EPISODE 1410

[INTRODUCTION]

[00:00:00] KP: Tabnine is an AI assistant that helps software engineers write more efficient code. It's been trained on a large corpus of source code or can be trained based on your specific team's codebase. Either way, the resulting model offers predictive completion of code that can make programmers more productive.

In this episode, I interview Eran Yahav, professor at Technion - Israeli Institute of Technology, and CTO at Tabnine.

[INTERVIEW]

[00:00:30] KP: Well, Eran, welcome to Software Engineering Daily.

[00:00:35] EY: It's great to be here. Thanks for having me.

[00:00:37] KP: Well, to kick things off, tell me a little bit about Tabnine.

[00:00:42] EY: Yeah. So Tabnine is an AI assistant to help you and your team be more productive when you're writing code. Right now, it focuses mostly on AI-based code completion, which means that you work as usual in your IDE and Tabnine contextualizes on what's there on your entire project and basically predicts what is it that you would like to write next in your code and completes that for you. So you can think of it as like smart compose, if you're familiar with that, from Gmail, in natural language for your code. And we have order of magnitude millions of users who are using this in their IDEs. And it helps them write significant amount of their code every day. I can talk later about in more detail about kind of automation factor, and how we measure things, and what kind of uplift we're getting.

So Tabnine is basically a sophisticated AI system that has been trained on huge amounts of code from various sources and is then able to predict code for you in your IDE. It can also train on your own code base, your team's code base, and produce private models for you and your

team, which gives you more – Gives the AI more knowledge about the nuances in your code base, the concepts in your code base, and therefore leads to a significant uplift in Tabnine's ability to complete relevant stuff for you in your project as you're working. So that's kind of the short story of Tabnine. I can speak more to that. Just tell me what's interesting.

[00:02:41] KP: So I think most developers will be familiar with that sort of experience where you type, I don't know, a variable name and hit dot, and you get auto suggested things that are very rule-based. It's like the compiler has dished up the list of possible methods for my object or something. We're really talking about something more sophisticated here, right?

[00:03:01] EY: Yeah. We're talking about something that is completely different. First of all, what the IDEs typically give you is, as you said, the list of things that can be done. Like if you do x.something, you get all the things, all the methods or functions that are available to apply on this variable X. But what Tabnine would give you is the things that should be done. And the things that should be done are based on what Tabnine knows or has learned from all the code in the world. And in particular, Tabnine provides completions not only when you hit dots. Tabnine provides completion at any point in your code based on the prefix of what's available there and based on basically richer context that is available than what is available to the IDE when it makes its predictions.

More differences are, again, Tabnine contextualizes on rich context that also includes natural language. So it understands the intent of what is it that you're trying to do and it also predicts more than a single token, which is what you get from the IDE when you get the dot completion. So it's a completely different animal. It's a system that uses AI to provide these predictions as opposed to something that, as you said, is rule based and kind of like baked into the compiler or the backend of the IDE.

[00:04:29] KP: And how much of an auto complete am I going to get? Is it going to finish my line? Give me a few other lines? What's the, I guess, expectation I should have going in?

[00:04:38] EY: It would definitely finish your line. It could do less than that. It could do more than that. Complete a few lines and complete the whole function body. It depends – Really, what we found out is that it really depends on the kind of interaction or the kind of cognitive load that the

user is willing to take. So it turns out that if you provide completions that are too large, let's call them let's say 30 lines of code or something like that, the user, the developer is often overwhelmed by that in the sense that, "Hey, I just got 30 lines. How do I know that what's in it is actually correct? Do I understand all the nuances there? Maybe there are subtle bugs there. Maybe it's not doing exactly what I wanted." So it's very hard to adopt these larger snippets. Unless you're looking at something that is very idiomatic. And so there's a tension here between the model, which is super powerful and can predict really a lot of the code forward and the human, which wants to consume this typically in smaller bite-sized pieces. Because what the human wants to do is to make snap judgment that what they're getting is actually correct. So what we've found through experimentation, and Tabnine is quite sophisticated in that, on what to suggest when in order to actually accelerate you and not slow you down.

[00:06:08] KP: Could you expand on that? What to suggest when sounds like the key to it, but also a tricky problem to solve.

[00:06:15] EY: Yeah, it is a tricky problem to solve. So for example, imagine that you just wrote a comment or a method signature. That is a point in the code in which Tabnine could make a more educated guess on the following steps, because the method signature or the comment often provide some clear intent on what is it that you are trying to do. And therefore, it's worth it. Kind of taking a bigger leap and trying to complete more.

If you're in the middle of something, in the middle of some logical flow in your code, it becomes quite harmful to your flow to actually suggest large pieces, because you're going, right? You're already doing something. You're already in the zone. And now you just want Tabnine to fast forward you to kind of like the next turn in the road and not fast forward you five miles out where you are disoriented. And so this is kind of a very – And I just gave like the obvious examples. But there are much more subtleties, many more subtleties to this kind of like decisions on how much should you suggest when. It depends on different languages, different levels of complexity in the code and other things.

[00:07:38] KP: So developers write code in lots of different environments and different tools. Where can I get Tabnine?

[00:07:45] EY: Yeah, the beautiful thing about Tabnine is that supports out of the box around 31 languages and works for most of the major IDEs. So it would work obviously in JetBrains ecosystem, it will work in VSCode, Visual Studio, Veeam, Emacs, I don't know, Sublime. I personally work mostly in Sublime text. So that's my edit worth choice. So obviously it works there as well. I can talk later about the deployment and how we make it work in all of these IDEs, because it's kind of a nice engineering trick, I think.

[00:08:27] KP: Yeah, that is interesting. I mean, I guess we tell people where to go to get it installed as a first step. Is there a general place for all these spots? Or do I have to check in on my tool?

[00:08:36] EY: You basically have to install Tabnine knife through your IDE as an IDE extension. So each environment install plug-ins slightly differently. But we want this to be friendly and work in your environment. So in Sublime, it's through package control. In VSCode, it's through the extension management, and you install Tabnine. And the Tabnine extension, within install itself, it also pulls down the Tabnine inference engine, which is the part that does the heavy lifting of running the models and communicating with Tabnine servers. When you have Tabnine cloud enabled and all sorts of this, more complex interaction with the model. And the IDE extension is just a shell that works over this inference engine.

[00:09:28] KP: So does that mean the inference in compute is taking place on my local device?

[00:09:33] EY: So that's definitely one option. So I think one important theme in Tabnine is that you, the developer, are in full control. You can decide what model is being used for inference. You can decide where this model runs. It could run on your local machine. It could run on Tabnine cloud. It could be self-hosted. You have the freedom to choose where you want to run inference and what kind of models, which particular model you want to actually run. Yeah. But definitely one of the options is to run Tabnine inference locally on your machine.

[00:10:12] KP: What are some of the reasons I might choose cloud over local?

[00:10:16] EY: Yeah. Basically, again, as a developer, maybe your company does not allow any external tools to have access to the code, or maybe you don't want to run anything that sends

code elsewhere. Obviously, for inference, Tabnine cloud needs to get some context from your editor, which is obviously not all of your code. But it does take small pieces of your code as context for inference. And if you don't want that to happen, you can run completely locally and actually completely air-gapped and have Tabnine run on your machine.

Obviously, this limits the kind of models that you can run, because your machine can run just as much as – Just a limited inference power compared to Tabnine cloud. So for larger models, you definitely have to use Tabnine cloud. But again, you have the freedom to choose wherever you want inference run. And Tabnine also in blended mode where you run some of the inferences locally and some on the cloud depending on availability.

[00:11:25] KP: So when I install Tabnine and I get up and running, do I just have out of the box things are good to go? Or do I have configurations I need to go in and pick which model in which language and things like that? What's the onboarding experience?

[00:11:38] EY: Yeah. So Tabnine works out of the box. No configuration is required. Configuration is possible if you would like to run larger models, or specialized models for particular language, or something like that. You have the ability to customize Tabnine. But Tabnine would work for any language and editor out of the box.

[00:12:00] KP: There are some languages and I guess some teams that are very specific about the way they want a language written. They might have a linter with a bunch of specific things in it that perhaps Tabnine doesn't have the same specificity or style of coding that this rigid organization wants. How can I get benefits? Yeah.

[00:12:21] EY: Absolutely. One of the cool features of Tabnine is it allows you to train a custom model for your team based on your code base or parts of your code base. Again, you can control what exactly goes in the data set for training. And you will get a customized model for your organization that does understand exactly the things that you talked about, the style, and maybe even kind of the concepts of the problem domain in which the organization is working. Because, again, it would know the code. It would know the libraries that are available in the code. It would know the APIs being used, etc., etc. So you will get a customized model that is more familiar with your environment.

[00:13:02] KP: Well, let's say I'm starting a greenfield project. I've got five or six team members, and we want to get going on this. And we definitely want our own custom model, because we're going to work in some niche industry that has a lot of specific jargon in ways it does stuff. And also I'm a stickler for certain code styles. But on day one, we have zero lines of code to train with. Can we start from some pre-trained model and ease into our own?

[00:13:27] EY: Yeah, absolutely. The universal model that you get from Tabnine, or you can pick also language specific models, will give you a good starting point. But again, as you just said, if you're working on the particular jargon or particular niche, then over time, Tabnine will learn. You will connect it to your repo, and over time, Tabnine will learn your project specific things. And Tabnine is also learning and training on the fly as you are writing code on your machine. So if you install Tabnine, as you work with it, you will notice that Tabnine becomes more aware of your environment locally over time even if you did not connect it to the repo. So it will improve by kind of learning from your local code base in your project.

[00:14:19] KP: So if I'm just going to rely on a generic model or I don't need my own custom trained model, where was that trained on? Can you describe the initial corpus that came about for the model?

[00:14:30] EY: Yeah, sure. So Tabnine has been trained and is continuously being trained on open source projects with permissive open source licenses. So that means Apache 2, BSD 2, BSD clause 3, and these kind of permissive open source licenses. We don't train on any GPL code or any non-permissive open source licenses. There's enough code out there with the permissive licenses for Tabnine to train on.

[00:15:02] KP: And there's a style I've seen a lot of people using in Tabnine where you'll first write a comment in whatever language, just in English, saying this next function will do X, Y and Z. And that this does seem to help the model produce a more precise results. Is that a style that you guys were expecting? Or is that something that has just emerged over time?

[00:15:24] EY: No. We kind of expected it, but also built the product around these kind of hints as I mentioned earlier, where Tabnine kind of triggers itself more aggressively as exactly at

these stop points where the developer is just about to transition from thinking about the declarative description of a task to actually how to do it. And this is the place where Tabnine, as users, have noticed, is making more aggressive predictions. And so I think this is kind of a behavior that reinforces itself that users learned that and then they started writing more comments. And actually expect this style of programming to increase as AI becomes a more inherit part and built in part of software development, which I think this is exactly what Tabnine is doing.

[00:16:20] KP: So when you look at the far horizon, I mean, what's the goal here? Do you want to put all software engineers out of a job? Or will it change to a job where we just, in natural language, speak our wills into existence?

[00:16:33] EY: No. I don't think so. Software development is not about compiling natural language descriptions to code, because the natural language descriptions would not be precise enough to do that. And they will become complex on their own and then you will just do programming in English, which is kind of seems misguided.

I think software development is actually about discovering specifications by doing iterative refinements, right? The whole point is that we don't know the exact specification when we sit down to write the code. We kind of have some idea or we got some high-level description of what the function or the project should do. But we don't have like a step-by-step description. And then we start writing the code. And as we write the code, we discover all sorts of subtleties that were not present in the partial specification that we got initially. And the actual job of the developers kind of like carve out the program or the full specification that is hidden in this partial specification, right?

So I think the goal is to accelerate the mundane parts of the job to make the system, the development ecosystem, much more forgiving and more accessible to more people. Just imagine how hard is it today for someone to write code. It's kind of a foreign language that is extremely unforgiving, right? It's enough for you to forget a semicolon, or to forget a parameter to a function, or add an extra parameter, and things go awfully bad. So the goal is to help you go through the foreign language more efficiently as you're interacting with it.

Yeah, I think, again, the important part here is that what we're trying to automate is the mundane stuff. We are trying to relieve the programmer from dealing with the nasty syntax of certain things, or what exactly – What was the name of the function or the order of the parameters? And kind of focus on the more creative part of the development job.

[00:18:47] KP: Well, Tabnine has a free level for developers and offers a solution that, at worst, I ignore it, at best, it makes me much, much faster, more efficient maybe catching obvious bugs. So for developer economics, it's sort of an obvious tool I want to explore. Are there other benefits that you see especially in groups, the other metrics and things besides just being a nice handy tool that makes developers feel comfortable?

[00:19:14] EY: Yeah. I think one of the important things that happened in teams is that think about how much – If you don't start a project from scratch, let's say we have already some code base. Think about how much knowledge and discipline of the team has gone into the code base, right? People have done code reviews, and people have discussed how to write certain things. And all that information went into the code base and is sitting there passively. What Tabnine allows you to do is kind of tap into the richness of that information and bring it to every developer on the team as they write code, right? So it becomes kind of an active single source of truth on how to write code in the organization.

And quite frankly, it exposes you to things that you didn't even know were done in your organization, right? You start to write something, and then Tabnine pops up and says, "Here's a completion." So like, "Hey, wait. Somebody probably has done something similar, because Tabnine has learned that this is the way we do these kind of things in our code base." So you get exposure to things, knowledge, that has been locked in the code base sitting there passively. And you also get harmonization of the code base, because people are likely to take the suggestions and basically keep themselves on kind of the main path where all – Know what has been done before or things in similar style to what has been done before. So you just keep the entire development team on the same path aligned.

[00:20:53] KP: And are there any KPIs or metrics? Any qualitative data points – I'm sorry. Quantitative data points you can look at in terms of improvement when people start using Tabnine?

[00:21:05] EY: Yeah, absolutely. So we look at something internally, but actually we also externalize it to developer, called the automation factor, which is basically the ratio of the things that Tabnine or the characters Tabnine predicted for you divided by the total number of characters that were created, right? So intuitively, the automation factor means how much of the code created has been created automatically by Tabnine.

And we see numbers, depending on language, and depends on also the nature of the code being written. We see numbers as high as 43% automation factor. Again, depends on the kind of code that you're writing. If you're writing like the one-off algorithm, you will not get this kind of automation factor. If you are writing some idiomatic Java code reaching boilerplate, then Tabnine can definitely reach super high numbers of automation factor.

[00:22:13] KP: And do you see any styles of code or industries that get a major punch up? Like what at about people using specific frameworks? Do they get a leg up on predictability?

[00:22:22] EY: Yeah, absolutely. So again, the whole point is that APIs are extremely repetitive, right? We have all connected to MongoDB the same way or pulled some data from Kafka the same way. And it's very easy to recognize the piece of code that does that once you see it. And Tabnine is really good in automating these kind of idiomatic pieces of your code. And so I wouldn't say that it's necessarily specific to an industry or to a language. I would say that if you're working with API-heavy code, which I think is most of the industry these days, you will get significant uplift from Tabnine. Definitely, also, in UI frameworks, React, Vue, these kind of things, that again are extremely repetitive, you'll get significant uplift from Tabnine.

[00:23:22] KP: When I think about if I were to start a new web app and I picked a framework like React or one of the other ones you mentioned, there's a very yellow brick path for how I get my app stood up and build it and deploy it. It's been done by so many people. We all know how to deploy web apps. It's not obvious to me how you train one of these code prediction models and then get it running inside my IDE. Can you talk a little bit about the engineering steps for how you deploy something like this?

[00:23:49] EY: Yeah, sure. So Tabnine is using a bunch of different models at different scales. So it's really, typically, when you run Tabnine, you're running an ensemble of like three or four different models, which some of them are running on your local machine. In terms of the engineering, I think part of the – Obviously, we have our own kind of inference infrastructure that allows us to run these models fast enough and accurately enough on your local machine CPU only, which is kind of, I think, an engineering marvel. It's done by a very highly optimized low-level implementation of the inference steps required for our specific model. So this is like an inference system that is optimized for the particular model that we're running. So we'll be able to run it CPU only on your machine.

It also goes up to the way in which we train the model and we build the model such that it could run in these conditions. Apart from that, in terms of engineering, this is almost – All of Tabnine 9 is written in Rust. There are some touches of C++ and assembly code involved. But most of Tabnine is Rust, which really helps get the high performance that we're getting from the inference engine and from Tabnine throughout. So I don't know how many listeners of these podcasts are Rust developers. But we're huge fans. I think we're pretty a big Rust shop.

[00:25:39] KP: More and more of them each day from some of the anecdotal evidence I have seems to be a language growing quickly in popularity.

[00:25:46] EY: Yeah.

[00:25:47] KP: Would you mind comparing and contrasting Tabnine to GitHub Copilot?

[00:25:52] EY: Yeah, sure. So Copilot is awesome. I think we've seen some of the evolution in the user experience that we've been going through with Tabnine that we've done like a couple of years back. User interaction here is kind of very hard to get right. And we see part of the evolution that we've gone through happening in Copilot, which is nice to see the model that Copilot is using is Codex, which is at the high-end 180 billion parameters model, which is huge. Running this kind of model, it can happen only in the cloud obviously.

I think, functionality-wise, it kind of follows what Tabnine is doing. Some of it, it is currently doing a little better, especially natural language in Python, because it has been trained and modeled

specifically for that purpose. Some of it, it does worse, JavaScript and others. But again, when you are running in Copilot, you are basically a subject in Microsoft's kingdom. When you are running Tabnine, you are your own emperor of your own kingdom, right? You decide which model, what it has been trained on, where telemetry goes, what exactly happens. And I think this is really powerful that you are in full control of what exactly is going on. You can train your own models on your own code and decide what to run when and where.

[00:27:32] KP: So that's got to be really appealing.

[00:27:34] EY: Yeah, if you look long term, I think the area of AI assistant to software development is going to be huge. I think when we started, we had a hunch that it is going to be huge. And I think it is increasingly validated that five years down the road all code in the world is going to be touched by AI even if it's not completely – Every line has been automated or partly automated. Then code will be reviewed by AI. Tests will be generated by AI. So the whole ecosystem of ai assistance is basically going to see massive growth. And I think Tabnine has been a pioneer in that. And we're happy to see this category get additional validation.

[00:28:33] KP: I think, in large enterprise, some of the things you pointed out are going to be very critical. When I want to run in a closed environment, or trained on only my models, or something specific like that, can you talk about how large enterprise IT adoption is taking place?

[00:28:49] EY: Yeah. So large enterprise is exactly doing what you said. It's about training models for the enterprise. Right now, most of them are running on Tabnine cloud actually. So that has been kind of a shift in what I would have expected. But I'm probably old school. I thought that more of them would like to self-host. But it is increasingly the case that even large-ish enterprises are willing to trust Tabnine and run on Tabnine cloud, of course with guarantees that they train their own model, which is accessible only to them. And no data from their usage is ever used to train any other models. There's no leakage of information, right? It is, as I said, your own kingdom in which all the user information is private, and the model is private and has been trained on private code, etc., etc. If you'd like, you can also self-host Tabnine and run it completely inside your VPC.

[00:29:56] KP: I know this is a deep learning-based approach. Deep learning has had some very public advancements in natural language processing, which is a similar, but different problem from code prediction. Are you using similar or different architectures?

[00:30:10] EY: Yeah, we're using similar architecture. So the solutions are inspired by NLP solutions, but we have much more structure to work with, right? We know program languages have much more structure to them. The syntax is known. And you can do a lot by tweaking the models to be language aware such that they don't have to relearn the language syntax just purely from examples, right?

So one of the thing, if you take like a vanilla language model built for English and you just say, "Let me throw code at it." Let's say Python code. The language model will have to re-learn what does it mean for program to have valid python syntax. Although Python syntax is well-defined, right? You can look in the Python parser and see exactly what are the rules of the Python syntax, right?

So the the cleverness is in building kind of the language models which are tailored for programming languages in a way that allows you to more economically represent the regularities in programs without having to relearn everything from scratch. And this is a combination of doing some, let's call it in the absence of better terms, some pre-processing or some pre-computation, to kind of extract irregularities from code before you train on it, right? But essentially these are – For the majority, these are sequence-based models. But the sequences that they train on are programs, and they have been adopted to train on sequences of this kind.

[00:31:58] KP: Well, natural language processing is pretty forgiving. I think we allow run on sentences, non-sequiturs. I can kind of change my topic mid-sentence, and you can often follow me. It's okay to make mistakes. I could probably even write some text and win a literature award while still breaking lots of grammar rules. If my code doesn't compile, I'm not going to win an award. How do manage it when you have syntax and such strictness to deal with?

[00:32:25] EY: Yeah. So for sure, Tabnine can make predictions that would not compile. That is totally possible in the realm of, as you said, deep learning. You cannot guarantee that all outputs will be syntactically or semantically valid. Tabnine is a combination of some syntactic model and

some semantic models that allow us to reduce the cases in which this happened. But we can't always know, right? Especially for kind of subtle semantic inconsistencies. So it is definitely possible that Tabnine would generate things that don't compile. But the beauty is that the way in which we involve the human in the process, it rarely happens, right? Because again, you have to – This tight loop of the machine and the human working together, where Tabnine is suggesting certain things and the human is accepting them. And if you build this kind of carefully, it's easy for the human to make a snap judgment of, "Yeah. I want this." Or, "No. I don't want this." Which, kind of together, the human and the machine get the codes to be in the right state.

[00:33:41] KP: Well, in recent years we've seen transformers and attention become a main feature of natural language processing research. Is that occurring as well with code completion?

[00:33:52] EY: Yeah, absolutely. And we are using – I heard your question about attention and transformers. So yeah. So we're definitely using attention-based model and transformer models as part of the models being used. In Tabnine, these are very powerful models. And again, their adaptations to code when you use them at the sequence level is very powerful. In research lends, we have done a lot of work on structural models that use more of the program structure as an abstract syntax tree with attention on parts of the tree to do code completion and code prediction. A nice work done by my former student, **[inaudible 00:34:43],** and by my current student, **[inaudible 00:34:44]**, who are working on these structural models, structural language models. These are extremely powerful and very nice, but they are still kind of too early for – Let's call that production use, right? It's very nice. And it's going to be useful, but there are some performance barriers to go through there before you can deploy these kind of models for realistic real-time use in the IDE, right?

Part of the challenge of Tabnine is that you need to make the predictions really quickly, right? You need to make inferences while the user is typing in order to provide useful feedback and suggestion that they can adopt. And when you go to these structural language models, their performance is not sufficient yet for production use. But they are very interesting, yeah.

[00:35:46] KP: You mean performance at inference time?

[00:35:49] EY: Yeah, inference time. Sorry. Yeah. Their accuracy is very good. And they can guarantee more of the syntactic structure of the result of the inferences. But their run time, or the inference times are not slow enough to be used in this particular use case yet. So we're working on improving that. But it will take a while.

[00:36:14] KP: Is there a fundamental reason why you'd have that challenge? That we can't just pay the price at the training time and get fast inferencing?

[00:36:23] EY: Yeah, it relates to the structuring the context for performing the inference. So when you are using these kind of models, you need to feed them with trees that are built from your code. These models work on syntax trees from your code. And the way that they use the tree is by walking paths in the tree. And if you're not careful, or actually almost always, the number of paths that you have to consider and look at is quite high, which affects the inference times. So again, there are some optimizations that we have not done there yet. These are kind of things that live in the realm of research.

There is definitely promise there. But the practical compromises have not been made yet. And once you start making them, their impact on the accuracy is yet to be determined, right? So definitely playing with that though.

[00:37:30] KP: Well, following up on that, when you think about where research can go, I'm thinking of two broad directions. Maybe you'll tell me there's more. But one is expanding the size of your training data, bigger and bigger corpora. And the other would be algorithmic advancements, whether that'd be more and more parameters or some clever architecture or something like that. Where do you think is the right place to invest your time?

[00:37:53] EY: Yeah. You can definitely brute force some of these problems with larger models, right? You can definitely throw 180 billion parameters in the problem. And then if you have sufficiently clean data and sufficient amount of data, you cannot – Basically, it will be good, right? But this is very non-economical way to tackle the problem especially given the fact that there's so much structure there in the programming languages themselves. So again, we're not talking natural language here. We're talking something in which the structure is known. And often, actually even the the semantics is known, right? The beauty of a program as opposed to

a paragraph of text is that you can execute the program. You can run it. You can see what happens, assuming it compiles, right? You cannot really run the paragraph of text to understand what it means. So program has defined syntax. They have defined semantics. And there has been a lot of research actually on trying to leverage these things and inform the model. Give the model more information based on program structure and program semantics. There has been a lot of work on modeling programs using graph neural networks. Very nice work by Miltos Allamanis and Marc Brockschmidt from Microsoft Research on these topics.

But again, these are things that live in kind of research land. They're typically too expensive to be deployed in this kind of real-time inference. But I'm sure that we'll see more advances on how to inform the model with additional syntactic and semantic information when you are working on programs, because kind of the natural progression of things is to make the models more specialized based on that kind of information, rather than just throw more parameters at the problem.

[00:39:49] KP: Do you see any opportunity to bring in auxiliary data sets like commit messages and pull requests and things like that?

[00:39:57] EY: Yeah, absolutely. These are definitely things that we've been doing both in research and in Tabnine, is to use this kind of information to inform different use cases and different tasks including surfacing information from code reviews while you're doing stuff in the IDE and things of that nature. It's definitely interesting and promising. The problem there is that you need to get this going early in the project if you hope to change the habits of developers to provide more information in their commit messages, for example. More informative clues in the commit messages. These data sets are very noisy. And depending on the project, they could provide really high value or not, right? But for well-disciplined, projects this works really well.

[00:41:01] KP: Yeah, very fair point. Even as I was asking, I was suddenly wondering if I had ever written a decent commit message. So I see the challenge there, for sure.

[00:41:09] EY: Yeah. Yeah, definitely seen my share of commit messages called fix or just dot, right? There are a bunch of those, right? And so there's definitely a challenge on the data cleaning and warehousing there. But we are playing with these ideas. Yeah, absolutely.

[00:41:32] KP: Well, is there anything coming down the pipeline you can talk about in terms of releases or next steps with Tabnine?

[00:41:38] EY: Yeah. So Tabnine is increasing the number of parameters in underlying models all the time and improving the models themselves. So we will see probably early next year the next range of improvements in all Tabnine models. We will see also improvement in inference speeds and quality. And we're also improving the pipeline of training on your private repos. So again, you will see improvements there.

In terms of features, we're probably going to roll more features in what we call Tabnine Assistant, which is not just the code completion, but all sorts of other insights about the code that you are writing. So Tabnine is definitely not limited to code completion. And our vision is actually to inject AI into all phases of the software development life cycle. And we are slowly but surely expanding to other points in this area including things that are outside the IDE. And you will see more of that early next year.

[00:42:54] KP: Well, very cool. I look forward to that. I see a lot of places in CI/CD where I'd love to see tools like yours show up. So excited to see what comes next.

[00:43:02] EY: Absolutely.

[00:43:04] KP: Eran, thank you so much for coming on Software Engineering Daily.

[00:43:07] EY: Thank you so much for having me. I had a good time. It was fun.

[END]

