# Evals for large scale classification: #24

69 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=5FY0HBZYDUU
https://www.youtube.com/watch?v=5Fy0hBzyduU

SUMMARY

*The episode features a discussion on applying AI engineering concepts to real-world projects, focusing on dynamic user interfaces (UIs) and large-scale classification systems. The hosts, By Bob, Dex, and guest Kevin Gregory, explore how to build effective AI systems that can be deployed in production, emphasizing the importance of iterative development and user experience.*

- *Introduction of the hosts and their backgrounds in AI and software engineering.*
- *Discussion on dynamic UIs and their potential to enhance user interaction with AI systems.*
- *Kevin Gregory shares his experience in developing a classification system for a hardware store with over 1,400 categories.*
- *The team emphasizes the importance of breaking down complex problems into manageable components for effective AI development.*
- *They discuss the iterative process of refining AI models and UIs based on user feedback and performance metrics.*
- *The conversation highlights the subjective nature of "correctness" in AI outputs and the need for flexible evaluation criteria.*
- *They explore the use of LLMs (Large Language Models) and embeddings in narrowing down categories for classification tasks.*
- *The episode concludes with insights on the balance between accuracy and cost in AI implementations, advocating for rapid prototyping and user-centered design.*

There you go. >> What's up, guys? How you all doing? >> It's I'm going. We are in for a hold the disk live on Discord. Ooh, maybe we should do it on Discord at some point. I would be >> you can stream it to Discord if you wanted to. >> If group site works on Discord, we'll set that up. I got a special guest today you can say hi to a pop-in experience. >> Oh, hey guys.

>> We got Aaron over here. >> What's up, dude? >> All right, today we have some So, welcome to the AI that work show. we have very special special guest today in addition to Aaron, we got Kevin Gregory. and this is a program where what we do is we tell you how to take real AI engineering concepts that have been working in production in the field and apply them to your own projects to build AI that is good enough to ship to production customers. And By Bob, you want to give an intro to what we're talking about today?

>> Yes, but before we do that, let's give quick background of who the heck we are, Dex. >> Okay, fine. >> So, I'm By Bob. >> We're still working on the intro. >> Yeah, we we as you can tell, this is not our official job. We write code most of the time. my name is By Bob. I work on BAML, which is a programming language. Dex. >> I'm

Dex. I work on Human Layer, where we're helping people use AI to solve hard problems in brownfield complex code bases.

>> And Kevin, give a little bit about yourself as well. >> Sure. So, I work at Evolution IQ. I'm an ML engineer. At Evolution IQ, we build claims guidance systems for disability and personal insurance carriers. >> So, you've got you've got solid AI in AI that works live happening in your company, is what I'm hearing. but >> That's right. >> So, today's episode, as many of you know, was scheduled around dynamic UIs. So, we did do a little bit of work on dynamic UIs. And before we get to the really, really exciting part that I that has exciting enough that has caused us to derail the topic today into something even better.

and which is why Kevin is on, so you should really, really get excited for that. We're going to show you a snippet of dynamic UIs. Why is my audio bad? >> Yeah. This is we had a we had a breaking news event. essentially, while you're pulling that up, I will just kind of tell story a little bit, which is we did an episode, actually the very first episode of AI that works, was about how to do large-scale qual- classification and how to build kind of not necessarily the evals themselves, but how to build your pipelines around being capable of leveraging incremental and kind of like very almost like unit testy style evals, so that you can build these complex pipelines while being able to understand like how things work and where stuff went wrong. so, I'm going to share really quickly just a couple of those, and then I'll hand it back to Vibhav to show you what we will be talking about in a very soon upcoming episode.

But, this is the episode from March 31st. We talked about the simple way to do classification, how to pass a schema in, how to add probes to be able to understand and like narrow down your categories with embedding or naive rag or using an LLM. and so, we we did this exercise, and we wrote some code for it, but it was as most of the time we kind of just riffing and doing the concepts. What Kevin has done is actually taken all these concepts and built a actual production-grade system with evals and testing and all kinds of fancy stuff. So, we're going to dive deep on kind of his journey in building that, and then also in in kind of the code itself and and what it looks like and kind of what worked and what didn't. And I'm really excited to learn. I actually intentionally didn't want to know too much about this, so I'm sure I'll have loads and loads of questions and we'll have eyes on the chat, so please drop your questions in the chat as well.

>> You can kind of view it like you can kind of view it as almost like a level three of the concepts that we normally have done and it is just insane what Kevin has managed to do. But I will give you a sneak preview of what of what we meant when we meant dynamic UIs. So I'm going to >> thought real quick I I just got a comment that Five O's volume is a little low so I'm going to turn down my dynamic volume and so hopefully now you can turn it up and you'll get a more balanced mix.

>> how do you do that actually? I'm need a Riverside. How do I set up my dynamic volume? >> if you go click the settings in the top right you can there's like an output volume. >> I don't see my >> slider. >> is it in settings? I I mean oh all settings. Output volume. Mine is maxed out all the way. >> maxed out I will cap mine a little bit so that people can turn it up a little bit and they hear us at the same volume.

>> Okay. Well, hopefully this audio is better. if there are any more issues, please let us know in the chat. So, what is dynamic UIs? Well, I think the idea of dynamic UIs is you should be able to build any sort of content. So like in this case I have a fun little image of like Dexter and I's thing over here or like let me see if I can get a different one. one. So like over here we did cloud code for non-voting task. Yeah, that works number 20.

It'd be really nice if I could build a UI for this element automatically. So here's the two steps that I do under the hood. I ask you ask an LLM to take the same code and describe code back to me in a way that actually Yeah, sorry. >> Yeah, you need to generate a new schema. >> Sorry, yes. I pressed the wrong button. >> any socks or dogs in that image. >> where are we at? So if I take this one I go to this, what I can do is I can actually ask an LLM to go and describe the schema that describes this model.

Once I have a schema that describes this model I can do a bunch of things and I'll just quickly describe what the schema is really fast and we can just burn through it really fast. Which is we have an episode which has title, substring, series info, all this other information. Series info, stream details, schedule, and a bunch of other metadata over here. And then the return type is just an episode. Well, after that I can run this code individually and what should happen in theory is given this image and given that schema, I should be able to pull out really nice structured objects coming out of this.

And you're seeing that I'm doing very dynamic stuff over here where I'm actually rendering I could render this whole thing as JSON, I could render as YAML, I could render it in like pretty format in some ways if that you consider this pretty. I can the last thing that will be showing soon is how to actually also build a React component that renders this. But if any of you have built dynamic React components before, it's actually quite simple as long as you show the React component this is my shape of my object.

Now just turn this into React component and I'm going to give you something of that JSON shape, it will know how to go do that. >> Amazing. >> is kind of dynamic schema stuff we'll walk through. We'll walk through this code in much more detail and talk about how to go do this. this code is open source so we'll send this out. >> Nice. Has mustache true. >> Say again? Has mustache. I do not have a mustache and it is not nearly as glorious as this man's. But I do have the glasses.

>> Anyways, that's a preview of what's coming soon. >> Yeah. And this is what we mean by dynamic UIs. But today's episode is I think infinitely more fun than anything I have to talk about. and then there's a really fun question is like do LLMs know YAML? The answer is yes. they're quite familiar with it at this point. but Kevin, why don't we take it away? Why don't we do a little primer on the classification video like Dexter was walking us through it. Kevin can guide us through a little bit of it in the beginning and then I want to talk about kind of the thought processes that you walked through, and I know you have some stuff to show us.

>> Mhm. >> But, let's describe the problem, and then perhaps show the final result, and then work backwards to how we got there. >> Yeah, that sounds good. So, let me go ahead and share my screen, and I can go through We can do some Scala trough to just help describe the problem. So, the problem is, you know, as as Datsun by Bob said, is large-scale classification. So, what that means is every take if we have, say, a lot of a problem where we

have a lot of classes, you know, in the case that I was looking at, we had, and I'll describe kind of in detail in a little bit, what is these categories are. We have over 1,400 categories. And we get from these 1,400 categories, we have a user query.

So, let's just call that query. Then, so we have a user query, and it goes into these categories, and our system has to select ultimately the best category, right? So, from 1,400 to one, and we'll just call this one Let's see. Call this selected final category. >> And to be to like make this a little more real, this was like a hardware store like thing, right? >> Yeah. Yeah, exactly. So, the example that I looked at, it was we had I can show actually >> Yeah, you want to show some of the categories?

>> Yeah, why don't we just walk through it? I think it's Looking at the data is always the most clear. 1,400 Oh my god, Andrew. >> Yeah. So, we had >> You want to zoom in? >> Zoom in? Sure. >> Perfect. >> Yeah. >> That's good. All right. So, these are the categories, right? It's It basically categories that you would see at a hardware store. So, if you go to like lowes.com or homedepot.com, you'll probably see appliances, refrigerators, and then in there there's like French door refrigerators, right? And so you'll see these are the categories that you would see in an appliance store. And you can imagine this would generalize to, you know, like Amazon or any sort of retail. And a lot of other examples as well, but specifically here we had 1,400 categories.

Right? And so how do we go from a user query of like, you know, ceiling fan, and how do we figure out which is the best category to do that? And, you know, there are a bunch of different ways that you can do this. The most kind of naive way is you just take all 1,400 categories, you dump everything into you can just dump everything into an LLM and say give me the best category. you can also just do straight embeddings where you just, you know, pick the most similar the most similar outcome to the category.

But the problem with doing both of those is you really, when you're doing that, you really only have a couple of levers that you can pull. So when you're just using an LLM, you you only have really the prompt and then the LLM that you're using. And there are a couple of keyword arguments you can use, but in general you don't have that many If if your system isn't performing well, there's not a lot you can do to change it.

with embeddings, really, you know, it's similar. You can change the embedding model that you're using, but there isn't too many there aren't too many knobs you can turn. So the question that they put that Vimov and Dex pose in the first video is how do we create a system with more probes, more breakpoints where you can kind of see the inputs and outputs and things that you can change? And so what I did is >> the naive version down here is like the basically like here's here's 1,400 calorie categories, excuse me, and like LLM pick which one this this query is related to, right?

>> So what I did is, let's see. Do you think it'd be better, Dex, to just create a new one or kind of break this one apart? I think it might be best to break this one apart. >> Yeah, go for it. >> Yeah, so So, what I ended up doing was, okay, so what if we take these 1,400 categories and and we use an embedder and instead of having it give us the best one, we have it give us the top, say, 100. We have it give us the top 100 categories, right?

And, let's see. This is Let's see. This is a vector store. And then, we could dump all 100 of these categories into an LLM and say, you know, pick a final category. But, I took it one step further and I said, okay, so now that we have these 100 categories, we can get an even smaller set, we'll call it 50. Where is the text? Here we go. So, 50 categories. And this one is getting narrowed with an LLM. So, we take the output from the embedder and we say, hey, hey, LLM, like, here's the query, here's 100 categories, give me the best 50.

And then, what we do is we take those 50 and then we put those into the final LLM and say, now's the final category. Select the final category of these top 50, right? And the great thing about this, I'm getting rid of this now, is there's a lot of breakpoints or probes as they were described them in the first video, right? There's That's fine. So, we can see, right, if we have the correct answer somewhere, we can see if the correct answer drops out in this first 100, in this second 50, or in this kind of final answer.

And we can control the different inputs and outputs into these. So, for instance, the >> When you say inputs and outputs, I I I think I think like one thing that's like kind of worth noting is like so the signature of narrow with embeddings, right? Is like query and then you basically just get out a list of categories, right? >> Mhm. >> or I guess it's like a a big list of categories. >> To a smaller list of categories.

>> To a smaller list of categories, right? But there's also like hyperparameters here, like, you know, number of number of like number of number of output categories, right? This is kind of like your you have your like top K and all these things that have like nothing to do with the query or the like content that the LLM is working with, but they kind of control the like the implementation, the like dimensions of your system. >> Yeah.

>> Right. And there's also something that's kind of even more fun to do is you can think about this narrow with LLM, right? This kind of second narrowing. The the signature would be you'd have the query. You'd have the array of categories. So it you would think that it's almost the exact signature as the narrow with embeddings. But one of the things that you can play with is there's no guarantee or there's not guarantee, there's no reason that the categories coming out of here necessarily have to be the categories that go into these 50 categories. You can have an intermediate step if you want.

And that basically says, "Hey, beef up these categories. Turn them from what we saw here of you know, just heating, venting, and cooling, heaters, wall heaters, and create a whole sentence around them or a paragraph or make them much more descriptive." Right? So you can by introducing these new breakpoints, there's a lot of knobs you can turn beyond just 50 categories, 25 categories, or what have you. >> I'm going to pause cuz I think there's a really good question. Akash asked a question, "Isn't this costly because you're just spending a lot of question you're spending a lot of input tokens and output tokens?" But I think what we're really focusing on here, Akash, is actually accuracy. There's some workflows, like for example, if you're a medical billing company and you need to bill ICDC codes. So, ICDC codes are these insurance codes that you use that have examples such as, "This patient had a laceration on the right pinky toe." It's super specific, and that's how they bill for insurance. And >> And there's separate codes for each toe, right? Basically.

>> Literally, yes. Literally, yes. If you There's like 80,000 codes or something. And if you're an insurance

provider, you need to If that code is wrong, you don't pay out. So, the hospital doesn't get paid. So, in some cases, like their entire jobs, whose job it is to take these categories and like go go fill them out. So, the important thing here isn't actually to consider cost. It's like First, we can worry about getting it performant with 100% accuracy or solving for every solution that we have, and then we can worry about reducing cost. And it's similar with latency.

There are ways to reduce latency. There are ways to make this fast. We can talk about that stuff in a bit. But, I think >> We talked about that a lot on the voice agent one, right? >> Yeah, we did talk about that a little bit. >> you How do you use a smart model to supervise the dumb model so the dumb model can move fast? But, yeah, no, I think I think what you're saying is exactly right. Like, there's this spectrum. We talked about this a lot.

There's this spectrum between like cheap and accurate. And what we've said in the past is like, "Cool, if you want to see if AI can solve a problem at all, use the expensive, like cheap in terms of engineering time. Just throw 03 or GPT-5 max thinking at it and see see if AI can even solve it in the first place. And then, once cost and latency becomes an issue, then you can engineer it down and be like, 'Okay, let's make the prompt really good for all the use cases that we know need to work.'"

>> Or we can take this whole pipeline and like get the final input-output pairs and train a fine-tuned model end-to-end that's super tiny, runs on the fly. Like, all that stuff becomes possible, but only if you have it working in the first place. But, there's something about Kevin's work that really inspired me. Cuz like, if you go back, like this is actually more advanced than what we did in our video, a little bit more. cuz he added that extra step in there that we didn't think about doing.

But, what Kevin really shocked me on is actually the UI that he came up with of how to evaluate these results. Cuz it's not actually sufficient to write this code. The real part >> Hell yeah, I haven't I actually haven't seen this yet. I'm super stoked to see this. >> Okay. The real part about these systems is how do you actually think through these systems? >> Mhm. >> So, let's go let's go Why don't we take a look at the final UI that we came up with and why don't we see the UI Let's start with the final cuz I think it's always the best and then we'll show how we got there.

>> Okay. We'll show the end and then yeah, forest from the trees first. >> Exactly. And then remember, for everyone who listens to this problem, then first thing to remember is just like the problem is we have a query that comes in and we have a hardware store that we want to pick the right category based on the query the user put >> Yeah. And I think to to, you know, the the point that they were saying, right?

Like at building this pipeline that and with Cursor and Claude code and even Cur- Cursor's new CLI that they just shipped, right? Like, this is pretty easy to build something like this now, right? This is almost free. Like, anyone here on the call could probably build this within a week or two, right? So, like, building this pipeline is not hard. What's hard is figuring out how to ingest the information in a way that's helpful and tells you what to do to make the pipeline really, really good.

That's kind of where the magic comes in. And so, here's the UI. So, this is a couple of different tabs, but right off the bat, I I came up with 68 test cases from some LLM generated, I reached out to some family members to see what they would look for. And right off the bat, I mean, you can see that overall, it's doing really well, so 14 it got wrong, 54 it got correct, right? But, what's I think even more exciting is when you scroll down here, you can see where in those different stages the correct you know the correct answer fell at.

And so it looks like I mean embedding filtering isn't on here because it made it through the embedding filter every single time. The LLM filtering >> And that's interesting. You took it out of the UI because it literally didn't matter. You're like it this just isn't a problem. So it's the embeddings are correct enough. >> it's still in the UI. It's just it's not showing up because it's not here. >> Okay, got it. >> Yeah. Yeah, yeah. It's just it's it's it's looking for the different types of errors and it's just graphing them.

Yeah. But what's interesting is you can see here like okay, so two of the 14 times the that that LLM filtering stage was where the mistake was. But the last you know the other 12 times it was that final selection was where the mistake was. And I mean there are a couple of things that you can do here. To me the most obvious thing to do is to make the LLM filter more restrictive and see if that final LLM selector performs better with fewer categories.

And so you can see over here I'm not sure how >> I did. Yeah, so this is just version zero, right? With 100 embedding candidates and 50 LLM candidates. And when you look at version one the accuracy gets even better, right? We're now at 13 failures versus I mean 14. But you can see that yeah the LLM filtered out more and the final selection it's is is still the main issue, but it's not making as many mistakes as it was before.

And this was with LLM candidates 25. I mean you can even kind of look even further at this, right? If you look at 15, performance went down a little bit. So maybe something closer to 25 is just you know it's kind of the right thing. >> But Okay, so you're you're tuning the knobs of the I mean I don't want to use the word hyper parameters cuz that's a very specific thing, but you're turning the knobs of the pipeline by building very visual easy to evaluate representations of how different combinations of parameters perform.

>> Exactly. And that's kind of the what's interesting is right out of the gate is you can see like where's the where's the wrong answer happening? Like what's the mistake in the pipeline? Where are the faults? And it's because we've added in all these different break points and created a UI that makes that just kind of jump out at you. All right, so what we're doing is >> jump in real fast. >> Yeah. >> You keep saying the right answer. Was the right answer manually classified?

>> Yeah, so the right answer was I don't know. It's it's funny you say that, right? Yeah, so the right answer was all all me and you know, my family coming up with like what they would want out of a search query. You can you can argue whether or not they're correct, right? Like if if you know, I don't know hardware that way or or that well. I mean you can argue and say like, well maybe I don't maybe when I put something in like the actual right answer to what I'm putting in is something different than I'm expecting, but I think that's kind of more of a philosophical question about like what does right mean in this case?

And I guess that's something that like the PMs would decide, right? >> Well, but yeah, like people that care about the product could be engineers. But yeah, but maybe the PM. >> So So what's what's interesting one of the interesting things is I mean if I don't know if I was looking at this yesterday, I think. And you can go over here to the test case analysis. And you can look at let's look at one that I got wrong. Let's stove with red knobs.

where was that? That was one we were looking at and I just I just passed it. >> Just pull any of them. It's fine. >> Here we go. So and you can see so you can see what it's doing, right? These are all the ones that made it this probably too small. Let's zoom in some. >> Yeah, that's perfect. >> You can see all everything that made it to the embedding filter and you can see the ones that made it to the LLM filter and you can see what it finally selected, right? It selected appliance appliances ranges gas ranges.

>> Oh, this is cool as heck. >> The quote-unquote correct answer was double oven gas ranges and this is where, you know, and if I would actually brought up the point and like, well, this might actually be okay cuz it might be okay to give a more general answer rather than a more specific answer. >> Can right? I if you have more to show here, I I I like I would love to see we can come back to this or you can keep going here.

I would love to take a peek at some of the prompts that you're using at each of these stages or like how the how like how you're embedding the query. Okay. >> Yeah, but I think Kevin's point here is about the general specific was really interesting. We actually got into really interesting like 20-minute side track on this or 10-minute side track on this is really fascinating cuz like what is right or wrong is actually really subjective in a lot of use cases. So like in this scenario for example, imagine I've gone to Amazon and it actually picked double oven gas In this case the answer is double oven gas range but imagine picked that and the answer the ground truth was like gas ranges.

Based on the UI that I had, it might be so easy to go to gas ranges for the user that it might actually be okay to be more specific. On the other hand, the LLM you might argue that I want actually want to take the user to a more general page rather than a specific page for some other reason because of the UI of how my product works the UX of it works. Because maybe if I go to gas ranges, I have a section that shows gas ranges double oven gas range at the top then shows electric gas ranges and everything else right below and it allows me to go see that very easily.

But I think the important part here is like what's good or bad is really subjective based on your use case and even the degree of correctness that you have for an answer in your ground truth set, you should ask yourself like how bad is this? Cuz technically this is a failure for our eval set, but I might argue that this is actually correct. and I might want to reword what it means to be a failure be like, if I got in the right subtree and I'm too specific, that's okay.

If I'm in the right subtree and I'm general by one level, that's okay. We can change the definition of scoring. >> Because you can put in your you like like like the whole point is like you work backwards from what's a good user experience and you can put into your UI like the breadcrumbs of like, okay, here's the narrowest category, here's the parent category, here's and you can let the user just navigate up themselves. And so it's like what's the

risk of getting it wrong and how important is it to get the final category right versus like the 50 categories right?

>> Exactly. And then like Charles brought up a really interesting point, which is in this kind of situation, you can ask LLM to disambiguate with the confidence as well. That's another way to build your UI where you can actually ask to disambiguate it's like if it's ambiguous, go ask for follow-up questions. But you can also build it directly into your UI so you not add an iteration where you just take the user to the right page and you make it easy for them to opt-in to the right direction.

Cuz it's unlikely that they're if they're looking for a stove, they want to end up in like showers or like buying like a rain shower. It that that seems very, very unlikely or they want to buy like a hammer or something or a safety eyewear. which I know some of the categories in there. So as long as we're really, really close and our UX makes it trivial to translate, we can get that's okay. The mistake is forgiven almost in your product.

>> Yeah, it's I think it's like a question of as a user, am I going to be upset if I type in stove with red knobs and I get gas ranges? No, like not at all. Cuz odds are there's going to be an icon for double oven gas ranges right there and I might see the stove with the red knobs in the picture and it's like that's a perfectly acceptable user experience. And actually I have a a drop down over here where we can change the correctness definition. So if we if we allow for more lenient, we can see it it it gets it right. All right, the table doesn't update cuz I I kind of just put this together like just this morning. But you can see it's right. And then we go back to the error analysis, we're at 84% correct.

And I didn't even do any additional prompting kind of get it up. And this this is very bare-bones prompting. >> Yeah. >> And it's like you can basically it's it's more about defining your problem in a way that really understands your language. And I think like what's interesting here is until you showed me this UI, I really wasn't thinking about that problem definition page. But when we were looking at an aggregate, we could actually go and be like, "Okay, let's just look at a failure really quickly and see what's wrong."

>> Yeah. Cuz initially you would think, "Okay, well this is this has all the information I need." Right? I can see in each of those breakpoints where the pipeline is failing and where the mistakes are happening. So why do I need to look at it on a case-by-case basis? And this is exactly why. Cuz when you do that, what actually happened is the pipeline and I disagreed on whether or not we should be too general or too specific. And it's like, "Okay, that's really interesting that we're having this disagreement because either answer could be right just depending on the UI, depending on the product, depending on the user experience that you want." And so that's why it's so useful to be able to look at it on a case-by-case basis. It made it very easy to toggle between ones that are correct and ones that are wrong, right? Makes it very easy.

And what's interesting about what I pointed out, kind of to the point we were discussing earlier, I think as engineers, like one of the things that we like to do is we like to show that we we're being good engineers, we care about latency, right? And I I did this where like I have the latency time here to show how long each of these steps takes. But for what we're looking at now, what we care about the prediction accuracy.

This data does not change I I'm not going to make any different decision with any numbers that are here or whether or not they're here or not. So, they shouldn't be here at all, right? They're not They're kind of just fun facts, but for when we're making a decision using a UI like this, it's not useful at all. It's It's just honestly it's just distracting. Which I when until Vibhav pointed that out, I thought that was a really good point cuz we want to show that like, "Hey, we're being good engineers. We're paying attention to latency." But for the use case of what this UI is trying to do, it it doesn't add anything. It just takes up space that distracts from the actual point.

>> how easy is this to rerun? Like if you change the prompts and can we get like a V6 out of this and like see how it performs? >> We can. Yeah. >> Okay. Cuz I'd be curious >> It takes It takes some time, but >> Okay. Okay. >> can see, it's not It's not very fast, but I mean it it it depending on kind of how many, you know, candidates we're outputting. But if we want to run it and then talk about something else, like we certainly can.

>> One One of my favorite things to do with Vibhav is roast my prompt because he has incredible intuition about like how LLMs like think and it like a per token basis. I think it could be really interesting to like look at the prompts, maybe make a tweak that we think is going to make it better, and then run it again, and then we can kind of like zoom out and maybe like do a couple questions and things like that.

but I'm Yeah, I'd be really curious to kind of see see that done end to end and just kind of like see this thing working. so that people can pull this down and mess with it. >> and we'll check the code into the AI that works for you. Vibhav, you've been working there, right, Kevin? Okay. >> >> We do this. Like we we can play with the prompts a little bit, and then we can kick it off, and then I can show how we got here cuz this is the end state.

There were some intermediate stages that were >> Yeah. >> kind of funny almost. >> Yeah. >> Yes. So, I mean we can do that because I did very little very little prompt engineering, right? Like no. I just have bare-bones prompts. So, if we look at the pick the best category here, I pretty much pulled this straight from the initial video that you all did. So, there are two prompts that it's using. there's pick the best categories, which seems pretty straightforward. And then there's pick the best category, right? This is the LLM filtering, and then this is the final LLM decision.

So, >> You know what we should try? which category is better? Pick the best cat That the second one is failing more, right? >> Yes. >> Why don't we just bump that up to GPT-5 if we can and just run that GPT-5 mini or something instead of GPT-4o. >> Okay. I don't know if I have Let's see. >> I thought that is not what I That is not what I meant by prompt expertise. >> Dude, this is this is the shortcut that I've been learning, actually. You don't actually need it there. You can just change change it right there. You can use the shorthand.

>> Okay. >> So, just change that to GPT-5 and then add GPT-5 mini. So, GPT-5 is good. Yeah, and then instead of OpenAI, do OpenAI-responses. Let's use the new responses API cuz there are slight differences. >> All right, just like that? >> Yeah. and >> Do you want to test it real quick just to make sure that works? >> Can I just do that? >> Yeah, do you have a test thing running on that? >> Yeah.

Oh, yeah. >> If you don't have access I'll send you an API key. Oh, it did not work. What did it say? >> it looks

like it was already running. Maybe you interrupted it. >> Oh, yeah. >> No, it just died. That's the right model? >> Play. Can you run an LLM CLI test really fast for me? This in terminal. We just run it in terminal real fast. And just run LLM CLI test, yeah. >> Just like that?

>> Yeah, that should work. LLM CLI test. I'll just do dash-dash-help. That's odd. live stream fail, I guess. >> You got an old version. >> Oh. >> You do UVX LLM CLI test. >> Yeah. It's got to be I Yeah, just the UVX family CLI test. There you go. >> Oh, wow. >> I Yeah, and then just add in the dash dash from. What dash dash from and then give it the path to >> source. >> That's so it's CLI man.

>> Kyle says this experience is >> so much. I was like, "Oh my god, what is going on?" We actually now output the version number for that reason. That's it. That's perfect. Oh, and you don't have env var set up. >> Oh, you can't stream You can't stream five without verifying your organization. That's what it was. >> Oh. >> Let's just do GPT-4o then instead of 4o mini. >> Yeah, that's fine. Okay, cool. >> Kyle says this experience is making me feel much better. Hey, we're all We're all building and learning stuff together.

>> we do code real time on the fly. I mean, that is what we do over here. We're real coding and make it work. >> That worked. Cool. >> Awesome. >> Okay, so now >> Let's go run this. Want to see if it got better by just doing a small thing like a model upgrade. >> Yeah. >> Cuz if we can do a model upgrade and get it better, then that I think the thing that a lot of people don't think about when they ship is what The most important thing you have to think about is actually not finances.

It's actually not latency. It's can you make it work? You have to find the shortest path to making it work as soon as possible. And then you ship it. You ship an update. You If you're really gated on number of users cuz you're a large enterprise, you gate how much users how whichever users can use it behind a feature flag. But you do not prevent You do not really cuz you need to get real data from production to actually know if it's working. Your sample set is a theoretical sample set. Kevin went one step He got synthetic data. He went one step further. Asked his Asked his friends and family to actually give him sample data. He actually got really representative sample data from like a real audience.

>> But the best he can do is get >> Sorry, finish. >> But the best thing he can do is actually get real sample data from real users. So It's screw cost. Spend like I don't know, spend like 5,000 bucks collecting sample data if you're a large entity. It It's worth it because the representative data set is infinitely more valuable for helping you iterate in real time. And like decide if it actually works. Go do that, collect the data, and then once it comes back like then you can figure out how to go update the model. And like if you think about like from a cost perspective, like engineering time and shipping speed is the most valuable resource you have.

If shipping something If you're going to ship one thing like 2 months later because you wanted to collect data and train a model or like update the prompts ahead of time, you just lost 2 months of shipping on a product. Like that's just not worth it, ever. Just Just go ship the thing that works. And then let's show the UI and stuff while it's running. >> Yeah, so So at first I didn't have a UI at all, right? At first I dumped everything into a JSON. and >> the JSON by hand?

>> I was just reading the JSON by hand. and obviously, you know, like you see this and you're like what What do I do with this? >> How many of you have done this? >> All the categories were here. This is not Like this this wasn't useful. All the information was here, but it just plain wasn't useful. And so, you know, you I needed some sort of UI. And so the question is, okay, so what's a way that we can show this information that's that shows what happens between all these different breakpoints?

And my initial thought, I was looking at the categories in the notebook. I mean, they're they're obviously hierarchical, right? Like we were saying before. So to me, the most natural thing to do when we're looking at something like this is some sort of a tree-based diagram. Cuz this is very naturally like this We probably have five or six high-level categories, and then they're you know, they're the sub-branches. And so my thought was using like a network X you know, visualization and visualizing the tree as all these different things are happening. And so that's that's that's what I did. And if we look over here, yeah, well, I can't load Streamlit, so that'll that'll come back, no problem. So, let me just zoom in. And you can see this this is what I initially did. And this is with just a really small category, only I think 30-some categories, just to get it working. And I only had like 14 things come down from the embedders.

And the LLM filtering was just like one or two or three, I forget, but not many. And so this is, you know, this this looks really good, right? This works really well in this case. You can see the things that got narrowed in the first filter. And then, if you when you would click to the next you know, the LLM filter, you would see all the things that got dropped out and what's still left in the final answer.

>> What are you using to visualize these? Is this just like a Python visualization tool? >> So, this yeah, this was NetworkX. And it was using like I think yeah, Matplotlib and NetworkX. But I think there are a couple of of problems with this, right? Like one is for one like there are way too many colors, right? The the key has one, two, three, four, five, six six categories. The green is very obvious. The orange is pretty obvious.

But when you start getting into like purple and blue, that's it's not obvious immediately what all the colors mean as you start getting more complex, right? Here's another example where it's just like what is just looking at it, you have to look back and forth between the the key, which is not very helpful. It's not the worst thing, but it's not immediately obvious. The bigger problem is that this doesn't scale. This is with let me me 20 categories coming out of the embedder, and then maybe I think four or five coming out from the LLM, right?

If you want to do the 100 categories and 25 categories, it it looks like this. And this is not at all useful. Right? And so this is like what I initially had. And I was like I first was like, "Oh, this looks great." I could even click between it. You can literally see the orange ones become, you know, little circles. Like, that's cool. But, when you start getting into even larger you know, using all the categories and with many more things packed into the filters, it just doesn't scale. Like, this is This is useless. This is almost as bad as the JSON file.

And so >> And it's somewhat worse cuz you can't do command F. >> Exactly. You can't do command F. Exactly. You can't do command F. And you can't like collapse the the the the the sub-pieces of it. So, you know, there's So, and so Vivaldi helped me out with this. He's like, you know, what what would be more helpful with this?

And so, how do we get all this information into a similar amount of real estate?

And that's where the idea of the table came from, right? You have a one column for each filter, and you can sort it, so you can just easily put all the ones that they they made it to the one filter at the top. And then you can color the actual rows themselves. So, that's where that ultimately came from. >> Can you show that UI again just so people can cross-compare? And I think it's it's really >> if this is done Okay, yeah. So, this is done. So, let me I can Let me throw up the UI.

>> Yeah, and I I want to Kevin gave me a little bit too much credit that I helped him out on that. I just told Kevin this UI is That's all I really said. It doesn't look good. It doesn't feel Kevin did something really natural, actually, for anyone Sorry. what I said politely was, "Kevin, this UI is could use some work." >> I think you said, "This isn't the right view." >> Yeah, I said, "This isn't the right view." I was a little bit nicer.

but, Kevin did something really natural when he first approached this problem that I think a lot of people do. Which is they took the problem of like I was like he had 1,400 categories, and he said, "Let me just work with 20 of them." That's actually a really really really good approach. Cuz what Kevin could do in that scenario is without having to think about all this UI stuff, he would actually just work on writing code that worked. Like, that pipe on that we drew out it just took less effort for him to iterate on that initial pipeline in the beginning because that's how you iterate in the very beginning.

>> Cuz we always talk about like the the better your iteration loop is, the faster you can change something and see the result, the faster you will get to the right solution. And we talk about this is the same reason that like you have like unit tests in Babel, right? Is like how do you narrow the problem to just one specific thing and then iterate on that until it's good and then move on to the next small thing.

>> Exactly. So and I thought that was really clever. And then once he did that, he built a UI for that because like the JSON files were useless effectively. then once we built the then once we built that, what we ended up doing were actually took that UI and said, "Now how does this UI scale with all the categories?" And the answer is it didn't. So then Kevin just went back, thought about this problem really hard, and then he came up with this, which is really good. I actually had no input on this. This is all Kevin.

so he's giving like he's giving way too much credit to us on the side, like Show the downstream UI at the very bottom. the the table stuff. Not this stuff. This was really cool personally. >> This? >> No, the other one where you can see the individual test. >> The individual test case. >> Yeah. >> Yeah, I'll go to one that didn't work. Yeah, this one. >> You Exactly. And you can just see very clearly what you what we ended up picking and what we should have picked.

It's super visually clear. >> Yeah, and this is one of the cases, right? Where this is this is more general. And so that's that that sparks these conversations. I don't know if you can zoom in more. But it picked a lot more. >> Yeah. Yeah. >> Okay, so it picked it picked mini split air conditioner {slash} mini split ACs, and the correct one was technically just mini split air. >> All right, the correct one was this the the red one cuz we missed it. So the

correct And you can see it really clearly right up here, right? You can see that the the ground truth was a deeper category, but it picked a higher up category, which again, like you don't think about I might, you know, this might not actually be an error until you look at the table and you see, wait a second, is, you know, if I were a user, would I actually care if I got a, you know, something more general where I would probably just have these four categories that I could click into versus if I got, you know, something really specific and I was trying to get into a sister category or a parent category.

>> Yeah. Eugene Eugene sent something really awesome. Eugene, we'd love to chat with you. I'll find and then see if we can bring you on for an episode, too, for this kind of stuff, this kind of deep dive. but Benjamin said something really fun, which was the parent category and the sub cat sub cat are the same thing. That is the other thing you'll realize when you actually start digging into the data in detail. Sometimes the data's just bad.

But even if you're being totally sane and you build all these data sets, sometimes the actual source of truth is not a source of a truth as you think it is. >> So, wait, you're you're telling me that when Claude code goes in and deletes the test case because the can't get the test to pass, that's sometimes the right answer? >> That's right. It might maybe it's always the right answer. That's how you make your code work. All the tests pass.

Malik asked something really really good and I want I actually wanted to I get your take on this before we show the results of the new thing, which is how hard was this to build? So, let's just put a time about I I'd love a time estimate from you, Kevin, like from from phase one of actually getting the code to work, pure code working, not the UIs or anything. How long did the code take to write? Like the test harness and everything else?

>> So, once I had the code working, >> Let's Yeah, let's How long did it take to get the code working? Like that's just the core pipeline, the BAML function and then dynamic typing. >> Yeah. couple hours. >> Okay. And then how long did it take to get the test harness? >> What do you mean the test harness? >> Like the one where you actually pass in test cases, you dump out JSON files, you do all that stuff.

>> Oh, yeah, that's that's what I meant. Yeah, a couple hours. And then the whole thing was like So, the actual cuz yeah, I mean, I kind of did it I did it together. >> Okay. >> Yeah, so a couple of hours >> most of it. >> Hard coded most of it. Yeah. >> No, cloud coded most of it. >> Oh, cloud coded a lot of it. Yeah, yeah, cloud coded most of it. >> Okay. >> Yeah.

>> and then once you did that, how long did the first version of the UI take you to build? The one with the little nodes and everything else. >> Mhm. probably probably four to five hours, maybe. >> Okay, four to five hours? And then how many hours >> was a little stream was a little iffy. Yeah. >> Yeah, no, it's fine. I think it's good to good to give people an estimate of how long they should be spending on these tasks before they should go think about them. And what about the final task?

Like this final UI, how long did this take? Like you just changed the prompt, you ran the whole thing, it was really freaking fast. How long did this take you to run? >> So, probably was There was we There was a lot of back and forth cuz I I it took me a while to figure out the right UI, but that whole process probably took me

probably closer to like s- five to eight hours to really nail down the UI. Because it's I think part of the part of the challenge is like And I think this is what the the point we're trying to get across is like you don't always know the right thing to build to visualize it until you build the wrong thing and it's like it's not very useful, it's not very helpful. Right? So, there was a lot of kind of things between that and this that that I had to like iterate through in order to get here. But I'd say once I had that kind of tree diagram, like this one, getting to here took probably yeah, another five to eight hours.

>> So, it sounds like what I'm hearing is this took about roughly two full days of work to get from like never having heard the problem to To heard the problem, written the code to having a test harness built that you feel like you would feel comfortable showing like your team and having them iterate on it. >> Yeah. >> Cool. so, that's like the approximate timeline that people should think about is like it's not like Cloud Code and all these other things are great at writing code. I'm guessing like all of this is pretty much AI code gen the whole time.

>> A lot of it. A lot of it. >> Right? And the beautiful part is because most of this UI stuff is throwaway code, you can kind of view it like Jupiter notebooks where like you don't actually have to save them forever. And just like get rid of them along the way once you're done with it and you ship the project. And just make sure the core part of your code, your actual pipeline that you're running, that is good code. Your test harness code doesn't have to be good code because that can be thrown away long-term. It's not actually going to customers. It just needs to be able to be run really quickly, and you need to be able to adapt it with Cloud Code, so it shouldn't be like that messy.

But, it's I would say like the most important thing is just like make sure your core code is good and like Vibe code the rest of it. Vibe code the UIs, Vibe code the Evals, Vibe code the testing harness. But, do Vibe code them. Don't skip those parts. Don't only build the main app part of the application. >> Yeah. One of the the things that Vibe Ops said to me was that like kind of if you can think of it, you can build it now. Like it's basically free to build these things.

So, yeah, the real the real art and engineering comes into figuring out like what gives me the most what gives me the information I care about in the densest way that makes it jump right off the page. >> Yeah. I'm going to show another question actually that came up while we were discussing which was should we Vibe code this in React? Why do we use Streamlit? and this is the mistake a lot of people make. This UI is not important.

Kevin, I asked him a very direct I was originally going to suggest React. I asked Kevin a question of like Kevin, how well do you know He was like, I know Python. I was like, don't think about it. Just use Streamlit. Like it's just like don't ask questions. Don't try and be optimal in parts of the system that don't need to be optimized. Who cares if his UI is perfect? It just needs to work. Does, what are your thoughts on this process? I know you've you're pretty much what I would call the vibe coding expert. I take a lot of advice from you on this. Like, how would you approach this kind of stuff?

>> A great question. Yeah, so I mean a lot of people talk to me and like we're even learning this internally on like how to vibe code things that are a lot like have a tighter iterate iteration loop. And so we're still tuning like the

the process that I'm sure everyone here has heard about plenty at this point is this like research plan implement. it actually doesn't work we're like figuring out alternative flows where we do like plan and then implement and then go back and do the research for things that are more visual, for things that are harder for an AI to validate by writing deterministic tests. Like even with Playwright, it's just kind of hard to vibe out a UI without looking at it.

And so like one of the things we've done been doing a lot lately is like I will go and like not do any planning up front. Maybe do a research that is just like show me the files that matters. But then I'll dump jump straight into back and forth vibe coding and get the thing we're like doing a style change, right? I want to get the thing looking how I want. And the CSS that gets generated is usually trash. It's like got like important like exclamation points all over the place. It's like not how CSS should be written. But I get it looking how I want and then I take pictures of that and then I share that with the team and then one of our expert front end engineers will go back and actually do research plan implement to turn it into like build a production grade implementation of it. So, that's kind of a little bit of a detour, but the idea is like for things where it's like the value is the humans like ability to understand what's happening. I think you should end up doing a lot more like tight loop iteration with the LLM to learn what you want and then if it's a side project like this maybe you keep what you got if it's something that needs to go into production and be production grade code then then maybe you go back and build it from from scratch.

This is if you if you've read there's a book called inspired by a guy named Marty Cagan which is kind of like 20 years ago developed a lot of the like principles of good product management and the idea of product management that isn't a lot of people's heads is kind of like a Jira ticket pusher or like the decider of what gets built and Marty's take is basically like your job as a product manager as someone building a thing and deciding what should be built is to learn as quickly as possible. And so if you can get customer feedback based on a bunch of Figma mockups and people will say like yeah I would pay $100,000 for that that's way better than spending 3 months building it and then showing it to people and finding out nobody wants it. And so like that's kind of zooming out on like tactical advice like that's kind of the core principle is like how can you give yourself the most leverage to be able to understand what you want. And if you're building for yourself then yeah just prototype it vibe code it whatever it is. I think I use vibe I'm not good at Figma so I use I use vibe coding as a substitute for Figma and I think a lot of people I think there was like a Lenny's podcast like 9 months ago where they talk about Amjad from Replit was like and I haven't talked to Amjad for I haven't talked to Amjad ever but 9 months ago at least what he said was like people are using Replit agent as product managers to prototype what they want and show it to customers and understand what people want and then if they decide they want it then that becomes the specification that goes to the engineers and they talk about okay how do we build this in a way that's scalable and production grade and secure and all that stuff.

>> almost think of this UI as something if you're building a Jupyter notebook this is what you would have shown. Now you can show something better than Jupyter notebook. Like this UI does that and I think that is like the mentality that I found is like to go this house. >> And models are much better at writing Streamlit apps than they are writing Jupiter notebooks. I will tell you that. They're They're getting there and like people have done incredible I've written Jupiter notebooks to do Bambu workshops using Claude code, but it doesn't work as well and like you really as most people say like you need to be able to design a testing harness. And so like the

hard part of using a coding agent to write something like a Jupiter notebook is like, okay, how do I build a separate script that actually runs the notebook headless with no output and runs every single cell and make sure there's no errors so that the AI as it's making changes has a feedback loop to like there's no linter for Jupiter notebooks.

The best way to linter Jupiter notebook is actually run it. >> Do you want to show the results, Kevin? from the most recent result. Let's see what happened. I don't I I have no idea. >> Yeah. Let's go to error. Oh, look at that. So before >> Did it get worse? >> No, it got better. So we are at Oops. See, and we were at version Yeah. So we were here since Oh, it didn't change actually.

>> The distribution it didn't really change. So it's not it's like in this scenario using a better Can you change the definition? >> So lenient Oh, we got Okay, so we got up to 84% which is >> And what about if we go to the other lenient? >> I think it's the same. >> No, no, the other lenient. There's another lenient. >> The other lenient. Yeah, where it looks at like more specific as opposed to strict.

Okay, got up to 85%. >> Yeah. And then the other one is >> Yeah, what is V1? Yeah. >> Much similar. Around the same. >> Did you run V1 with GPT-4o? Is that what happened? >> No. Well, I think what it means is like in this problem in this problem the problem is not the problem is clearly not the model. It's like the definition likely of the categories is what needs to be updated to make it work better.

That's almost definitely what I would have to do here. It's actually not about the prompt. I would just go and look at the actual prompt for each category and be like, "Okay, for these cat And like if we go look at them, can we go look at some of the error cases in V5?" >> I'll just pick some failures, yeah. >> And just go dig into this. And I think people can get a grasp of how like we're thinking about these problems. Like let's pick this. So the user said desk shelves.

Honestly, I'd probably be like this passes. Like it's sub-optimal, but I'm like it it it it looks pretty good. >> Yeah. >> And maybe they didn't want decorating shelving, but they probably want shelving. >> Mhm. >> And that's probably what I would pick it as. I think the ground truth is wrong. Let's look at the next one. >> Yeah. >> Let's just go through and like look at some There's only 10 cases. Cool. That one passes. We're good.

>> Mhm. Oh, this is This Yeah, this was a one of the really interesting ones. >> Or gasket seal? Actually, it's correct. The ground truth is wrong, right? I guess these are parts. >> Yeah. >> which one is it? I actually don't know. let's go on to the next one cuz it's like it's very close on that one. >> Induction stove. Oh, interesting. Induction ranges like that's fine. Right? >> So I think I think what I really took away from this like we've looked at a bunch of these errors. It honestly looks like there's just like two categories for each of these as multiple categories that are actually correct answers.

So what I would actually probably do is I'd probably go back to my ground truth to some and make it an array. Like actually before updating the prompt and say like, "Any of these ground truths are fine." as long as I pick one of them. And I likely will in my UI make it a another thing where in my UI I'd actually say, "Here's the

selection I made, but you might have also meant this." And show them a totally separate category that's related. So I'd both make a UI change and an eval change over here.

So, exactly like Andrew said, what we do is if you go back to your code, I would literally just take this and the pick best category would just become a category array. I'd say pick the top three categories here that make sense. Or like and then I'd be like and just have the user go select that and then I'm done. And I in my UI I'd pick one as the primary. And for the secondary, I would go ahead and just pop it into my UI somewhere in the final application that says, here's the breadcrumb layout, but you might have also meant this one instead.

Because it's like sampling through these is like I let's go on. Let's see if we see any that are actually wrong. >> Interesting. Okay. >> Like that's okay. It's it's two parts for wine coolers. >> Yeah. >> -huh. >> And then appliance parts and beverage coolers. You're like the fluorescent bulbs, S shelves. >> And I think this is kind of what I meant by sometimes the data is wrong. If you just took this as a raw metric and you didn't dig in, you'd be like, oh, this system's only 80% correct. But if we actually looked at 100% of the failure cases, all of them are actually not failures. It's a problem specification problem much more so than a failure.

Like my data is just polluted with bad data. Where >> It's it's it was created by people who don't really know much about hardware stores. >> Yeah, or like the >> It might only have like the categories have so much overlap. >> Exactly. And like it's going to be wrong in that scenario. There's nothing that they can do about this. but this was really really fun, Kevin. Thank you for going into this really deep with us. this code will be checked into the repo. We'll send a summary of the video out. I know we missed last week's summary. You guys will get a summary, I promise.

for last week. We've just had a very very busy week and we'll announce shortly why. But questions for anyone. Feel free to pop them into the chat and or raise your hand or something and we can bring you on. >> Yeah, this is cool. I'm going to share real quick. Nick shared an old article. Let's see. Let's get a tab. >> Kevin, you want to stop screen sharing? There we go. >> Yeah, there I got it. So, this is this people have been trying to figure out research opportunities in e-commerce search for a very long time. This is an older paper from 2020.

but it's just a single You can go find a hundred other papers on this topic. And so, there's This is as far as I'm concerned, these are like not toy problems. This is a real thing that a lot of people are trying to figure out. >> These problems are worth millions and millions of dollars. like hundreds of millions of dollars. >> And yes, Andrew, we can apply this to categorizing architecture for how you do tool calling. We actually did an episode about this of like, if you have a thousand MCP tools and like, this is coming back to the coding agents thing.

I think Jeff Huntley did a talk recently where he I mean, lots of people have highlighted this. If you add the GitHub MCP server to your Claude code and you don't turn off any of the tools, it's 60,000 tokens of just tool definitions. And that's going to tank your performance compared to like And a tool call is the same as a classification in in many points. We're asking the model to output structured data about what type of thing is this, whether it's like, what type of thing is this query about or what is the right next step in the workflow based on what the user is asking for. So, the same concepts there of like, how do you narrow And there's actually

there's a post on the very top of Hacker News right now of a team who built an MCP server to allow you to kind of like inject a thousand tools or thousands of tools into your AI and that does the narrowing on the MCP side for you because that's the only way to do it.

>> doing the same thing Kevin did here. We they just use some embedding or some LLM filter to like go from a thousand to 50 to a hundred. This code is really easy to go write. if you guys like this kind of content, this is AI that works. If you want to tune in next time, we're going to do another fun episode. You can check out the link here, and we'll get back to answering questions in a bit. But like this is like one of my favorite things to do every week. it's very fun compared to the regular work that we get to do every day.

and it's just a nice break for us. Mike asked a really good question of, "What do you think of ML flow? Is it dead since by putting UI is so easy?" Personally, I think I mean, I've totally stake the horse in this. I think we need a new programming language to go solve some of these iteration loop problems. like there's just a totally different workflow that people have now that has never been true before. You need to iterate and code at the same time.

And like you wouldn't use some you wouldn't use like you wouldn't use a bunch of HTML code to do what React can do. And you probably shouldn't use a bunch of random smashed together ginger not ginger Jupiter notebooks to like build a prod production app. And like I think like things like ML flow long term are not just not going to be as good. what do you do eval? How do you eval? How do you do evals on candidate screening chat?

Something that is conversational not zero or one range based evals. so like how do you do dynamic evals? We did an episode on evals a while ago episode five that talks about some of this stuff, but I think we should go deeper into this. But Dex, Kevin, how would you guys think about this? Like let's say we have a chatbot and we want to build an eval system for it. What would be your first gut? >> If I wasn't going to vibe code the UI, >> Well, you you might vibe code the UI, but how would you define the problem with eval? How do you define success, ground truth, or something along that array?

>> I mean, I'm just going to say what we said in the we did that episode a while ago that was like policy to prompts, and it was like how do you take, you know, a terabyte of not not terabyte. It was like several gigs of like raw email content, and how do you like evaluate that against certain compliance rules to decide if the emails were breaking certain rules of like what is ethical corporate behavior or whatever it is and it's like start as small as possible pick a really narrow thing and pick a really narrow slice of the data and don't grow the problem or the slice. You have two dimensions you can play with, right? And I think you kind of want to grow them in parallel. Like okay, now we're going to do two problems and run it across the 100 emails and then you're like okay, those are working. Now let's do 1,000 emails and let's try and you just keep iterating as small as possible so that like you want to start in a way cuz the first like going from 40% accuracy to 80 or 90% accuracy, you can probably get most of that from a really small data set and you want to be able to go look at the data. If you start by dumping in 10,000 emails, one, your iteration loop is going to be way slower and two, you are going to not want to go look at the data. So frame your problem in a way where you can vibe eval it either by creating a UI on top of it or just by looking what do what Kevin did was like start by looking at the JSON and then when you

have so many cases that you can't read the JSON anymore, then make it easier to read and easier to debug and easier to work. Why not just iterate and code using cursor instead of baml?

>> before we go to that one. Kevin, what are your thoughts? How would you approach the chat problem? Like if there's a chatbot you want to build an eval for? >> Yeah, I mean that's a that's a really tough one for me. I think I really like everything that Deck said. I think the first thing that I would do is is is think about it on a kind of a a problem by problem basis. It's like what is the cuz it's that's a really big problem. It's just cuz like what is is is this chat good? Is this is this what we wanted this to look like, right?

That's that's almost an impossible question to answer and so I would try to break it down into like really individual questions. It's like kind of like to Deck's point, right? Like is this chat following this rule? And then I would you know build test cases and kind of vibe code and build my UI and then I would keep, you know, just keep adding more roles. And then you know, adding more chats and kind of try like Dex was saying, kind of trying to doing the horizontal and vertical. and see where that got me.

>> One thing I think that I have found successful in these kinds of domains for like really, really ambiguous problems is like instead of thinking about like hey, I just screw the AI part, and then let's just think about a let's just think about a world where we're doing a call we're running a call center. How do call centers run? You have people taking the calls. You have people that randomly spot check their calls. And then you kind of go up the chain and it just goes up the chain forever, basically, until you basically get this. But what they really do to make all of this work is actually tie all of this to actually deal with to deal with end of end of line business metrics.

It's like how much how many returns are we getting? How many orders are we getting through? How many sales calls are we getting through? It's really easy to tie it to revenue at that point. And so what you can do is you just you're just like hey, we have a median revenue number that we're running across the whole company. This group is performing worse. This group is performing better. And what you do is you just spot check and you just say is there can I see something that makes this spot check worse? So before you actually build evals, just spot check. Spot check, spot check, spot check. And but tie it to an end business metric.

Like putting a chatbot on a page is freaking useless unless you can tie it to an end business metric of some kind because you won't actually be able to converge on anything meaningful. And then once you can do that, you can spot check, make a change, and see if the metric goes up. And you can do it localize. You can say I will do this metric for this kind of user persona. I have a chatbot on the website, and if a Python user comes up, I expect their Python code to be I expect the code sample that spins up by default to be Python. If a Rust user comes up, I expect their code that generates to be Rust.

That's an easy thing to go build a system for my chatbot to make it work, and I can build like memory or whatever I need to go solve this problem. But, the end business metric I might be is the number of times people copy and paste code directly from the site. If they copy and paste code directly from the site, that gives me a key metric on how good it's how good it is. If they copy and paste code on the first generation of the site, on the first generation of code versus the fifth generation code, that tells me how long it takes to get to good code snippets.

So, it's almost like I can I can I can build a pseudo metric of something and even for most problems. But, I think you just have to go and think about how do you tie this to the end result that you're trying to drive in some way and then like you can build any sort of UI UX you you want around that, but you should build UI UX and you should look at chat logs and spot check.

I think there's one last question is like why not iterate using cursor to code code deterministic code instead of vibe code the entire solution and use like probabilistic models from Wagdin. And I think this question is really interesting. I think Eugene in the chat was saying great point. I was like, "Hey, we use deterministic code to solve some of this classification problem." I suspect maybe a decision tree or something of that kind or something else along that road.

And that's not a bad solution, but there's certain problems that are just hard. Like, for example, in the past world when we used to go and say if there was an object in an image, we used to do a lot of computer vision math to go figure that out. Now, I literally just toss this at an LM and say, "What is this?" and it'll tell me what this is. It is really nice that I can just go do that and don't have to train a model to like train it on like charger blocks.

But, the trade-off is it may get this wrong. So, I'm living in this world where I'm getting much more capable systems at the cost of accuracy. And it's kind of like when I use Python for the very very first time in my in my life and I trade off performance to get really fast ways to ship code. Like, I could have written C++ code and managed memory myself or I can write Python and just ship it.

And the trade-off there that you should think about is like people used to be like, "Oh, why not just manage the memory yourself?" Well, it turns out as computers got better, we didn't need to. It was okay to use for like this app remember size probably using a 70 gigs of RAM or something stupid on this on this machine while we're running this, and that's okay. None of us care. The internet systems are good enough. the computers are good enough. That's not the problem of today.

So, I think it's the same with LLMs. Where like you should use them sometimes because they're really good calculators. And they're they're a little probabilistic, but sometimes that's okay for the kinds of problems that they have. And like Eugene said, it's just like they're just problems that are better for LLMs. and there are problems that are bad for LLMs, and you got to understand them. But, there are problems that are good for them, and you should use them.

I think that's it for questions today. but again, always a pleasure having everyone on here. Thank you guys for joining us on this Tuesday morning as per usual. if you guys want to come check it out in the future, sign up here, and we'll put on a new episode pretty shortly, and likely we'll do dynamic UIs coming in soon for this one and show this off. But, Kevin, huge huge props from from me and I'm certain Kevin >> Dude, this was a blast. Thank you so much for sharing this and for hustling on this. And like I know I know we do a lot of coding prep for these episodes most of the time.

I have I have called myself out when we underprepared, but I know how much work goes into stuff like this.

And I know you've been having fun hacking, but I don't want to like diminish that like you put in a ton of work to like hack on this and find something that works. So, I hope you learned a lot, and it was super fun having you on, and I would encourage everybody else to get hands-on and try some of this stuff and get in touch and and tell us how it went.

>> Yeah. >> Yeah. Well, thanks for having me. This was a blast. >> All right, everyone. thank you guys soon. We're going to log off and get back to work. >> Catch y'all. See you.