EPISODE 708

[INTRODUCTION]

[00:00:00] JM: Google Search allows humans to find an access information across the web. A human enters an unstructured query into the search box. The search engines provides several links as a result and the human clicks on one of those links. That link brings up a webpage, which is a set of unstructured data. Humans can read and understand news articles, videos and Wikipedia pages. Humans can understand the results of a search engine.

Google Search solves the problem of organizing and distributing all of the unstructured data across the web for humans to consume. Diffbot is a company with a goal of solving a related, but distinctly different problem. How to derive structure from the unstructured web? How to understand relationships within that structure and how to allow machines to utilize those relationships between different entities across the unstructured web through APIs?

Mike Tung is the founder of Diffbot. He joins the show to talk about the last decade that he has spent building artificial intelligence applications from his research at Stanford, to a mature, widely used product in Diffbot.

I have built a few applications using the Diffbot APIs. I encourage anyone who is a tinkerer or a prototype builder to play around with it. Diffbot is an API for accessing webpages as structured data. It's an API for a knowledge graph. Diffbot is really cool and I recommend checking it out. Diffbot crawls the entire web. It parses websites. It uses NLP and NLU to comprehend those pages and it uses probabilistic estimations to draw relationships between the entities on those pages. It's ambitious product. It's an ambitious set of ideas and Mike has been working on it for a long time. I really enjoyed our conversation.

I want to mention, we recently launched a new podcast, Fintech Daily. Fintech Daily is about payments and cryptocurrencies and trading and the intersection between finance and technology. You can find it on fintechdaily.co or on Apple or Google podcasts. We're looking for other hosts who want to participate. If you're interested in becoming a host for Fintech Daily,

send us an email, host@fintechdaily.co, and I hope you enjoy Fintech Daily and I also hope you enjoy this episode.

[SPONSOR MESSAGE]

[00:02:44] JM: Managed cloud services save developers time and effort. Why would you build your own logging platform, or CMS, or authentication service yourself when a managed tool or API can solve the problem for you? But how do you find the right services to integrate? How do you learn to stitch them together? How do you manage credentials within your teams or your products? Manifold makes your life easier by providing a single workflow to organize your services, connect your integrations and share them with your team.

You can discover the best services for your projects in the Manifold marketplace or bring your own and manage them all in one dashboard. With services covering authentication, messaging, monitoring, CMS and more, Manifold will keep you on the cutting edge so you can focus on building your project rather than focusing on problems that have already been solved.

I'm a fan of Manifold because it pushes the developer to a higher level of abstraction, which I think can be really productive for allowing you to build and leverage your creativity faster. Once you have the services that you need, you can deliver your configuration to any environment. You can deploy on any cloud, and Manifold is completely free to use. If you head over to manifold.co/sedaily, you will get a coupon code for $10, which you can use to try out any service on the Manifold marketplace.

Thanks to Manifold for being a sponsor of Software Engineering Daily, and check out manifold.co/sedaily. Get your $10 credit, shop around. Look for cool services that you can use in your next product or project. There is a lot of stuff there and $10 can take you a long way to trying a lot of different services. Go to manifold.co/sedaily and shop around for tools to be creative.

Thanks again to Manifold.

[INTERVIEW]

[00:04:59] JM: Mike Tung, you are the founder of Diffbot. Welcome to Software Engineering Daily.

[00:05:02] MT: Thanks for having me, Jeff.

[00:05:04] JM: Diffbot is a set of tools and APIs that are built around in Knowledge Graph that you have constructed within the company, and this Knowledge Graph is the result of taking unstructured data on the web and putting structure around this data. Let's start off by discussing the difference between unstructured and structured data. Can you define those two terms?

[00:05:28] MT: Sure. Unstructured data are things like documents, webpages. They are things that are created primarily for human beings to read. Humans read and understand webpages and make sense out of them. They can convert it to meaning or structure. Structured data is essentially a data that's think of like a relational database. It's data that broken in two different fields where each field has a certain type and a value. For example, a person might be an entity that has a name. They have a job title. They might have an employer, a street address.

Actually, we refer to this as knowledge. We make a distinction between just raw data, which is just a sequence of symbols, and knowledge, which is something that's more meaningful. It's more structured. It's more semantic, if I can use that world. That's basically like a statement about the entities in the world.

[00:06:27] JM: As an example, if you took a document like a news article, then you could process that article and provide structure to it. Give an example of some kind of structure that you might add to a news article.

[00:06:44] MT: Yeah. Actually, most of the information on the web starts out as structured. Most webpages these days are dynamic webpages. They're not handwritten HTML anymore that you might write in an editor or using Microsoft FrontPage. They're documents that are generated from some underlying database. So the data is sort of born structured. Then people use something like PhP, or ASP, or a favorite front-edge language to add a bit of styling a little bit of

positioning using HTML and CSS and basically convert it into a document for human consumption.

What Diffbot essentially does is it kind of reverse engineers that process. It starts out with a document and gets back into what is that structured database representation, and we do this using artificial intelligence. We do this by truly reading and understanding the page like a person does, because that's what it's meant for and converting it back into structure and then return this back to the developer as a JSON object, which is essentially just a set of keys and values. Yeah, does that make sense?

[00:07:55] JM: It does. As an example, the way that I have used Diffbot is as an API for getting structured data about URLs. You have some other features that will get into later, but this API specifically where I can send an API request to Diffbot with a URL for wikipedia.com/stevejobs, and Diffbot can take that URL, it can look at the webpage and it can extract information from that webpage and send it back to me as a JSON object and it can send information, like a collection. It will be a collection of fields as you said. This like, "What is the title of this page?" "What is the featured image of this page?" "What is this page about?" "What kind of page is this?" "Is it an article? Is it a product that is being purchased?"

If I put in a Wikipedia article, like wikipedia.com/stevejobs, what is Diffbot going to return to me in that API response?

[00:08:54] MT: You really hit the nail on the head. If you're familiar with the traditional way of web scraping or manually creating a scrapper, what it involves is basically using either something like Scrapy, which is something in Python, or whether it's scraping tools written in JavaScript or it's using one of these visual tools, like an import.io or like Kapow. You essentially rules for each site using things like CSS and XPath selectors and regular expressions, and that's okay if you want to gather information from just that one site and create your own sort of homespun crawler. But if you're doing at any substantial scale, like across thousands of sites or the entire web, it's not really feasible even to create and maintain rules for all these different sites. If the webpage redesigns or changes their layout, then the rules you've created will break. This happens on average 15% of the time each week for any sort of rule set you have to maintain.

How Diffbot is different is you don't have to write any rules. You just pass in the URL and Diffbot handles all the rest. How do we do that? The first thing we do is we take the URL that's passed to us through the automatic API, just one of our products, and we basically render it in our own web browser. It's in our data center and it's basically running in a virtual machine in that web browser. The first thing we determine is what kind of page is it? Like you alluded to, it might be an article page. It might be a product page. It might be a person page. It might be an image page. It might be an organization or a company page. It might be a place page.

We found that there are 20 or so different top level entity or page types that comprise 98% of the surface area. So Diffbot can, first of all, determine what kind of pages. Is it the homepage nav page or one of these what we call lead pages? Let's say it is an article page, then what we do is we run computer vision and natural language processing algorithms on the page. We look at all the pixels on the page. What are its colors and fonts? How things are positioned? Just like a person does, we'd see everything that is visible from the perspective of what's the rendered document look like. We see everything that's visible from the renderer itself, like the actual internal states of the rendering engine, like the layouts, the DOM, the HTTP conversation that's going on. We make a determination of what type of pages, and then we extract the properties or fields of this type. If it was an article page, we would return back the article's title, its author, its publication day, the clean text of the article, the clean HTML formatted version of the article, the images and captions in the article as well as the entities that are mentioned in that article, the publication country, any comments to the article.

It's essentially a nearly perfect representation of what's in a database that generated the article, and then the developers can now utilize that information inside their apps or their business processes. An example of applications that use the article API are Instapaper. So it's for reading article later. Snapchat, which has the discover stories feature and like DuckDuckGo, which has like these article files in their search engine.

[00:12:12] JM: So if I am building an application where I would want to save articles for reading later, for example. I use Pocket. I think that's kind of like Instapaper. For all I know, Pocket uses Diffbot. Instapaper, for example, if I'm saving an article for later, why would I want to use Diffbot in this case? Why would I want to use a service to just get an API call that's going to give me the

structured information rather than just scraping that page myself? Why is it so much easier by doing it through an API request?

[00:12:50] MT: Yeah. Before Instapaper adopted Diffbot, they had basically a homegrown system that had different rules for each news site out there. If your sites that you wanted to save an article from wasn't in that set, then you would have like a very subpar experience. They actually crowd source the creation of these rules. Their users could actually go in and edit these results when they broke. It was like a huge maintenance effort. You think about a typical article page, there are a whole bunch of things that you don't want to include inside your read it later experience. You don't want all the ads that are surrounding that article inside that article view. You don't want the popup and the cookie front. Sometimes articles are multipage, so you'd have to click the next button through all 10 pages to actually get the full article.

There are many of these encumbrances having to maintain and write all these scraping rules, right? With Diffbot, since it uses machine learning and does it automatically and it's able to recognize these visual patterns very robustly with a high degree of accuracy. It's over 97% precision on this. It's much easier. You don't have to do that anymore. You just ask any URL. You can get back the response and you save it in your database and you can use it for later.

[00:14:05] JM: When I send a URL to Diffbot, maybe it's wikipedia.com/stevejobs, and you receive that API request. Are you going to the Wikipedia page on the fly or are you hitting some cached version of the URL that you've gotten from crawling the web and saving the page and saving the structured information about the page?

[00:14:31] MT: When you use the article API or what we call the analyze API, you're generally requesting the page on the demand. It's not a cached version generally, because unless it was recently requested like very recently, that's because the articles do change and products do change the prices. You want the most up to date information.

[00:14:52] JM: Right. You're mentioning products. That's another utility for this URL API. So I could put in URL for amazon.com/toiletpaper, or bounty toilet paper, whatever, and Diffbot is going to give the information about the price and all these other kinds of things. So I can build applications on top of that as well.

Before we go a little bit deeper, can you give any other applications that people are building with these APIs and how these APIs are used more broadly?

[00:15:26] MT: Yeah. We have quite a diverse customer base. So if you're interested in hearing sort of like how we built this out overtime, we have generally people building consumer applications and people building enterprise applications, right? In the consumer space, I already mentioned individuals apps, like Instapaper, Snapchat, Shop Spring that utilize like our article data, our product data, our people data, our organization data.

We have major consumer search engines, like Microsoft Bing, Yandex, DuckDuckGo, ecommerce search engines, like Amazon, eBay, Walmart. I'll also include in the search engines like these voice assistance as well that require lots of structured information and knowledge. On the enterprise side, we have people that use it for all these different kinds of functions you might imagine that knowledge workers need information for.

For sales, you can query our databases and get sales leads for recruiting. You can help find talents for marketing uses. You can build a clearer picture of your customers, understand content for content marketing. If you're a large brand, like say you're a Nike. You can monitor all the places on the web that sell Nike shoes and determine if anyone's breaking the NSRP or how well that product is received. With structured data, it allows you to analyze these much more easily.

Business intelligence and kind of Wall Street applications. So you might want some of the structured data to build quantitative trading algorithms. You might want the news data on organizations and entities. If you're an analyst trying to track particular sector, as well as government. So just trying to improve national security.

[00:17:07] JM: Indeed. The reason I'm asking for these different use cases is because when I first came across Diffbot, I was like, "This is so flexible," and just the general idea of – I use Google Search all the time, and I put it kind of a search query and I'm looking for a bunch of additional information about that entity that I'm searching for. It almost feels like Diffbot is that for programmatic information in some ways.

It's such a flexible way of building on the web and building on the structured data that can be derived from the unstructured web. I mean, much as when I'm searching for information on the Google Search engine, I am inferring structure about those articles that I'm reading, but that has not been turned into an API, at least as far as I know from Google. So I know Google has some kind of knowledge graph thing, but I haven't used it or I don't know as much about it as I do about Diffbot. Anyway, I really like what you have built with Diffbot and I think it's very flexible. I'm actually surprised that I don't hear more about applications being built with it, but maybe that's just because it's kind of like a backend sort of tool.

[00:18:30] MT: Yeah. Some people have described Diffbot as kind of like a Google for machines, where Google is more of a search engine for humans. We crawl the web white Google, but we go a step deeper, where Google is more like a card catalogue, where when you type in your query, returns back basically a set of results which are pointers to pages to read. Ultimately, you as a human being, as you're doing research, still have to read those pages to read and understand them and get information.

We're taking the step deeper of actually analyzing and reading those documents and converting them into structure and then unifying that information into a knowledge graph so that it can be actually actionable inside intelligence applications. Whereas Google is more of sort of routing your attention to these different places on the web and ranking things that you should read and injecting some advertising along the way.

[SPONSOR MESSAGE]

[00:19:31] JM: We are running an experiment to find out if Software Engineering Dailydaily listeners are above average engineers. At triplebyte.com/sedaily, you can take a quiz to help us gather data. I took the quiz and it covered a wide range of topics; general programming ability, a little security, a little system design. It was a nice short test to measure how my practical engineering skills have changed since I started this podcast. I will admit that though I've gotten better at talking about software engineering, I have definitely gotten worse at actually writing code and doing software engineering myself.

But if you want to check out that quiz yourself, you can help us gather data and take that quiz at triplebyte.com/sedaily. We have been running this experiment for a few weeks and I'm happy to report that Software Engineering Daily listeners are absolutely crushing it so far. Triplebyte has told me that everyone who has taken the test on average is three times more likely to be in their top bracket of quiz course.

If you're looking for a job, Triplebyte is a great place to start your search. It fast tracks you at hundreds of top tech companies. Triplebyte takes engineers seriously and does not waste their time, which is what I try to do with Software Engineering Daily myself, and I recommend checking out triplebyte.com/sedaily. That's T-R-I-P-L-E-B-Y-T-E.com/sedaily. Triplebyte, byte as in 8 bits.

Thanks to Triplebyte for being a sponsor of Software Engineering Daily. We appreciate it.

[INTERVIEW CONTINUED]

[00:21:29] JM: The API that we discussed, the information extraction API where I put an URL and you extra structured data from it, that's one feature of your product offerings. The Knowledge Graph that you just mentioned is this other one where this is where you're actually building a database. You're taking your knowledge from crawling the internet and parsing it. I guess you're using the same APIs that you do for API extraction, but you're actually crawling the comprehensive web. You're extracting that structured data and then you're building relationships between the structured data on those webpages. Tell me more about how your knowledge graph works.

[00:22:17] MT: Yeah. The mission of our company is to build the first comprehensive database of all human knowledge. That's sort of like our North Star. But from day one, we didn't have the resources to crawl the full web. It's kind of a pretty – It's a pretty large endeavor, right? There's only two other U.S. entities that do it, which is Google and Bing.

So that's why we first refined and created a very highly accurate extractor, because that doesn't require building a crawler in this large infrastructure to visit the whole web. The business model for that is essentially developers, "Hey, as you go, similar to Twilio, we get like a fraction of a

penny every time someone sends us an API call." So we're sort of getting paid and we're analyzing just what the URL the customer sent us, right?

[00:23:03] JM: And you can save the data, right? Every time they –

[00:23:05] MT: Yeah. Then we take the structured data. We cache it so that we can use it to retrain our models. You can imagine over the course of operating the service, we just hone and hone and got it better than human levels of accuracy and processing about a billion URLs per month. Over the course of 15 months, we processed about 50 billion URLs, which is about the size of the useful web. With just only a little bit of incremental crawling, we could get full coverage of the web, which is what we've done. So you can imagine applying this automatic classification and extraction algorithm on every page that's on the surface of the web. Then what you do, you have all these extractions, right? Sort of not only that.

What we do then is we look at these extractions and we say, "Which of these extractions are about the same intensity?" Let's say, like an iPhone 7 might be sold on the Apple Store. It might also be sold at BestBuy. It might be sold on walmart.com. Those all are referring to the same product, right? That's just different pages about it.

Similarly, a person might have, let's say, like a LinkedIn profile, some profile on their school webpage, some social networking profile, like we mentioned on a blog or in the news, but those are all about the same person entity We use automatic machine learning techniques, much like a person does, and we try to resolve which of these are the same real-world person, place or thing, then we're able to cluster those together.

Once we do, then we know, "Okay. All these extracted information is about the same real-world concept." It's about the same person or the same product, the same organization. Then we take all those extracted facts and we fuse them together and we calculate the probability of truth of all those facts and then we sort of build an entity, which is a single JSON object out of all these. Then we inject these highly confident facts into the knowledge graph.

So the knowledge graph represents basically our view of all the entities in the universe. The way you use it as a developer is different. So instead of having a pass in a particular URL and then

waiting for us to analyze it on demand or instead of passing in a domain and waiting for us to crawl to all domain, you just issue what's called a Diffbot query language, DQL query to the knowledge graph and then it answers that much like a structured search and returns back their results in less than a second.

[00:25:27] JM: I want to comment on that business model. I think it's pretty creative. The idea that because it's expensive to crawl the web for this kind of structured data, you amortized it by standing up an extraction API and then selling that extraction API so that anybody that wants to query a webpage and get structured data about it, they pay you a little bit and you get the information from that extraction. It takes a while, but eventually you get to build up a knowledge graph and you get to do it actually at a profit.

[00:26:09] MT: Another side benefit of that is we got really good at extraction, because people wouldn't pay us money if it wasn't actually better. Whatever processing we're using before or using human curation or human gathering of data. It's sort of a forcing function that forced us to make the technology really good and really robust, because it's not like other scraping tools or you create rules per site. Literally, someone can pass us a URL from anywhere on the web. It could be written in any language, in Japanese, German, some languages we've never seen and we have to extract it perfectly, right? It's that environment that enabled us to develop a technology and just make it really solid, make it industrial grade that all these large companies can use it, does performance and highly accurate.

[00:26:55] JM: While we're on the business side of things, this company is 10-years-old. You started Diffbot 10 years ago, and I think you've only raised 13 million, which is a lot, but it's kind of a small amount for a company that's 10-years-old, and I believe your profitable. Just kind of interesting, it seems antithetical to some ways of structuring their businesses in Silicon Valley, I'd love to know more about kind of the story behind your monetization and your reasoning around fundraising and kind of the business side of things.

[00:27:32] MT: I think we definitely go against the typical Silicon Valley trend. I've been working on AI since the third grade and I've been fascinated by this field ever since I can remember. I was a PhD student at Stanford AI Lab. So I've been working on this on my own for like a long time. We didn't formally incorporate the company until like 2011 or so.

[00:27:52] JM: Okay.

[00:27:53] MT: But the early period of the company basically me sitting alone in a dark room working out the math and getting the algorithms to be very accurate, and it took a while to get it to a human level of accuracy. That's the point sort of when it became commercially viable and when companies would pay to use it.

We launched on Hacker News, that first developer API, and immediately a bunch of sort of Silicon Valley based developers that worked in companies thought it was interesting and started kind of prototyping with it and using it and some of the applications for their company.

In terms of the funding aspect, so it's really like after launching that that I figured, "Hey, we need to actually build a company around this. Start buying more machines to operate this thing and hiring people to help." I wrote initially probably the first version of most of the systems that use and then hired much smarter people that have come along and did things much better.

The first round of angel investment we got was from Andy Bechtolsheim. He was the first investor in Google and he really saw the vision of it and just kind of led our seed round. Then we recently raised a series A led by some of the early backers of SpaceX and Tesla. It's on both of Elon Musk's board, as well as Tencent, which is one of the largest internet companies, and also Sky Dayton, who is the founder of EarthLink.

Our kind of investors are a little bit nontraditional too. They're all operators, founders that have built tech companies and scaled them. They are not your typical Silicon Valley kind of Sand Hill Road money manager type of person. They're people that I just felt complemented my skillset, understood what I was trying to do. Had that same long term technical vision I had and could provide experience that I don't have and just being kind of a new PhD grad.

[00:29:41] JM: In the avenge of these API kind of business, like the Twilios and the Stripes of the world, my understanding is you just have good unit economics from the beginning, because every API request is just – You're like, "Okay, we just need to charge enough to make money

from it." So you're like profitable early on. It probably does let you control your destiny a little bit more, which I guess would explain the seven years on nothing more than seed and series A.

[00:30:10] MT: Well, definitely yea. I'm no expert in what makes consumer internet companies popular or successful, but I know with enterprise, we've got like a well-defined technical challenge, which is getting people accurate timely data. It's an existing problem that many businesses have, businesses run on on structured data and information, and a huge chunk of what knowledge workers do is just gather and maintain and update data, databases.

We're not trying to convince people of sort of some newfangled way of doing business or trying to create some demand. It's already there. It's up to us to make sure we deliver on accuracy, and it's a high bar when you're comparing it against human processes. That makes it challenging.

[00:30:53] JM: Definitely. Okay. So I want to get to the engineering. I kind of wanted to – Even though I know you're, as you just said, like you're not trying to convince anybody, but I guess I'm trying to be your hype man, because I just think this is really, really cool product especially for kind of how long it's been out there and how long you've been working on it.

This knowledge graph that you now have assembled and you now do some crawling around it and you now also offer crawling as a service, because you've gotten much like you've built up a core competency extraction. You've also built up a competency in crawling. So you have this crawler as a service. If somebody wants to crawl an entire domain, for example, you can help them do that.

But in any case, your goal – I mean, the whole extraction and crawling thing is sort of a Trojan Horse just to build up a knowledge graph of the world. So this knowledge graph, it is like a database. What's the model for querying that database? I mean, do you use a graph database? Do you store it in-memory? How is this knowledge graph of the world stored and queried?

[00:31:58] MT: Just getting some rough stats. There's about on the order of 10 billion entities in the Knowledge Graph right now and about a trillion facts that would get — which are the edges. The entities are the nodes. The edges are these facts in the Knowledge Graph.

To give you an example, Mike Tung is – Me, I'm a type person in our Knowledge Graph Diffbot. It's a type of organization, and that relation is employed. I work at that company. It's quite a large database, and if you think about most traditional databases, the information to that database is largely human generated. I think we're going to really push the limits of these large database systems when we have this AI system that's essentially discovering entities and generating and has – It reads documents on the web. We discover 150 million entities per month and we haven't really even scaled up the datacenter yet.

So when we were thinking about how do we store the Knowledge Graph, which is essentially the output of this machine learning pipeline, the input is the web, which is much larger, and any output is this highly confident, highly accurate information, which is the Knowledge Graph. How do we store the Knowledge Graph in a way that's queryable?

We evaluated basically like a dozen or so of the current of the shelf graph databases, and what we found is most of them generally fail or lockup between 10 million to 100 million entities depending on which one. So none of them really met our engineering requirements. We build a new knowledge graph every two days. So just to load the data into one of these databases would take weeks, but most of them never even got to a point where they load 1% of the data.

So what we ended up doing is making some compromises. We de-normalized our data to edges out and injected them into a key value store. Then what we do is we mapped the incoming DQL, debug query language queries into queries in the store. That's how we can return results pretty fast. But it's still an area that we're actively researching. We're keeping in pretty close touch with all these graph DB vendors, because they all would be pretty interested in being able to host the world's largest Knowledge Graph and using their system. We would love not to have to build our own graph database, but we do want to be able to have a design that would scale up to like a quadrillion entities.

[00:34:23] JM: The model of storing all the entities in a document database where it's de-normalized and you can store up to two edges out in I guess one of the fields. Did you take one of those document databases off the shelf, like Cassandra or something?

[00:34:41] MT: Yeah. We're constantly experimenting with a couple. Cassandra was one we tested. Elasticsearch is another one we tested. Basically, key value stores that can operate on multiple machines is kind of that space.

There are some tradeoffs to this design, I should mention. So when you just de-normalize a couple of edges out, that means you can't run a query. You can't run certain kinds of graph queries, at least not using DQL. I mean, you can run these offline in our own development system, but you can't make a query that spans like three or four tables, right? You can't say, "Give me a person who work at that organization, or that organization, it's based in a city, where that city is also a place where another person lives, right?

But what we found is at least more than 90% of kind of our business customers and developer customer don't actually need to write queries that join four tables, or write queries that have to involve traversing a random walking, like a graph.

[00:35:36] JM: Man! I do not envy these engineering problems. That's tough stuff. By the way, can you explain what de-normalization means in this context? Because there's people listening who don't really understand what that means and why you need to do it.

[00:35:53] MT: Yeah. What de-normalization is, it's a database term. It's a way of making queries faster by making a copy of the information that is one edge out. What one edge out – If you are in the relational base field, you have this concept of like primary keys and for end keys, right? If you have a reference to another row and another table, what de-normalization would do is make a copy of that other row and put it into your table. So you're duplicating the data across the places where it's referenced.

The disadvantage of that is you have data duplication, although you can overcome that with some clever compression. The advantage of that is when you make a query, it doesn't actually have to hit that other table, because information is already been copied over there. It's superfast querying.

[00:36:41] JM: This database is stored in your own data center you said?

[00:36:44] MT: Yeah. We have essentially colocation facilities. We have two in the Bay Area, in Freemont, where we have basically a bunch of – We have a lot of machines that are just hand built here at Diffbot and then we rack them there. We also do use cloud services to handle auto scale for the automatic APIs. So we try to serve as much of the load off of our bare metal, but we have sensors that can detect queries of high load, spin up VMs on the cloud, on GCE and native AWS to catch that load and then scale back down when it's not even.

[00:37:15] JM: What's the motivation for running your own machines?

[00:37:19] MT: We did like just kind of the cost benefit analysis and the payback period is measured in months basically from programming our own machines. Another reason – Or some of the machine learning workloads that we have – Our typical machine to give you an idea of what our build out looks like, is basically like a 48 core and has like 32 drive [inaudible 00:37:41] 4 terabyte SSDs times 32 rated together, and about a terabyte of RAM and with some with multiple GPUs. They're pretty large machines and you can't really rent instances like this right on GCE or AWS currently.

[00:37:58] JM: This is just because you have weird querying semantics. Is that ultimately – You need to serve these queries really quickly because they're hitting this knowledge graph that's built in a weird way.

[00:38:07] MT: No. It's not because of the querying actually. It's because of the machine learning pipeline that actually builds it. This whole process of taking webpages and building a knowledge graph involves a lot of computer vision, a lot of natural language processing, a lot of determining if this entity is the same as that entity and fusion. A lot of those algorithms just require high amounts of memory and compute.

[00:38:29] JM: Wow! I love the fact that you're running your own hardware too. It's just like a big ball of heresy is your business. The actual database itself, do you store that in these machines too?

[00:38:44] MT: That is true. Yeah. The datacenter, what it looks like, is essentially a whole bunch of different farms. Diffbot is pretty much similar to like a Google scale operation. We don't

have quite their employee headcount, but it's all these different farms of servers that have some function. We have a farm that performs rendering. We probably run one of the largest rendering deployments, right? Because we're basically not just reading the HTML of the web, or actually rendering in a browser, all the pages on the web. So we have farms and machines that just do that, they're farms and machines that do natural language understanding. So take text and understand, "Okay. In this text, what language is it? Is it English, Arabic, French, German? What are the entities in that text and the relationships involved between those entities?

We have other farms that are responsible for looking at the pixels of an image and determining, "Okay. Inside this picture, I have a Macbook Pro," or "inside this other picture, it's a picture of a cat," or "in this other picture, it's a BMW i3." So these are all kind of specialized computer vision algorithms that detect these entities.

[SPONSOR MESSAGE]

[00:40:00] JM: This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with container technologies like Docker and Kubernetes so you can monitor your entire container cluster in real-time. See across all of your servers, containers, apps and services in one place with powerful visualizations, sophisticated learning, distributed tracing and APM.

Now, Datadog has application performance monitoring for Java. Start monitoring your micro-services today with a free trial. As a bonus, Datadog will send you a free t-shirt. You can get both of things by going to softwareengineeringdaily.com/datadog. That's softwareengineeringdaily.com/datadog.

Thank you, Datadog.

[INTERVIEW CONTINUED]

[00:40:55] JM: Let's talk a little bit more about the computer vision and machine learning pipelines that are taking place on your own datacenter. So you've got this knowledge graph. It's hosted in this datacenter, and when you hit a page, either you're parsing the page yourself

through your crawler or somebody has made a call to the extraction API and you get data for free out of that. In any case, you get this additional piece of data and maybe it's a new webpage. You can derive some meaning from it using NLP. You find some entities in it and you've got these entities and you can kind of look in other entities in your database that you've already got and you can sort of figure out how to merge this new page and its entity information with you preexisting model.

You also do this thing you call knowledge fusion, which is where you assign truths using probability. So you can sort of say facts about the page and the entities on the page and how this entities relate to each other and how these entities might relate to other things in your database, and you have a probabilistic model for doing all of that, which I love so much, because you even just think about in today's world of assigning truth to news stories and people have this binary outlook of real or fake news, or you talk about science. People have this binary outlook, like, "Does this drug work to treat cancer or not?" Many times, the actual truth that we can say scientifically objectively is a probabilistic model.

[00:42:38] MT: Yeah, it's a distribution.

[00:42:40] JM: It's a distribution. It's not a binary model. I just love that that is baked in to the Diffbot model of the world. It's like we're not sure about this. We just the probabilities that we assign to it. Tell me more about how you're doing this model of building the world and continually learning about it.

[00:43:00] MT: Yeah. We treat the internet as essentially the world's largest manmade sensor. When you're combining and getting information from multiple pages, there's many reasons, there are many factors of noise involved I'm afraid. There could be noise because one page is still just very practical. You might not have updated your LinkedIn profile on a while, or your current company page has the correct current information about you. It just changes overtime? Your school webpage, it says you're still in college.

Also, different pages have different aspects of an entity, like Instagram account is going to have very different information than your professional CV. There could be errors, just not necessarily malicious errors to the degree of some Russian troll farm, but very benign errors. Just like there

could even be errors in extraction. So you have all these possible sources of potential noise or difference. You might have in product, one page it says this product is five ounces. In another page it says 5.2 ounces. How do you believe that? Which one do you believe and put confidence in, right?

It's much the same problem like human has to do when they're using Google, right? You query for something, but then you have to read all these pages and then sort of like you have to synthesize the information about all these pages to come up with a clear picture of what you really believe. We call this process knowledge fusion, and we have something akin to like a page rank kind of algorithm, which recursively assigns like trust to origins that have a history of producing true facts or just that are trustworthy, we assign greater weight to their facts as well. It would take – If there was some new source – I gave an example earlier of Wikipedia, Wikipedia has a lot of good factual information about it and they've been around for a while. The algorithm would naturally assign much more trust if that fact was produced from Wikipedia than like a brand new WordPress blog that just came up last month.

However, if it got enough signals from these news sources, it could eventually overweigh this kind of like prior. It's very similar to just how the scientific process works. You have preponderance of evidence. You can definitely counteract kind of like the inherent prior trust worth of an origin.

The other kinds of things you can do in Knowledge Fusion are to actually use your knowledge graph. For example, there was some page out there on the web that said Mike Tung lives on the planet Venus. We know in that Knowledge Graph that Mike Tung works at Diffbot. Diffbot is in the Bay Area. It's in Mountain View, California. California is in the United States, which is on the planet Earth. Earth and Venus are millions of miles away. It's very unlikely that I commute millions of miles each day from my home planet of Venus to Earth. You can use things like ontological reasoning as well, logical reasoning over the Knowledge Graph in order to discount certain spurious facts. It's a quite an active research area.

What we are trying to do is we're not so much trying to weigh in on this more or like kind of subjective topics of the day. These are all pretty much facts that are objective statements, that 99 humans would agree, this person has this gender, works at this company. We're not trying to

determine people's preferences or these other – Or sentiment or these kind of more subjective measures, right? It's easier to validate.

[00:46:30] JM: But these rules around truth probability. For example, something like you said, where you might have information about – Like coming from a Wikipedia article, versus a WordPress site that was put up yesterday. The Wikipedia article has been around for 10 years. Maybe you've got a lot of traffic data around it. Maybe you've got a lot of citation data around it. You could do a page rank kind of thing in order to understand its trustworthiness, but ideally you would like to be able to have machines derived these rules, these trust-based rules, rather than hand-defining the trust-based rules, like, "Oh! This page has been around for 10 years. Therefore, we trust it more than a page that's been around for one year."

So when you think about this truth system, this probability of truth system, is there a way to have a deep learning model for assigning probabilities to facts and truths?

[00:47:35] MT: No engineer like hand codes that we trust Wikipedia. It's the fact that there's this kind of recursive definition, right? The original page rank paper. Basically, the reason we trust Wikipedia, because it has a history of generating facts that were determined as true and is collaborated by other sources. So therefore, it gets sort of like more truth weighting to it. Because it has more truth weighting, then it can also give more truth weighting to other sources. It's similar to how page rank initially worked. If no one defined that any website was more important, but if it got a lot of backlinks from other sites across the diversity of places that were also important, then you could infer using machine learning that that was important.

The kind of truth that it converges to is sort of like the consensus of what like the most well-informed person would arrive at. But it's obviously possible for everyone in the world to just be wrong about something, and we just discover later that this wasn't true. We're just trying to meet the bar of human quality.

[00:48:43] JM: Google has a Knowledge Graph. I heard you say in a talk, and I think it was your talk at Strata, that your Knowledge Graph is 500 times larger than Google's Knowledge Graph?

[00:48:56] MT: Yeah, that's right. First of all, I would say you should always take these accounts with a good grain of salt, just like when we were kind of the first search wars. People would talk about whether Bing was larger, whether Google was larger web index. I think the much more important metric is does any given Knowledge Graph have the entities that you care about in it?

For your particular use case, if you are doing sales, you're building an app, you're doing market research, is that universe of entities that you're concerned about in the Knowledge Graph. If you look at the Google Knowledge Graph, which his basically that panel that shows up on certain queries on the right hand side, you'll notice it's pretty obvious that most of those entities come from Wikipedia, and Wikipedia is essentially an encyclopedia of celebrities, or the famous people. We call those head entities.

I don't know if you or I have Wikipedia entries about us or your family or coworkers or kind of the day-to-day people that you interact with in your job, the kind of people that you might be doing business with, might be recruiting, might be your customers. Generally, none of those people have Wikipedia pages. I would say probably less than 1% of those people that you interact with daily have Wikipedia applies, and it applies to other entity types too, the products that you interact with, the places and restaurants we go to.

I'm not sure that the Chipotle down the street from where we are at our office has a Wikipedia page. But all of those entities I just mentioned are actually in the Diffbot Knowledge Graph, because they all have web presences and we're able to extract from them and fuse them and put them in as entities. So we did our own sort of benchmarking, taking universe of these entities and querying whether they were in the Google Knowledge Graph versus ours and found that significantly less than 1% of them are in the Google Knowledge Graph.

[00:50:59] JM: Google, do they offer the kind of things that you offer as a service. Do they have some kind of extraction API or like a knowledge API?

[00:51:08] MT: No. They don't have any sort of extraction API at all. The Knowledge Graph API they have is severely handicapped. It just returns back the name and description of an entity, these are generally Wikipedia articles. You can't get the properties or facts about all those entities.

[00:51:23] JM: Weird.

[00:51:24] MT: Yeah. If you are looking for an open source Knowledge Graph, the best one I can recommend is Wikidata, which is a crowd-sourced structured Knowledge Graph that is produced by the Wikimedia Organization, the same folks that do Wikipedia.

[00:51:36] JM: Okay. A couple more question. You do use cloud for some things like serving API requests and scalability, maybe you use it for fault tolerance or backups. Can you tell me more about your interaction between the cloud and your datacenter?

[00:51:54] MT: Yeah. Like I said, we try to serve most request off of our bare metal, because that's the most efficient way of doing things and most cost effective. Because we have an on-demand service, which is this extraction API, there are periods where we just don't have a physical hardware to serve all of those requests with a certain latency guarantee.

So what we do is we use the cloud primary for auto scaling the farms. So we've written scripts that can detect the machine load and can spin up virtual machines really quickly to match the load that's coming in. Then when it detects that load as leveled off, then we'll scale it back down. I think every 30 seconds to every minutes, machines are scaling up and down.

[00:52:38] JM: Okay. Final question, this is another thing you talked about a little bit at Strata, but you have ideas about how knowledge work will change in the future, particularly pertaining to how maybe white collar jobs, white collar knowledge work jobs will evolve and the role that a knowledge graph might play. Can you share a little bit about that vision for the future that you have?

[00:53:02] MT: Yeah. As a summer job back in high school, I used to work as a data entry person in a call center doing fulfillment for BellSouth, which is like AT&T. The lowest levels of knowledge work is just about gathering information and entering information. It really aligns with our mission to try to automate some of these sort of knowledge front work as much as possible. I think human beings using their brain power to derive insight from information and think of new

ways of getting information rather than spending upwards of 30% of their day just entering in information in various databases and keeping these databases up to date as the world changes.

We have a really strong belief that in the future, because of technologies like Diffbot that essentially automate knowledge acquisition, the relationship between these systems and humans beings is going to be one where these systems are doing most of the heavy lifting, gathering the information, using the internet as the world's largest sensor to update your CRM, your sales database, your recruiting database, your inventory and parts databases. Humans will essentially just be managing these autonomous systems and directing them as to how to get information that they need rather than doing the data entry themselves.

That's kind of what we're starting to see already actually as the Knowledge Graph is being used in many sorts of intelligent applications. So we see, yeah, kind of this trend continuing. I think it's going to leave knowledge work in a much better place, like analogy, just think of the agricultural revolution, where the past people were actually manually plowing and tilling the fields. Whereas a modern farmer is actually just managing like John Deere Tractors that are GPS guided and the machines are doing the heavy lifting. I think the same is going to be true with knowledge work.

[00:55:00] JM: Mike, thanks for coming on the show and thanks for working on a really cool product that I have made good use of myself.

[00:55:06] MT: Yeah, thanks a lot for having me. This has been really fun.

[END OF INTERVIEW]

[00:55:11] JM: Kubernetes can be difficult. Container networking, storage, disaster recovery, these are issues that you would rather not have to figure out alone. Mesosphere's Kubernetes-as-a-service provides single click Kubernetes deployment with simple management, security features and high availability to make your Kubernetes deployments easy. You can find out more about Mesosphere's Kubernetes-as-a-service by going to softwareengineeringdaily.com/mesosphere.

Mesosphere's Kubernetes-as-a-service heals itself when it detects a problem with the state of the cluster. So you don't have to worry about your cluster going down, and they make it easy to install monitoring and logging and other tooling alongside your Kubernetes cluster. With one click install, there's additional tooling like Prometheus, Linkerd, Jenkins and any of the services in the service catalog. Mesosphere is built to make multi-cloud, hybrid-cloud and edge computing easier.

To find out how Mesosphere's Kubernetes-as-a-service can help you easily deploy Kubernetes, you can check out softwareengineeringdaily.com/mesosphere, and it would support Software Engineering Daily as well.

One reason I am a big fan of Mesosphere is that one of the founders, Ben Hindman, is one of the first people I interviewed about software engineering back when I was a host on Software Engineering Radio, and he was so good and so generous with his explanations of various distributed systems concepts, and this was back four or five years ago when some of the applied distributed systems material was a little more scant in the marketplace. It was harder to find information about distributed systems in production, and he was one of the people that was evangelizing it and talking about it and obviously building it in Apache Mesos. So I'm really happy to have Mesosphere as a sponsor, and if you want to check out Mesosphere and support Software Engineering Daily, go to softwareengineeringdaily.com/mesosphere.

[END]