Optimizing the R in RAG: Hybrid Search Is Just The Beginning

54 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=QWFHNK6MVU8
https://www.youtube.com/watch?v=QwFHNk6Mvu8

SUMMARY

Doug discusses hybrid search, which combines lexical and vector search techniques to improve search results. He emphasizes the importance of understanding both methods and how they can complement each other, particularly in the context of using large language models (LLMs) for query and content understanding.

- *Hybrid search merges lexical (string matching) and vector (embedding-based) search to enhance retrieval accuracy.*
- *Lexical search relies on techniques like tokenization, stemming, and synonym management, while vector search uses embeddings to capture semantic similarity.*
- *Effective hybrid search requires careful candidate selection and filtering to ensure quality results, particularly when using embeddings.*
- *Query understanding is crucial; LLMs can significantly improve how queries are interpreted and matched with relevant content.*
- *Pre-filtering candidates before applying vector similarity is essential to avoid diluting the quality of results.*
- *The relationship between pre-filtering and hybrid search should be strategically managed to optimize retrieval performance.*
- *Tuning and hyperparameter optimization are necessary to balance different scoring methods (e.g., BM25 and cosine similarity) effectively.*
- *The choice of vector database should align with the team's expertise and the specific needs of the search application.*

going to talk about hybrid search today, which we'll define what that is in a little bit, but yeah, I'm Doug. you may know me at any place where internet accounts can be had at software Doug. you can find me as software Doug on LinkedIn, on Blue Sky, TwitterX, the web, worldwide web, Pelaton. find find me on Pelaton if you want to do rides with me, start a Pelaton group. and a lot of people may know my blog or know me from having worked at Open Source Connections or just through the years talking about search in lots of different aspects since the early 2010s. I've been doing some version of search. Ironically, one of the very first search projects I did was a vector search project with John Barryman where before people really had too much of a name for what an embedding was, we used something called latent semantic indexing to do some matrix factorization of some text to try to see if we could use that to sort of like generate synonyms and that kind of thing. And this was like pre-wordve, pretty old school stuff.

we had to use Hadoop and and it was a much more painful experience than it would be today. The world has come a long ways. but before we get started, just like obligatory plugs, Hamel has his class AI evals for engineers.

It's a great class for just learning how to actually like approach your application and build it from a system systematic way of measurement. and I'm doing a class in July for cheating at search with LLMs, which is kind of like, can we get 80% of the way there without having to think too hard just with some LLM query and content understanding, taking apart queries with LLMs? I feel like a lot of the search stack is going to be is being rethought right now around LLMs when it comes to query and content understanding.

now that it's become even more e it's gradually becoming easier and easier to and cheaper to either host an LLM yourself or to or to like call open AAI and all the other providers are having for their have lots of small model options that are increasingly cheap and sort of like pushing down the price over time. So a lot of cool stuff there to play with. So that's all coming. I really encourage you to come and see if my assertions about whether we can how far we can cheat at search without having to break out something like pietorch how far we can push that what I I'm going to talk about hybrid search which is the combination of these two search worlds lexical which is pure string matching and vector which is embeddings more like statistical similarity of of text. what I'm not going to talk about because I feel like most people probably have a sense for these things.

but what these thing how something like lexical search actually works. Lexical search is the OG search that has based on thinking about how we tokenize text so that strings match really well with things like stemming or limitization. People use synonyms. people build their own crazy semantic search using taxonomies. A long time ago, when people talk about semantic search, they usually meant taxonomies and managed vocabularies. They didn't mean embeddings maybe five or 10 years ago.

And of course, that's changed. but all of that's built on string matching. And if you're interested in how all that works, like the core of lexical search and techniques around making lexical search better and getting more out of BM25 kinds of scoring techniques, definitely check out relevant search. Also, not going to talk about like what is an embedding. if you know an embedding, I feel like there's you probably heard a million times what an embedding is. It's it's just a array of floatingoint numbers where when the arrays or vectors maybe 256 to 1024 to higher dimensions when they're close together that means whatever they're representing be it a passage of text or an image are semantically similar.

and if they're farther apart they don't have anything to do with each other. And there's a whole other world and set of books and things that have been written and need to be written about just embeddings in general. But these are the two universes we're talking about. I'm assuming a little bit you're coming with that those that information. because I feel like we've been we've all been inudated with like the vector database folks and everything at this point of like what an embedding is and how to use it.

The other thing I won't cover which a lot of people start with are like sort of naive strategies of interle two search systems. So a traditional thing that comes up if you're starting with an old school search system that's relatively mature and you just want to layer in vector search is to use something called reciprocal rank fusion. I have a blog article called RRF is not enough. But the reason I'm not talking about it, I mean, I think there's a lot of some people, you know, you may get value out of it. It's useful when both both if you're it's basically RF is a way of combining two completely different systems, two completely set different set of search results into a

single set of search results.

if you are interested in in that definitely it's not necessarily useless but it's only useful if vector and lexical search provide results of equal but different quality. If your hybrid search if your lexical search is terrible and you're combining it with a pretty good vector search you're just going to drag down the vector search and vice versa. so it's typically it's maybe something to kick the tires of, but I I don't think it's that particularly interesting or necessarily where we want to be as an end state.

It's kind of a stop off for a lot of teams. my and then I'm going to talk about a bunch of assumptions that go into how we think about building a hybrid search ranking function. The other thing I'm not going to do necessarily, what you should do is systematically go through I have a blog article that does this and I really encourage you to check it out. I'm not going to systematically go through and be like I measured this approach and the NDCG or some metrics prove that it's it's good. I have done that and I if you go to my blog and look for like the elastic search hybrid search blog posts, you can check that out where I have a sandbox of trying a lot of these different techniques to to just kick the tires, try different things out. and and a lot of the approaches here have proven effective. but I do want to give you an intuition for my assumptions on why building a ranking function like I'm going to talk about is an effective way to do it. So my first assumption is typically embeddings are a good first pass of similarity. So you can see here I have these two maybe document titles. Mary had a little lamb. Little Red Riding Hood had a baby sheep. And they don't really share maybe they share a little but they don't really share important terms that like a lexical similarity would catch but they do share you can see the vectors are just eyeballing them. You see they're pretty close to each other. So it's kind of a sense of we are in a general sense without thinking much about what the query is or how we should serve this query. it gives us in the ballpark semantically of what should be their search results. It's sort of like a good first pass set of candidates. So that's my first assumption. My second assumption, this applies especially to rag folks, but it's not only a rag thing. in search it's underappreciated. How what is the document in my search engine I'm representing? I'm assuming that you've taken the time to chunk this into something that's a semantically meaningful chunk. a long time ago, I worked on an a project for O'Reilly Media that was about of one of the incarnations of the Safari online book project. And the biggest the biggest thing that improved search was not and this was pure lexical search but the biggest thing that improved search was how we subdivided the books into sections in a semantically meaningful way and so things that were kind of roughly article length in a way that you could peruse through them. it's really important. So how how your similarity systems play with and how you're chunking the the content up is really important. and your data may just be naturally represented like you have a product with a description and an image and these and a name and that kind of thing. But it's an important data modeling thing to think about with search.

the other assumption that I think is important when you're doing hybrid search, and this is starting to get to like hybrid search techniques, we often think about an embedding as a thing for a piece of our content. in my mind, and this is this is not always the case. We have, but in my mind, we want to evolve gradually to having as much as possible an embedding for the thing being searched. That's our first pass embedding. We could have like other embeddings. Maybe we do some kind of reranking and stuff, but we need a first a good first pass retrieval embedding.

that sort of represents the thing being searched. And you can see here as a naive example, making sure I'm getting for this product data set the name and the description in the embedding. But there could be other metadata that we want in there. and the other thing is the embedding is sort of it's not necessar people may know the concept of a two tower embedding but at a minimum that the embedding is sort of like two towerable in a sense that if you give it a query like a short bit of text and you give it a document that they will relate to each other that they'll be seen as similar even though the query is relatively short and the document is longer. and that you can put both types of text into the embedding and it'll work. This if if for example one thing to think about let's say you have a an embedding model maybe you have an embedding model that's like extremely super targeted to like medical users. like it's really good at it's like taking apart medical documents and all this stuff. It can really speak that jargon. If you have lay users like people who don't know medicine coming in to search, is their vocabulary going to map to the the documents vocabulary? If you want a really cool thing to search for in the realm of information retrieval, it's like a old old problem.

It's called the vocabulary mismatch problem. And it's basically like if I hold something up, there's there's research. If I hold this up and ask people to name it, there's a lot of variety in like people's first word that they will name this thing. people will say mug, some people will say cup, some people will say glass, some people will know a lot more about like like the manufacturer of this thing and what it is. And that's something that search is like constantly working against both when we're developing embeddings and when we're building our our using embeddings themselves and even better because I talk about embeddings as a great first pass retrieval system everyone it's great if we can move to a as much as possible you may have heard like by encoder models all these things but A notionally a two tower model is specifically a model where we take query features, document features and to solve that vocabulary problem, we learn the similarity space.

Maybe we start with some embeddings in each of these attributes, but we learn the similarity space. won't get into that here, but that's like a bonus that will that is something you want to evolve that like first layer of embedding retrieval to get to a a good even better and incrementally get better at this at this process. So the other thing after we do this embedding, we've retrieved some embeddings, that's really where we want to think about our query understanding and be like, "Oh, we've got a good set of of candidates here.

Let me move up things like the query mentions color. Let me move up the color matches. Let me make sure I prioritize the name matches because the user I understand the user search for something by name." That kind of thing. and I definitely recommend Daniel Tung Tunan Lang's work on query understanding. That's like his whole thing. Query understanding is going to be a big topic at my course, cheating, cheat with LLMs. I feel like LLMs are basically going to this this used to be the kind of thing where like to understand if a color was mentioned in a query would be a, you know, monthsl long project or something.

Now it's like, you know, hacking it together in a week or something. so ideal some kind of query understanding first pass embeddings and then some later layer of boosting reranking reranking could be a model reranking could be manually tuned all of those are totally valid. the reality is we can't pull back. If we did this for every result in our index, we would be processing millions and billions of results out of our search engine. we would just see everything ranked in the engine from simo similar to least similar. And that's not something that's

realistic. The realistic thing is we get maybe 100 embeddings. And out of those 100 embeddings, we have to do that like reranking and making sure that we have enough candidates that are like of things like exact name match or color when the user mentions that it to actually boost them or rerank them or treat it to treat them in some specific way for that query. yeah. So do we have the right 100 to boost? So to do that is say we like know we want to get the the right we want to boost we know this is a from a query understanding we know this is like a product name query they're searching for a specific brand of something of you know sneakers or of like you know a book by name or something then we need to make sure when we get the top 100 embeddings that like those exact name matches are like represented. So there's kind of a chicken and egg problem here, right?

It's like I want to promote those things out of the initial candidates, but to do that, I need to make sure those initial those things I want to promote are in the initial candidates. So tricky thing to get right and that's really a lot of hybrid search problems really sink or swim on whether or not you get the right mix of candidates in your first pass retrieval so that later re-ranking stages knowing more about the query can actually do something intelligent with them.

So let's start to get into like our how we would think about this from a retrieval standpoint. Our 2021 or whatever vector database is like literally this. It's like select documents from search engine. We just rank by vector similarity getting the top 100, right? we have a query embedding and like a title embedding or whatever we have and or document embedding and we're just getting the top 100 and what they would say is if you wanted to filter which I would talk about maybe you post filter and then you get some more and you post filter you get some more that's all like mishuga it's like no not very exciting 2025 vector databases have realized this is a problem that to get good to solve this chicken and egg problem to get like the candidates we need. Let's say a user searching a garden like search engine for trrows.

Trial is a type of shovel. to get the tri matches to rank to like see the tri matches that are like exact product name matches, I have to filter down to them and then rank them by vector similarity. I need a capability to select for those candidates to explicitly say I want these types of candidates in my mix. So in this case getting a set of candidates that's like where trial is in product name order by vector similarity. So now, and this is something that's really important, if you have a vector database or search engine, you should really be asking and digging into how exactly does filtering work for before I even for my vector retrieval so that I can get a good mix of candidates up to higher layers of of boosting and re-ranking. This is a huge topic and you will see people talk about, oh, we do filters, but actually they're post filters. are just like reducing the result set which is less than ideal. What you really want is a good prefilter filtering while searching and you want to understand if especially if you're at high scale what are the performance characteristics of this and I've been posting on blue sky about this like just talking about doing some interesting looking at some interesting databases from cudrant does an interesting way of building up their vector index to do filtering pine cone has a really cool way. Turbo Puffer has a really cool way, but it's not like baked into the like original like research of things like if you know like how vector databases work, the HNSW data structure or or any of the other it's not something that's usually baked into that. It's something added later that people realize they needed.

So definitely make sure you go and figure figure out how your vector database does this. You can you can do

this. This is absolutely needed to do hybrid search. And of course, when we get down to this level and you start thinking about what are all the things I would want to filter on, well, there are some things that are like maybe mandatory filters that you're just going to have like the right department and like that you're searching the right item type. And then there are like making sure I'm getting the right set of query terms in the mix. Okay. And so this wear could really represent anything. It needs to represent any kind of tokenized text, any filters, cate like boring filters, old school filters like getting in the right category department, you know, whatever you think is whatever your data has represented fairly well. And I'm at this stage, yeah, we're completely again, we're completely ranking by vector similarity. We're ignoring BM25 for now.

yeah, as I said, there's practically a top K. Now, it's kind of boring if we have vector search and I like force you to only get these candidates of like, oh, you I have vector search, but now all my vector match matches are always going to be keyword matches. It's like, well, what was the point of building a vector database if like you're forcing me to do that? Like, what what the hell? like but luckily the reality is we have lots of different candidate sets that we might want to express for our later rankings. So or this top one that I've shrunk down we get 100 from that set that are like keyword matching candidates and then maybe 100 from this set where we're just we're not enforcing the keyword match constraint. we're getting 100 from from a different set of results that just like the mandatory maybe it has to be in the department or whatever. And this is where the rubber hits the road on your design, your domain. How is your data represented? Do you have what characteristics how are users querying? What characteristics are of attributes of your documents are actually on the items themselves? Do you need to use something like an LLM to like enhance your documents and queries to get at that and then improve your filters so that you can build better hybrid search? So with lots of squiggly lines, you might have candidate set A, this is lexically filtered, then maybe candidate set B pure vector. Then we pull these candidates up and then there's a boosting, reranking, whatever phase.

yeah, and we of course we don't want to just get this candidate today because those are like pure life matches. there's a huge benefit to just having to also just having vector matches like the the things that don't directly match the keywords the user search for but have the same meaning. Yeah. And there's trade-offs here like of course pure vector is more recall and the lexically filtered to the keywords is going to be maybe more precise and and this is where and you'll hear these terms brought up on search teams. If we have L0 retrieval that's like usually our first pass. It's good to a lot of a big pattern you'll see in information retrieval are like layers.

So you start with like higher recall and you refine as you go down. and you can dial knobs at each layer to be like, oh, I'm going to like increase recall at the base and then at the next layer L1 I'm going to like improve precision or do reanking a different way and so on. so L0 retrieval can set a filter to lexical can be b true vector. call these you often hear these talked about as arms and sometimes arms are complete systems built for this. So sometimes people have like a retrieval arm that's like a set of candidates from like a cache or something or what's historically worked well for this query or or you know like I have here lexical but ranked by vector those kinds of things and yeah there can be many things you can imagine like maybe we have a a candidate arm that's like one term matches maybe all terms match we could even even though I have like ordered by vector sin here. Maybe we have a set of candidates that's actually a bunch of vector or lexical results.

and then we maybe have just like the same just the category or maybe there's like a direct phrase match. Maybe we have a set of candidates that's like based on some really deep understanding of like someone built out a really great taxonomy and managed vocabulary like this equates to that using some kind of synonyms or something like that. So all of those are totally valid different kinds of retrieval arms. and then later we pull this up into higher level reranking. but a lot of this is depending on the query. So we switch things on or off depending on the query.

So we may have some queries where it's like we really want to focus it on we know that the color is important. We really want to focus on the color. We've parsed out a color from the query. We're gonna enable turn on the filter for the query so that we're pulling up those candidates and bring that to subsequent stages and so on. And then finally we have a boost maybe here is a pretty naive score plus equals let's take the product name in index for all the L0 matches and if there's a garden trial maybe a phrase match on garden trial. This is like made up Python pseudo code. we increase the score somehow. and those get pushed to the top or a model like at this stage we may have like cross encoder learning to rank all the cool stuff people all the cool things people are doing these days at a bit higher. But the core the main thing I want you to take away from this is that the core bit of functionality that you have to really focus on when you're doing hybrid search is this filter to in sim in vector to into a vector similarity system pulling out the top most similar vector results filter to this and that is like at scale that is the operation you really need to care about when you're doing this work.

so that is the slides. I do have a notebook and I'll even paste it to Can I paste this to the chat? People can go nuts. Well, it's not letting me copy paste, but I'll do it later. So, we have a hybrid search notebook. This notebook actually does two things. It has a has a built-in lexical index using search array, which is a just a dumb thing for prototyping. full text search that kind of feels like a numpy array.

and this is all based on some basic Python PI data stuff. and also just straight up brute force numpy for vector matches. So numpy is just the python array library and if you do like a dotproduct for example you can get the similarity of two vectors. It's it's that simple and you can do things like filter down to the only the vectors that are matching the lexical matches and that kind of thing.

so I'm going to walk I'm not going to execute this because the embedding takes a while, but I just want to walk through it relatively quickly, but we more or less we're just doing a mini LM dumb way if you want to prototype. We're creating a I'm using a data set called wands which is a wayfar's furniture open furniture search data set. great. I really appreciate if anyone on the way is on the call on the call, good on you for releasing this data set just just loading it to a Python data frame and then concing it.

I create an embedding that's just this is just a numpy array of every embedding just with the name kind of d-description. And then as a lexical index, I've built an index for description and name using snowball tokenizer, which if you're not familiar with lexical search, this is just a way of taking English to root forms. So if someone searches for M A R Y or M A R Y or something like that, it all gets sort of stemmed down to the same representation. A lot of lexical search is like playing with tokenization so that terms map to a kind of equivalent space.

but we get a tokenized view of the product description, a tokenized view of the name. And what we can do when we have this is we can actually get we'll take a query like leather love seat and we can get a score. In this case, we're summing this is the name. So we get a for leather we get a BM25 score for love seat we get a BM25 score and we sum them we get a score for each document and we really just see like whether there's a match by checking we're doing greater than zero same thing for description same exact code and then if we pass a list score here by the way is a search array function if we pass a list of query terms it is a phrase search so we'll get like phrase matches direct like if you actually search for leather's love seat that leather love seat the phrase pops up and so as an example here are the phrase matches for leather love seat in the data set same this is just any mention of leather or love seat not phrase and if you play with this notebook you'll see that and then of course cosine similarities so cosine similarity this is Under the hood, there's lots of different similarities. This is a really straightforward one to implement.

that's just the dotproduct. If the nor if the vectors are normalized, we don't do this, but it's the dot productduct of the two vectors, and we get the most similar. We get the ones that have the highest, the the rows with the highest similarity that goes from negative 1 to one. if we take our query again this is a leather love seat and we get the similar to the embeddings and we'll get top 10 you can see here and then this might be our L0 we talked about where we are pulling up match a pure vector search arm of results with a product name leather square arm love seat leather love seat all these things are different kinds of leather love seat which is good. Some of them directly say leather love seat like as a phrase. other ones are it's kind of square arm love seat. So we get like this is 10 vector search results perfectly fine but we want like filter vector search results. So this is the step where in real life you need to figure out how your vector at at scale like you can actually use numpy and dumb things that you know relatively large like hundreds of thousands of vectors and it actually works pretty good. It's a brute force vector similarity. But what we're doing here is we're taking the embedding and we're saying where are their phrase matches?

Let me get down to a filtered view of the embeddings. within this filtered view of the embeddings we search and we get the top 10 out of that and this is just mapping back to the indices in the original data frame. So don't stress too much about that plumbing. But if we get the most similar where there is a phrase match you can see here the similarity. Whoops. Don't have the similarity. But the most similar ones are these guys. Obviously, these all have a phrase match. And what's interesting is some of these are actually like relatively high similarity and other ones because of the description or whatever, they're not necessarily as high of a similarity in our in our embedding. and we can get lots of arms. We can get lots of candidate arms.

We got like just name matches, just description term matches, description phrase matches, name phrase matches. And this is the same plumbing code above. And just building up a set of indexes into the into the documents and scores. And you can see there's a lot of variety. They tend to be higher, but some things are like 0.57 cosine similarity. So whereas if we entirely relied on the vector which is the first one we'd only get these candidates. we can combine the arms and you can see we got a set of documents and a set of similarity scores and they're not sorted. So you see here there's like 0.57. At one point I thought there was a point 4.

we can sort them and you can see like this is actually the entire index. Oh no it's not the entire index. Oh, I think

this code just didn't execute. But you can see here we got our top results. And then we might Okay, so that's gets us however many like what is this 40 or so results and I should add when there is something from two arms, we just take the maximum which of course the similarity here is the same so it doesn't matter. So let's think about our L1 part of our ranking.

so L1 we have a we're going to get the BM25 score from the name index of doing a phrase search. This is exactly the same code before except we're not doing greater than zero. We're just literally giving a list of query terms to search array score function gives us an array of BM25 scores which are mostly zero. Some of these are not. we see what are matches to this phrase phrase and that are matches in the original and we end up adding them right here. We add the phrase score and of course this is an interesting point because adding a BM25 score to a cosine similarity these are like completely different on completely different scales. so that requires a bit of tuning. And finally, you can see, oh, we've boosted, we've pulled up a bunch of candidates of different kinds of leather love seats, based on different lexical criteria. And we've also said, oh, well, let's make sure if there's anything that directly matches the name leather love seat, we pull this to the top. And if we're really smart, what we would do is have an LLM in the loop to say like leather love seat. Do I want to pull out like an item type like love seat? Do I want to pull out a material like leather? Do I want to like explicitly have certain types of like filters and things I'm pulling out of that user's query. and then similarly we would do the same on the on the documents themselves be like how could I extract like maybe there is no material in this but if I look at this document could I extract a material and actually add a material field for this document so that these filters which were so far just done on description and name matches could be done on other things that are relevant to the query like the like material or color or anything else that matters to the user.

So, so with that, we're just about at time. I don't know if people want to Yeah. Can I ask you a question? Yeah, go for it. Yeah. so how do you think about like the relationship between, you know, like pre-filtering, hybrid search, post filtering? How you decide, you know, what mix to do, how how to do it? What's like some general guidelines? Yeah, post filtering is always like a safety net. So if you're like removing things from results, you want to not reser remove a lot of them. So it's like, oh, I need to remove something that's it's hard for whatever reason to remove from like ideally you would have removed it from the first set, but for whatever reason it's not possible.

So you you've pushed through a lot of results. You're pulling things up and maybe there is some thing that occurs 5% of the time and you want to eliminate eliminate that. maybe it's like maybe an example would be like the rare case in a social media or not rare, maybe it's not rare, but like the case in a social media site where something's like offensive or whatever. you're like, I don't want to like think about this in my core retrieval all the time and like build out these expensive filters because if I do it earlier, it's going to be more expensive. and the earlier stuff is like stuff that's just like I guess like in some ways hotter or like very always going to be like active cuz you're going to spend CPU on that every time and it's going to if you didn't have it there and you did it later, you would end up like getting a bunch of results and then like completely removing them. You'd have like start with 100 and get down to like five a after you filtered it out. And that's something you want to avoid.

And what about like pre-filtering that versus hybrid search? You know, when it comes to like how do you trade

those off? Yeah, I always think about always I always think about hybrid search as like the pre-filtering is a thing that enables the hybrid search. but sometimes to sometimes it just has to do with whether or not you want to think about sometimes things come out and it's like do we want to filter or do we want to boost this?

That's another thing. It's like is there an attribute that I want to enforce as a hard filter to make sure it's either in the results or not in the results or is there do I want to take that and based on the query or situation or the type of attribute is do I just want to promote that higher? an example of that is like if if for some reason like you wanted, I don't know, you're searching for something and you're searching for like if you're searching for a book by an author, you definitely probably want to filter on that. If you're searching for a book, but maybe it's in a genre or something that could be fuzzier, maybe you start to like that's become starts to become a ranking criteria. But you also want to make sure it's a filter. You want to make sure one of those arms you want to make sure it's specifically selected for as a as like one of your candidate sets that like I'm making sure like I have a mixture of science fiction and stuff because I explicitly queried for that if that makes sense. a lot of people are wondering and it might be a good time for you to talk about this like what are you going to discuss what are you going to discuss in your course like you're doing a course that like dive deep into search.

yeah. Yeah. So, getting deeper definitely getting deeper into this stuff more hands-on with this stuff. and then also the actual process of query understanding with LMS with a real data set like what sorts of attributes should we pulling be pulling out and then what kinds of way how are we lading up the representation of documents themselves to those it's like dimensions of intent I like to call it.

So are you what are the how are people searching and some of that's not necessarily expression of the query some of it's like user based and even in things that you think are like completely pure like knowledge based information retrieval there are there are like subdivisions of use cases out there so at at Reddit for example is something like a trending news topic or something like is Is this a product name that you want to get reviews on? Is this a is this a like some cool like meme thing? And if you break out those things and within each of those, how you search them would be completely different. And now we have LLMs that really help us with this class both the query sort of routing problem and the and query understanding. So within like the within the news article maybe there's like what are the specific people being talked about? What are the specific types of like entities or businesses being talked about or within if there's a if we're detecting a product review case, what is the product name?

what sub like what subreddits do we know are like associate with that? And there is a tremendous untapped potential there because I feel like in the past for traditional search teams there's almost like a a feeling in the gut of like oh that sounded like it's going to be hard. But I I think what I'm trying to sort of go over maybe discover with with the with the attendees is like how hard is that anymore? It seems like we should be able to do that and where's the boundaries of of what's possible with LLM in real time like processing live search queries. So very cool. Yeah, I look forward to it. I signed up last night. Oh, good. I appreciate that. Yeah. So, you're the here sign up for D's course. Yeah, definitely. I I recommend people doing that.

one question that came up that got some up votes is when you were discussing in the notebook about combining

the BM25 and the and the cosign similarity, you mentioned something about tuning it, were you were you like scaling the BM25 to be on the same scale as cosign similarity or were you doing something else? Can you talk about it real quick? I wasn't doing anything in that notebook. I was just adding BM25. there's a strategy that I talk about in relevant search that comes up when you're doing a manual ranking function called tiering. And what that is is like you have your for your first pass you kind of have a set of scores that are like in a range like I don't know negative 1:1 and then maybe when you're you're if you want to boost on something like BM25 and it works out well in this case but you kind of want to like have like a stepwise these get like 10x the score or something like that. So it's almost like yeah they're they're cosign similarity plus BM25 and it does turn out that BM25 is like 10x cosine similarity just by happen stance and then everything else is like a is like at that cosine similarity layer. but yeah, I didn't when you do this kind of work, that's always like the art in science is like playing with these magic numbers to find the right thing. And when it gets so complicated, that's really where it becomes you want it to be like a machine learning problem. but that's its own set of tradeoffs.

Makes sense. So in the simple case, maybe you just do do some you do some tuning like some hyperparameter search in a way like against the D. Yeah. Yeah. that hyperparameter search is underrated a lot of at Shopify we got by for a long time with just doing basian optimization of our hyperparameters of this theme I just like you can imagine a in our case it was a elastic search query that followed a similar it spirit practice and it was like how much should we boost this how many candidates should we pull back but these like we kind of like generally have a sense of how what works well but to like at the end we would say okay well let's put on our hyperparameter optimization and sort of like what what are good parameters of this ranking function we feel like we've done a decent job crafting and understand makes sense okay I can pull some questions from the chat some other one. Yeah, sure.

Markx asks, "If you don't have a golden data set, how do you evalidate how do you evaluate your candidate selection strategy?" yeah. well, hopefully you get one. I have a whole thing that's on NDCG is overrated. If you go to my blog, you can go look at that. But it's all about alternates to to ND like traditional evaluation. There's someone posted LLM as a judge is a whole thing and that's also something we'll talk about in my course is using LLM to evaluate search results.

the tricky thing I find the best way to do LLM as a judge is to at least get some basic golden set down because you you want to be you don't want to like blindly trust the one as a judge. You kind of like a lot of times you start out like 80% there which is useful but then like that last trying to get to that like 90 95% where you can really trust it is a whole other level. but like without without that, I feel like here's this here is a flow that works and has worked in the last couple of teams I've been on. Regardless of golden set, a lot of times there's a golden set, but what I will sit before we're about to launch something, we will have some kind of at Shopify, we call it a sense check.

at some places it's just a bug bash, but literally at at Reddit we called it a Pepsi Coke challenge, but we literally just like got side by sides of here's my thing I want to do. Here's the old thing. And it's tricky because like to do this really well, that's why I call it Pepsi Coke challenge because it's like when they do Pepsi Coke challenge, they don't tell you which is Pepsi and which is Coke. If you sit on a bug bash and you're like this is the new

thing, go team, we want to release it.

everyone likes the new thing. but you literally get side by sides of like dozens of queries. So if you if you have your new approach and you know the population of queries that have changed. So let's say you have thousands of queries in in some set and you don't even have to have labels for them, but you know out of those thousands, these couple hundred have changed. let me sample out of those couple hundred that have had significant change and socialize them around to the team and just get people's gut check on them.

Now, there's a lot of flaws with this. There's like gut checking from your teammates if you're like they have nothing to do with your user population. but and I and like for trade-offs and like signal to noise ratio of like sometimes getting a really good golden set is kind of like people really invest heavily in it and it may not be worth it at whatever stage you're you're at in your progression as a search product. I love it. looking at data comes up again. Yeah, exactly. You really have to look at it. You can't just be like, "Oh, my NDCG went up.001." You really have to be like, "What changed?" And the other thing I'll say is when you have a golden set and you most people see the average change in their metric, they don't look at the tails. So they don't like they don't think about the thing they the use case they completely obliterated and destroyed versus the thing that got better. And if there's like tremendous variance in that, that is way riskier because your golden sets often rarely super unless you're Google or something, not like representative of everything. There's way it's way riskier than like the thing that really worked well and targeted the five queries that represent your problem and didn't change anything else, but NDCG like barely moved. Like that's I'd rather have the five queries change and ndcg barely move than like have a big ndcg change and like huge variance in like every query without like doing tons of checking and like having lots of checks and balances. Makes sense. I actually I really want to ask you so this is the last question.

so I think the number of people doing retrieval has like 10xed in the last two years because of LLMs but like you know retrieval has been along been around for like you know a while and you have been working in retrieval for a while. What do you what are like the biggest mistakes you think people are making with rag in the retrieval they're doing for Rag, you know, that that you've noticed or that drive you crazy?

well, Rag has a couple of things that are help you get away with stuff that you wouldn't necessarily get away with in a traditional search. So traditional search, you issue a search and you want results back in like tens, hundred milliseconds or something. Of course, like with an LLM, you're like are chatting with a chatbot and you're singing the words scrolling the screen. So you get away with a lot of stuff. I think a lot of what people do is well first I'll say there are a million things that can work like people are like you can manually create a ranking function you can like you can build a model you and a lot of it depends more on your team's competencies one mistake is like maybe caring like too much about like especially most people have like are dealing with millions of documents and at like some scale billions of documents it starts to matter but at most people's scale to say something controversial like I'm not sure it matters what vector database search engine you're using as long as they can do like pre-filtering and like basic search and these kinds of these kinds of things.

The other thing is like to that point a lot of people got into the like I put that slide of like vector database 2021

versus now it's like over time pe almost every vector database has gradually evolved into a search engine and that's the one area I might be like if you're thinking about this stuff like make sure that your make sure that you are your vector database is along far enough along that curve that can do keyword search it can filter reasonably prefilter at a reasonable speed. and it has the bells and whistles that you sort of take for granted in like a search engine or really a relational database.

that you want to make sure that your thing has it. And I think a lot of them are much better and farther along. But yeah, I'd say like people get like too cute about like the like, oh, what vector database are we going to choose or like all this stuff. I'd say a lot of things can potentially work. One thing I'm really happy about is we got through this whole talk on rag and we didn't ask you what your favorite vector database is and so I'm not planning on asking you that. So I I always say like go with whatever your team is like comfortable with. Like that to me is the most important thing.

Like at at Daydream we have a team that's like very comfortable with elastic search and so that we went with that. But like other places are comfortable with other things or like maybe got started in a direction they really know that technology and have a good relationship there. It's like there are some people love post PG PG vector. It's like there are a million ways to skin cat, right? That was good. Okay, so we'll email everyone the link to this recording. Sometimes it takes 48 hours.

And also a link to Doug's course and other information that he went over, even the collab notebook. And yeah, thank you for coming and look forward toward your course, Doug. Thanks. Yeah, thanks. Thanks, everyone.