

00:41:31.000 --> 00:41:48.000

You can ask the code to invoke this tool on your behalf, just as we could ask Joe to run the tool for us. And then also, we have this piece here that says, you should think step-by-step. Now, most models these days already do this. Most models will already think and plan this out

00:41:48.000 --> 00:42:04.000

But for some smaller models, we may need to tell it, think step by step. Figure out what you would do first. Decide if that has happened or not, if you're successful, and then go on to the next step. This is what defines an AI agent. We give it a high-level goal

00:42:04.000 --> 00:42:19.000

It uses its tools, it uses things at its disposal to decide how to accomplish that goal. So the way this works in practice, the user says, what's the weather in Paris? The model says the user's asking about the weather. I need to extract Paris

00:42:19.000 --> 00:42:36.000

And I want to call a tool. So it puts out a message in its output, and simply says, please call this tool for me, and it can't call the tool itself, but it can get the latitude and longitude, maybe by calling another tool, maybe it's got something built in from its training that it's learned

00:42:36.000 --> 00:42:52.000

And so the code can go in and say, hey, the AI model in its output has sent message and said, please call this tool. I'm going to parse that output. I'm going to go and actually call the API. I'll take turn that tool call into an API call. I'll get the information back

00:42:52.000 --> 00:43:09.000

I'll reply to the message. By reply to the message, we just add it back on. Think of the interaction you have with ChatGPT. You ask a question, it provides a reply, you reply back. You have that thread of conversation. This is just code sitting there interacting with the model

00:43:09.000 --> 00:43:26.000

And saying, here's the response I got back. So we send that back to the AI model, the response, the AI model looks at it, says, oh, cool, I've got the results from the code running the API, but that's not really usable by the user, so I need to make it into a nice readable user

00:43:26.000 --> 00:43:42.000

So it comes back and says the current weather is 13 degrees Celsius with light rain. So that prompt thought, action, observation, result. The AI models are trained to be able to ask that tools are called on their behalf. They're trained to be able to say

00:43:42.000 --> 00:43:59.000

Please run this tool for me. The code in the agent reads the output from the AI model and says the model wants this tool to be run, goes off and runs the tool, and just appends the result back like you're replying to the message there, and then the model can take that and work with it.

00:43:59.000 --> 00:44:17.000

The model doesn't ever call the tool, it just asks that it be called and gets the result back. All right, I know we are going fast here, but let's talk about one more topic, model context protocol, MCP. So it turns out, surprise, surprise, that these companies that produce the models

00:44:17.000 --> 00:44:32.000

As we said, they train their models to be able to call tools, but guess what? They didn't train them to be able to call tools in the same way. They trained them to say, we can invoke these tools differently. So when the agent started being written before MCP

00:44:32.000 --> 00:44:50.000

You had agents, you had APIs, right? The tools that you wanted the AI to be able to use, to be able to ask to be called. But you had multiple ways of calling the APIs. You also had multiple APIs. So what happened was, if I wanted to call my agent your APIs

00:44:50.000 --> 00:45:07.000

And another company's APIs, and another company's APIs, I had to write a lot of code for that. I had to write a lot of paths to be able to call everybody's tools. And it was a very brittle process. You ended up with a very precarious sort of stack of things where the code could easily break because

00:45:07.000 --> 00:45:25.000

If the API definition change, then your agent broke because it was no longer right in there. So we came around with for Anthropic, the company that OpenAI, I'm sorry, Anthropic, the company behind Claude, and Claude Code invented or came up with model context protocol

00:45:25.000 --> 00:45:40.000

Model context protocol, just a standard, just a definition. And what the definition, though, does, though, is it says in front of these tools, we're going to put a server, we're going to put a server that speaks this thing called MCP.

00:45:40.000 --> 00:45:56.000

Model context protocol. And we have an interface then. So this server on the back end knows how to call each APIs. You put a server in front of your APIs, I put a server in front of my APIs, we speak MCP. Anybody who has a MCP client

00:45:56.000 --> 00:46:13.000

can talk to your server or my server. As long as we all speak MCP. Your AI app, most commonly your agent, can go in and has a client, and that client can go over and talk to the server. So anybody who writes an AI agent where the agent manages the model

00:46:13.000 --> 00:46:29.000

Can write and have a client. The client can go off to the server and the server can call tools. So the client can ask that any tool be called just by using the MCP language. It can discover that as well. Now, you'll see a lot of times the USBC

00:46:29.000 --> 00:46:47.000

As kind of the common court, or the kind of common specification for this example, I like to think of it kind of as a contractor and a specialist, or a subcontractor. Here in the Us. If we use a contractor, we use a contractor to be able to talk to them and ask that things get done

00:46:47.000 --> 00:47:03.000

They, in turn are able to speak the language of the electrician, or the builder, or the plumber to get things accomplished. MCP is kind of like that. If you speak MCP from the client side, your AI can ask that your AI model

00:47:03.000 --> 00:47:18.000

can ask that a tool be called, it gets translated into MCP, and then it can go off to the server and say, please call that tool. Now, the nice thing is, the AI app implements the client side and the server side is just sitting in front of the tools

00:47:18.000 --> 00:47:36.000

As long as you speak Mcp, your agent, you can call my server tools, and vice versa. The other thing that goes with this, though, is that a discovery process. What that means is the contractor, the of the agent side can go out to a public MCP server

00:47:36.000 --> 00:47:38.000

Like one that has

00:47:38.000 --> 00:47:53.000

I don't know, GitHub MCP server. They can have tools, they can have resources, resources are just static data, or even prompts, and they can say, tell me about what you can do for me. And the MCP protocol can come back and say, I've got tools that can search the web

00:47:53.000 --> 00:48:14.000

Or I can compose a report for you. You might have an internal MCP server. That can be a way to actually roll out AI. You might have an HR MCP server where people can ask about how much PTO they have left, or an IT one that they can ask about how to reset their passwords. And it can say, I can interact and manage with our databases

00:48:14.000 --> 00:48:30.000

You might have an external company. So we can use MCP, it can automatically connect, discover the tools, and then automatically call them. There's no brittle coding in there, there's no having to know how the API works in there

00:48:30.000 --> 00:48:40.000

There's none of that. It's just MCP handling it all. So we could immediately start using the different tools and the different things in the server.

00:48:40.000 --> 00:48:55.000

Now, how does this work in practice? Okay, this… let me kind of tie this together for you. On the user side with the agent, we'll have our agent, we've got the AI model, we've got the code that interprets what the model has said. We've got our system prompt

00:48:55.000 --> 00:49:07.000

Notice that our system prompt has a place for tools. Remember in the example we had earlier, we had a tool definition to get the weather. We had it hard coded. When you're using MCP

00:49:07.000 --> 00:49:23.000

We can use the client to go off and discover tools. So our MCP server, for example, might implement a tool called GitWeather. It might have a weather API on the back end that it can call, and it has a definition. This app server tool

00:49:23.000 --> 00:49:38.000

That is a definition that declares that the server has this particular tool available. Notice it has some text. It has some text to discover it. Remember that AI models work well with text, so we can give the tool, we can give the model a description of the tool

00:49:38.000 --> 00:49:54.000

The model can turn around and say, oh, I need to get weather. I should ask this tool to be called. So what the client does in this case is it reaches

out first, and it says, please tell me, Mr. Server, or Mrs. Server, about the tools that you have. And the server side says, hey

00:49:54.000 --> 00:50:11.000

I have a tool that you might be interested in called Get Weather. Here's the tool I have. The client says, cool, I'm going to take that and I am going to inject it into my system prompt so I know I have that tool. Great. That's how it discovers. It asks the server

00:50:11.000 --> 00:50:27.000

What kinds of tool do you have? The server says, I have the tool does this. The client on site says, oh, cool, I'll put this into my system prompt. I've got that available. The AI model then, if the user says, what's the weather in Paris? They can go out and it can say, I need to get the weather for Paris

00:50:27.000 --> 00:50:43.000

So, oh, by the way, I've got a tool for that. Remember that models have a cutoff in their training. It can't determine what the current weather is in Paris. Now, if you have a chat bot or something on the front of it, yes, it can have tools to do that. But this is another way to have the tool in the agent.

00:50:43.000 --> 00:51:01.000

So the model can say, I've been told that there's a tool I can ask to be called to go out and get the weather. So I'm going to ask that this tool be called, just like sending a message. The code reads it, says, oh, the model's asking me to call a tool. I'll go out and I'll send to the client, please call this tool on the MCP server.

00:51:01.000 --> 00:51:16.000

This time the server actually invokes the tool, runs the API, gets the response back, sends it back over to the client. The client says, cool, here's the response AI model, just like appending or replying to an email

00:51:16.000 --> 00:51:36.000

The model says, cool, here's the response. I'll put it into a nice format for that. So the difference is, instead of having that tool hard-code it, yes, all that happens in seconds very, very quickly. The difference is, instead of

having to have it hard-coded in the agent, having to know the exact API call, having to know the exact format and how to process everything

00:51:36.000 --> 00:51:52.000

We delegate all of that to MCP. The server can be maintained by somebody else. The server can be maintained by a company or an organization, and to find the tools, MCP can be used to discover the tools, get the information, and execute them

00:51:52.000 --> 00:52:04.000

All without you having to know all of the critical details when you put your agent together. Final piece, we're going to tie this together. AI in your day-to-day. What were these pieces show up at work? Let's take a

00:52:04.000 --> 00:52:20.000

So we have LLMs and models, we have the chat bot itself. This is the underlying models in their chat GPT. Let's say open AI. We'll say the models behind the chat interface is really Claude or Copilot summarizing writing questions

00:52:20.000 --> 00:52:35.000

A bot that knows your company, you know, embeddings and rags, something where you've trained, you've actually been able to chunk your data, and you've been able to put it into a searchable vector database. An agent that is an AI assistant that does the task, books the meetings, drafts the emails

00:52:35.000 --> 00:52:51.000

NMCP being able to plug into your apps, into your MCP servers. So, with RAG, we build our knowledge base. Let's say we've got our email data, we've got years of history in our email, we're going to have the situation is processing email

00:52:51.000 --> 00:53:09.000

So we've got a series of email here, we've got contacts, we've got rules, we've got calendars, lots of things that we can learn from. We have our data. We can take this data, we can turn it into chunks. Maybe the chunk is an email, maybe it's a paragraph, maybe it's a calendar invite

00:53:09.000 --> 00:53:27.000

And we put it into these vectors, and we preserve the context. The vector database is about chunking your data, but it's also about placing things in a way that we can… it can understand how things are related. Then, if we want to go and store that in a vector database

00:53:27.000 --> 00:53:43.000

We can search for things, we can search our information, right? Not just what the model learned, we can search from other information that we've got in there. So we could retrieve information, we could say, who are our VIP contacts? We could take that, we could turn that question into embedding

00:53:43.000 --> 00:53:54.000

We could then search and get the top hits, and we could use that information back in… pull the top matches from that. So how does all this then tie together?

00:53:54.000 --> 00:54:12.000

With that information, we can have our agent sitting here in the middle that thinks and acts and checks repeat. Remember that an AI agent, the idea is we give it a high-level goal. Maybe that high level goal is clean up my inbox and flag anything urgent. And then it has tools, it has MCP servers

00:54:12.000 --> 00:54:28.000

It maybe has RAG vector databases at its disposal. It can decide what tools to call, how to access data and get it, so it can go off and it can actually talk to the RAG and look up things like VP, VIP contacts, find the most relevant chunks again, what's most relevant

00:54:28.000 --> 00:54:46.000

get past email patterns, check your priority rules. It can talk to an MCP server that connects to your Gmail interfaces, or to your calendar, or to your Slack to get information. And then it can go through, read each email, decide what to do with it, maybe it marks it high priority

00:54:46.000 --> 00:55:02.000

Maybe it says, yeah, we've already handled this one. Maybe it says this one can be deleted, this one could be high priority, but it's one prompt the agent orchestrates the RAG provides the knowledge and MCP allows you to connect the tools

00:55:02.000 --> 00:55:18.000

So, getting the most out of this, a few habits to help make these tools more useful, give it context. The more detail you provide, the better the answer. The more information you can provide, either through, like, RAG, having your documents parsed, embedded through that

00:55:18.000 --> 00:55:33.000

or even sometimes attaching things in there, then you'll give it information. Treat it like the sharp new colleague who wasn't in the room, or the intern, the new person in your group, who is very competent, very capable, but does not have all of the background and all the experience

00:55:33.000 --> 00:55:49.000

Give it clearer prompts, give it more direction, more context in there. Trust but verify. AI can sound confident, still be wrong. It can make up stuff. It isn't checking real time in most cases. So check anything that matters. It's got to be subject to the same review

00:55:49.000 --> 00:56:06.000

The same kinds of tests, those kinds of things. Connected up to real data, hook it up to your documents and tools. Hook it up to your documents through RAG. Use the RAG process to embed your documents, have them available to it, use the vector certs, the retrieval part of RAG to go out and search relevant things

00:56:06.000 --> 00:56:22.000

Also connect it up to your tools, your MCP. Have it able to do your APIs, or to talk to your databases, those kinds of things, and make it a conversation. Don't expect one perfect answer. Ask, refine, redirect. It gets better as you go.

00:56:22.000 --> 00:56:39.000

How it all fits together, models trained on massive debt… massive text data can understand language, embeddings in RAG, convert your data to vectors, retrieves the most relevant context, and injects it into your prompts. AI agents use the LLM as a brain to observe, think, and act

00:56:39.000 --> 00:56:55.000

Mcp is a standard protocol, so any AI can discover and use any tool. Each layer solves a limitation one before it. Together, they can form a complete AI system. All right, I know that was a huge amount of information to dump on you

00:56:55.000 --> 00:56:59.000

I do think we do have a few minutes for questions, Laura, if you're good with that

00:56:59.000 --> 00:57:10.000

Yes. Yes, I noticed that you answered a few questions that were in the chat as you were going along, but I'm not sure if you saw the Q&A tab yet. So

00:57:10.000 --> 00:57:11.000

I haven't, sorry, I, yeah

00:57:11.000 --> 00:57:15.000

Yeah, there were a few there that I can read out.

00:57:15.000 --> 00:57:17.000

Yeah, if you would, that'd be great.

00:57:17.000 --> 00:57:34.000

The first one is, and this was early in the presentation at 12:16 PM, a question. So I'm not sure what they're referring to, Tom, if you want to put in the chat some context, but the question is, is this similar to the way a perceptron machine works?

00:57:34.000 --> 00:57:35.000

Which

00:57:35.000 --> 00:57:50.000

Perceptrons, I'm guessing, Tom, you're talking about basically like, yeah, like a neural network in there. It is similar to the idea of these AI models use neural networks that are modeled kind of after the neurons. In other words, they make strong connections between things

00:57:50.000 --> 00:57:51.000

And they can decide. Yeah. Yep. Cool.

00:57:51.000 --> 00:58:01.000

Okay, I see a thumbs up and a clapping, so I think that made sense. Next question, do we chunk the data using some kind of tool or is this a manual process?

00:58:01.000 --> 00:58:21.000

Yeah, Patricia, I think we talked about this briefly before, but basically, yes, you can use… there are all sorts of chunking strategies. Again, chunks could be sentences, they could be lines of code, they could be paragraphs, whatever makes sense. There are lots of toolkits that help with that, and there are also lots of the tools, like Claude Copilot, or I'm sorry, GitHub Copilot, Cloud Code

00:58:21.000 --> 00:58:26.000

They will do their own indexing and chunking in the background. So yes, there are lots of tools to do it. Yeah.

00:58:26.000 --> 00:58:33.000

Okay, great. Next question. Do the current LLMs extend their knowledge by using a sophisticated RAG system?

00:58:33.000 --> 00:58:51.000

Yeah, good question, Tom. The LLM's knowledge is not extended in the sense that it doesn't get new knowledge embedded in it. It's just the additional kind of thing of being able to look up like something like you look up

information in a book or do a Google search. It augments maybe is a better word

00:58:51.000 --> 00:59:01.000

It doesn't change the LM's knowledge. It simply goes and just supplies extra knowledge at the prompt time. I think Tom has a hand raised there, but

00:59:01.000 --> 00:59:02.000

Okay.

00:59:02.000 --> 00:59:09.000

So a tool like Copilot or Perplexity identifies its RAG resources by including web links

00:59:09.000 --> 00:59:26.000

It doesn't necessarily identify RAG sources by including web links. Most times there's going to be some kind of a data store. If you're using RAG, you have a data store that's been set up. For example, if you're working on

00:59:26.000 --> 00:59:41.000

a code base in Visual Studio Code. The Visual Studio Copilot, or GitHub Copilot, rather, will index that code base on its own to produce the RAG source. Now, some things will do some RAG lookups and stuff

00:59:41.000 --> 00:59:58.000

If they don't find a quality enough result, they will fall back to going out and searching the web. But it's usually more of a fallback to say, I didn't find something relevant enough with a high enough cosine similarity score, so I'm going to go out and search the web and see if I can find a better answer

00:59:58.000 --> 01:00:04.000

Okay, great. Last question. Do MCP clients cash discovery results?

01:00:04.000 --> 01:00:19.000

They should, yeah, they absolutely should. That's one of the things, that's one of the keys about using MCP. Just like any other sort of programming interface, caching is a key aspect of that. So if they find a result, and the same question comes back

01:00:19.000 --> 01:00:33.000

then yes, they could have caching built in. It's usually more a result of how the framework that you use to implement MCP. There's a thing called fast MCP, and also an SDK. Those kinds of tools will have caching functions built into them.

01:00:33.000 --> 01:00:39.000

Great. Thank you so much and seeing some great feedback in the chat, some compliments to your presentation, Brent. So we really do appreciate you.

01:00:39.000 --> 01:00:42.000

Well, thank you so much, everybody. Yeah.

01:00:42.000 --> 01:01:00.000

Yes, I know it's at the top of the hour, but I will send the recording to everyone and emails with links to the slide deck and the upcoming courses that Brent will be facilitating. So please do reach out if you have feedback ideas for future webinars or needs for training

01:01:00.000 --> 01:01:01.000

Absolutely. Thank you, everybody. Thank you.

01:01:01.000 --> 01:01:06.000

And with that, Brent, if you'd like to close us out, if you'd have any closing words to close us out

01:01:06.000 --> 01:01:10.000

Just appreciate everybody joining, hope it was useful, and I hope I get to talk to you with you in a future training. Thanks, everybody.

01:01:10.000 --> 01:01:18.000

Thanks, everyone