EPISODE 378

[INTRODUCTION]

[0:00:00.5] JM: Reinforcement learning is a type of machine learning where a program leans how to take actions in an environment based on how that program has been rewarded for actions that it took in the past. When a program takes an action, it receives a reward for that action and it's likely to take that action again in the future because it was positively reinforced. It might also be negatively reinforced which might influence it to take fewer of those types of actions.

Michal Kempka is a computer scientist who works on ViZDoom; an AI research platform for machine learning. ViZDoom is based on the first-person dungeon game Doom. In ViZDoom an autonomous agent navigates through a maze avoiding enemies.

Reinforcement learning is a widely used tool for machine learning and we will be doing many more shows on it in the future to explain how it works in further detail, but Michal does give great explanation for how it works and it's a great example in ViZDoom. I also recommend checking out the paper that he wrote with his coauthors, which is in the show notes.

[SPONSOR MESSAGE]

[0:01:20.6] JM: Spring is a season of growth and change. Have you been thinking you'd be happier at a new job? If you're dreaming about a new job and have been waiting for the right time to make a move, go to hire.com/sedaily today. Hired makes finding work enjoyable. Hired uses an algorithmic job-matching tool in combination with a talent advocate who will walk you through the process of finding a better job. Maybe you want more flexible hours, or more money, or remote work.

Maybe you work at Zillow, or Squarespace, or Postmates, or some of the other top technology companies that are desperately looking for engineers on Hired. You and your skills are in high demand. You listen to a software engineering podcast in your spare time, so you're clearly

passionate about technology. Check out hired.com/sedaily to get a special offer for Software Engineering Daily listeners. A $600 signing bonus from Hired when you find that great job that gives you the respect and the salary that you deserve as a talented engineer. I love Hired because it puts you in charge.

Go to hired.com/sedaily, and thanks to Hired for being a continued long-running sponsor of Software Engineering Daily.

[INTERVIEW]

[0:02:50.9] JM: Michal Kempka is a computer scientist who works on ViZDoom; an AI research platform for reinforcement leaning. Michal, welcome to Software Engineering Daily.

[0:03:00.2] MK: Hello.

[0:03:01.1] JM: Reinforcement leaning is a type of machine learning where a program learns how to take actions in an environment based on how that program has been rewarded for actions that it took in the past. When a program takes an action, it receives a reward for that action and it's likely to take that action again in the future because it was positively reinforced. That's the definition of reinforcement learning. Could you give a more detailed explanation for reinforcement learning?

[0:03:30.9] MK: I'm not sure if there's more to say about the channel. Few of these, it's just like you said, the agent makes an action and gets the reward. How it learns and trains, it depends on the algorithm itself. Basically, it's just this framework when you get the reward after making the action. Talking about more details would just be talking about specific algorithms, or DQM or something else, Q learning. This definition you gave is just perfectly fine.

[0:04:11.8] JM: Okay. All right. Reinforcement learning is a subset of machine learning. Describe the difference between reinforcement learning and the more general set of machine learning techniques. Why is it that not every machine learning technique is reinforcement learning?

[0:04:28.0] MK: Okay. Basically, the most common divide or classification of machine learning algorithms is into supervised, unsupervised, and reinforcement learning which is kind of something strange. Unsupervised learning, the AI, and algorithms learns to see the patterns, like clusters. Supervised learning, it learns what we tell it to learn. Reinforcement learning, it's kind of makes of this and kind of strange thing on its own because the agent is not told or said what to do. It just gets the rewards. It must learn the policy, the things to do in order to learn the reward.

For example, in supervised learning, typical case, the agent would learn, for example, the reward. The reinforcement learning case, it just learns to act. It's something a bit different. We can actually use kind of supervised learning for reinforcement learning and also use unsupervised learning for making agents recognize some patterns and we can mix it there.

[0:05:52.3] JM: Right. This is what — I think something I understood was that you're often using supervised learning to develop a basis for how the model might take actions and then you use reinforcement learning to improve overtime in a way that's not like supervised learning. The supervised learning step, you're going to do the training data and its labeled data and then you make sure that the model is working somewhat properly and then you start to explore using reinforcement learning. Is that accurate?

[0:06:27.9] MK: Not so much, because it's all merged in one. There's the reinforcement learning algorithm which learns to act, and the supervised learning part is the part of, for example, optimize its expectations. The reinforcement learning, let's say, pure reinforcement learning part chooses the actions which are probably the best for him, and the supervised learning provides evaluations for these actions, for example.

The role of the reinforcement learning algorithm is to determine what actually is supposed to be fed into the supervised learning algorithm. For example, Q values, which say, "Which par state action is good?" basically.

[0:07:21.1] JM: We recently did a show on this program called Libratus, which beats humans at poker and uses reinforcement learning. Why are games a domain that reinforcement learning works well in?

[0:07:36.2] MK: Because you can repeat it all over. You have no cost of failure. You can just play, play and play. Actually, reinforcement learning would work in real life. For example, when training a helicopter or a robot, you should have lots of robots to learn because after each failure the machine would be potentially killed. Maybe answering the question in broader terms, reinforcement learning works in situations when you need to make actions in the environment, and you don't know the environment. There's lots of exploration you did in games. Like I said, they offer you infinite number of tries or as many as you like or as many as the time lets you — Yeah.

[0:08:35.9] JM: You're exploring the world in reinforcement learning. Does that mean that you need to model the world as a state machine so that the program can understand what sort of world it's exploring?

[0:08:49.9] MK: Yeah.

[0:08:52.8] JM: Okay. Libratus, for example, is poker bot. It made lots of mistakes early on. Basically, the way that they built this bot was they taught it the rules of poker and then it learned to classify certain decisions as mistakes or good decisions because it just based a decision off of the reward function of, "Did I make money off of this decision or not?" In order to keep learning, there is this temptation to do new things. In order to learn, you need to explore. You might end up losing money because of it. Although if it's just simulated money, it doesn't actually matter.

More generally, it takes time to explore. An exploration may end up at a dead end. You may just waste your time. You're always balancing this explore and exploit tradeoff. Explain explore versus exploit. Why is this a tradeoff that's fundamental to reinforcement learning?

[0:09:59.7] MK: Okay. While exploring, while gathering information, you may encounter states or actions which are very stupid, like hitting a wall, which normally you wouldn't like to do. Sometimes it could be something very beneficial which you'd like to repeat in the future. It's very important to explore to some extent not to constrain your options, like actions that could benefit you in the future.

Then on the other hand you could lose lots of time exploring. Lose lots of resources. That's why computer simulations are better in these terms than the real world. It's still computationally really bad, but you don't need to sacrifice resources. I'd say that, for example, evolution is very cruel and bad process of optimization on life beings when lots of animals, humans just get killed because they do bad stuff. Sometimes they do something clever and ingenious and then it's the good exploration.

If everyone explored, then possibly the whole race could go extinct, and races or agents which are too bad — Which explore to much can annihilate themselves. There's the tradeoff, that too much exploration can kill you or waste all your resources, and not exploring can make your actions safe but not very beneficial, or not very optimal, or poor, just to say.

[0:11:54.8] JM: How has reinforcement learning advanced in the past few years?

[0:11:59.1] MK: I'd say basically from doing not much to beating humans in lots of domains, like famous Atari, or AlphaGo, or driving cars, or robots who can walk maybe not as good as we can, or helicopters which can fly themselves past 10 years, maybe 5 years.

[0:12:27.0] JM: These are some dramatic improvement. How have these improvements been made?

[0:12:34.3] MK: I'd say that most probably it was the development of better hardware. I don't want to diminish virtues of people who designed the algorithms, but then there was not much point of doing this stuff 20 years ago when it would take years or just prohibitively long time to train an agent on anything reasonable.

It was done on some games, but it was not so popular. Now, basically, anyone with this interview can do it, and it's not very expensive. I would say that it's the key. Fortunately, it seems at least for now, that it's going to expand and thrive even more.

[0:13:25.6] JM: We've done several shows on deep learning, and deep learning uses models built out of layers of sequential processing units. How can deep learning and reinforcement learning be used together?

[0:13:39.8] MK: In reinforcement learning you often have to see some very contrived patterns and really complicated relationships between your data and your actions. Most often, deep nets are good at it. For example; vision, when playing Door, or Atari or any other game which is based on visual part. Actually, even Go, they used deep networks with convolutions there. These networks are really good at visual perception and they're very deep. It's very good for reinforcement learning agents to be able to see these patterns and create representation of the world. The more complicated the world gets, the more patterns it should be able to recognize.

[0:14:39.1] JM: Is it that deep learning tends to be good for building a model for how the world is, how the world appears, and reinforcement learning is good for improving the actions that you're going to take based on that model of the world?

[0:14:57.2] MK: I wouldn't necessarily split it and say that reinforcement learning starts here and deep models start there. It's all engrained. It's all in the networks. When the output comes, it's like our brain. It's just, let's say, very complicated function of our inputs, and the reinforcement learning just, if you will, gives us the way or to train or what should be in the output. I wouldn't say that it's starts or ends and it's for some purpose here. It's not very easy to distinguish where deep models and reinforcement learning ends.

[SPONSOR MESSAGE]

[0:15:55.6] JM: Simplify continuous delivery GoCD, the on-premise open-source continuous delivery tool by ThoughtWorks. With GoCD, you can easily model complex deployment

workflows using pipelines and you can visualize them end-to-end with its value stream map. You get complete visibility into and control of your company's deployments.

At gocd.io/sedaily, you can find out how to bring continuous delivery to your teams. Say goodbye to deployment panic and hello to consistent, predictable deliveries. Visit gocd.io/sedaily to learn more about GoCD. Commercial support and enterprise add-ons, including disaster recovery, are available.

Thank you to GoCD and thank you to ThoughtWorks. I'm a huge fan of ThoughtWorks and their products including GoCD, and we're fans of continuous delivery. Check out gocd.io/sedaily.

[INTERVIEW CONTINUED]

[0:16:56.4] JM: My understanding was that with these different layers in a deep neural network, some of the layers might be supervised learning, some might be reinforcement learning, some might be other strategies, and it's the sequential modeling. Is that accurate? Is it this sequential process, because you're describing it as more of an intermingled rather than a serialized process?

[0:17:23.6] MK: The reinforcement learning is, let's say, a framework. You can implement many algorithms that can be deep networks, then can be with or without memory. Basically, most of this sequential processing, it's done better with memory. Like I said, I would treat reinforcement learning as a framework and as well which we use to supervise learning and sequential processing, like LSTM networks. There are just some variations which remembers stuff.

[0:18:01.7] JM: I see. I guess this ties in with the fact that reinforcement learning can be used to tune their neural network. Explain what that means. What needs to be tuned in a neural network and how is the neural network being tuned by reinforcement learning?

[0:18:19.5] MK: A most common and most known algorithm called Q-learning is used by DQN, there is a magical value, Q-value, which is a function of the state and action and it tells you how much reward you'll get from this point on to the future making action from the state. The reinforcement learning part tells us what it should be, this Q, or at least some way of [inaudible

0:18:55.2] estimating it, and, per se, supervised learning part jut gets these values and minimizes the, for example, squared error or some form of error of its evaluation. Basically, the agent chooses the best action with the best Q in a given state. It's the most common setup.

This Q is based on reward. For example, having 10 actions when I get reward 1, 1, 1, 1, 1, 1, 1, and most — In the simplest scenario, let's say just the Q of the starting state, it's 10, because in the future it will get 10 rewards. I'd say it's the simplest form of using reinforcement learning here.

[0:19:47.3] JM: Reinforcement learning is often used in the back propagation step. For example, after you play a hand of poker, if you made a decision that led to you losing all of your money, you would want to have that lesson propagate back into the model. Explain how back propagation works.

[0:20:10.3] MK: For example, with this Q-learning, let's say we've just lost all our money, where reward zero and there's no chance that in the future we'll get reward because we just lost our money. Q-value of this state and any action is zero.

The network should now learn that the expected value of this state and action is zero. Then it compares with its original output of the last layer, which is based on all previous layers of the network. For example, if it's already estimating the action to be worth zero, there's no error. For example, if the action is very good for the agent, for the network, the error is huge, it's back propagated according to the gradient descent rule. I'm not sure if I should explain what a gradient is. It's just a derivative vector. I'm not sure if I should —

[0:21:19.2] JM: I know it would be useful to some people. It would be not useful to other people.

[0:21:24.0] MK: Basically, what it means is how much you should go in one direction, for example, on the X, Y plane to minimize some error is defined by some function. It's like square error. Many people know linear regression or something like that, it's minimized. It's like minimizing regression but it's not linear. There are some crazy function, and the gradient is computed by just deriving, and deriving, deriving, and each weight in the network is updated.

Basically, when you have the output layer, it's updated according to the gradient computed on the initial loss, the error function. For example, if we estimate one and its zero because we lost, the error is one. We compute the derivate and we've, for example, increase or decrease some weights.

Then, for back propagation step works in the next layer, each update is computed according to losses — The derivatives from the previous layer, and it works that way till you reach the first layer.

[0:22:48.5] JM: Okay, great. Let's go into an example, which is the project that you have worked on called ViZDoom. You published a paper last year about this, called ViZDoom; a Doom-based AI research platform for visual reinforcement learning. For those who don't know, Doom is a first-person shooter game. You navigate through this hellish world, you're shooting evil creatures. Explain what ViZDoom is and why you started working on it.

[0:23:17.6] MK: First, maybe what was the idea, why and when. It was the time when deep minds was — It was sometime after they published DQN and was quite famous. It was after the nature publication about DQN, and they used it on Atari which is very simple and crude for humans. It's 2D. Our supervisor got the idea to create an environment which is similar, but works on some 3D shooter, which is more like real world. Not the shooting part so much, but moving in 3D space. It's more like a robot simulator, very crude. Well, it depends on the engine.

We have started — By we, I mean my colleagues, which with whom I've made ViZDoom, in [inaudible 0:24:15.4]. So we decided to use Doom, or actually ZDoom, it's part of Doom, because it was complicated and sophisticated and real-life enough to be good for 3D simulations and very light weight. You could run Doom on the bar on Mac or MP3 player. It's really not very hungry for resources. It's very good for lots of simulations, lots of tries, and lots of monsters. It's perfect for simulating and learning, because you can simulate lots of episodes, lots of scenarios. We also wanted to create a way for users to create their own scenarios, not only playing Doom itself, but task to shoot, task to move, task to solve a maze and more — Basically, imagine is your limit, or maybe imagination and parts of Doom engine. It's very robust and, for sure, it should be enough for most of reinforcement learning applications today.

[0:25:34.6] JM: Right. We spent the first half of the conversation talking about reinforcement learning and back propagation and Q-learning and supervised learning. We talked about all of these machine learning concepts. Now, we're taking those and applying them to Doom, this first-person shooter. You've got the character who is navigating around. You don't want them bumping into walls. You want them to be able to defend — You want the agent to be able to defend itself from these monsters that are attacking it. Explain how you created an artificial intelligence that can play Doom, and describe how you used some of these machine learning concepts.

[0:26:20.1] MK: In most truth scenarios, it's enough to use the algorithm that I just mentioned; DQN, which was in Atari. In many scenarios, it just works out of the box, like the most simple scenario which we call basic when you can move left or right and shoot, and you need to shoot the monster as soon as you can. It works perfectly. It works perfectly, even with some bit more sophisticated scenarios, like we call it Pac-Man because it's basically a Pac-Man map. The game of this yellow guy — It's dots and could be killed by ghosts. We have a scenario which is basically Pac-Man, but it's 3D. It also works very well. It can shoot monsters and can also catch a power pill and everything turns white and he can kill the monsters then. It's not very complicated. It's just using DQN out of the box to work in these scenarios.

For more sophisticated stuff, like playing Doom, shooting to real people and computing in a tournament or something, that there's a lot of tinkering needed like, for example, shaping. It's a similar concept like in working with animals. You give them smaller rewards for making a small improvements or making just little steps and good directions and it should be easier for them to grasp the whole concept. For example, it's very difficult for the agent, not for the dog — Well, maybe for the dog also, to know that you need to pick up a key and go to doors and close the door because it's very long sequence. You could use shaping to have the agent thing. Yeah, shaping is very good, but requires lots of tinkering.

For example, in our competition which we organized, most people actually used shaping because it was not enough for the agent to receive only rewards for killing. I'd say it were auxiliary rewards for picking up med kits for moving in some particular way and so on.

[SPONSOR MESSAGE]

[0:29:09.5] JM: Hosting this podcast is my full-time job, but I love to build software. I'm constantly writing down ideas for products; the user experience designs, the software architecture, and even the pricing models. Of course, someone needs to write the actual code for these products that I think about. For building and scaling my software products I use Toptal.

Toptal is the best place to find reasonably priced, extremely talented software engineers to build your projects from scratch or to skill your workforce. Get started today and receive a free pair of Apple AirPods after your first 20 hours of work by signing up at toptal.com/sedaily. There is none of the frustration of interviewing freelancers who aren't suitable for the job. 90% of Toptal clients hire the first expert that they're introduced to. The average time to getting matched with the top developer is less than 24 hours, so you can get your project started quickly. Go to toptal.com/sedaily to start working with an engineer who can build your project or who can add to the workforce of the company that you work at.

I've used Toptal to build three separate engineering projects and I genuinely love the product. Also, as a special offer to Software Engineering Daily listeners, Toptal will be sending out a pair of Apple AirPods to everyone who signs up through our link and engages in a minimum of 20 work hours. To check it out and start putting your ideas into action, go to toptal.com/sedaily.

If you're an engineer looking for freelance work, I also recommend Toptal. All of the developers I've worked with have been very happy with the platform. Thanks to Toptal for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:31:13.8] JM: You described earlier this term Q-learning. Explain what Q-learning — Give us a refresher. I know you explained it earlier. It's this model freeform of reinforcement learning. Explain in more detail what Q-learning is and how that applies to building your Doom bot.

[0:31:31.5] MK: Okay. You have multiple states and multiple available actions, and Q-value —

[0:31:39.3] JM: That's true for any game you could play. You can model any game as just states and actions.

[0:31:46.9] MK: Yeah. I'm not sure if we shouldn't constrain states to some finite realms or something like that, but yeah, basically you could somewhat — I'm not sure. You should have a finite number of actions and finite number of states, especially finite number of actions, because otherwise it would computationally be invisible for Q-learning.

[0:32:13.5] JM: Anyway, sorry to interrupt. Go ahead.

[0:32:16.1] MK: You have this Q-value which tells you how much reward you will get now and in the future by making action A in state S, let's say. Rewards could be discounted and it means that future rewards count less than the immediate rewards. For example, if I was to collect dots and I'd get 1 point for each dot, after collecting one dot I get a reward one. One is my immediate reward, but I estimate that in the future I will get 100 dots. Let's say that my discount is 0.99. The second dot would count as 0.99, and the third dot would count as 0.99 squared and so on. This Q-value represents how much reward we'll get.

Q-learning, in its most popular formulation, estimates this by adding R to the estimate of the Q-value of the next state which we got. For example, I'm in state S1. I make action A and get to state S2. My Q-value would be for S1 and action I just took would be the reward I just got and maximum estimated Q-value of this state I just got into. I'm not sure if it's clear. It's really better to show this stuff with equations, because then it gets more clear, but it's just —

[0:34:18.7] JM: Okay. That's okay. Let's talk in a little more high level. In this experimental part of your paper, you talk about the environment, testing the environment by trying to learn bots for two different scenarios. One is this basic move and shoot task, and the other one is the more complex maze navigation problem. Describe these two tasks and you trained the bot to accomplish these.

[0:34:50.0] MK: With basic, I just said, it was just plain DQN. I'm not sure if I should explain it again, the algorithm, but there was not much to do. It was just plain DQN. As for the second scenario, which was about moving in a toxic environment, so each move — Maybe not each move, but periodically you get hurt and you need to pick up med kits to heal yourself. It's very tricky, because the agent gets the reward every time it's able to survive. For each step, it gets one point, and it gets minus 100 points for getting killed. It's very hard for the agent to notice that what makes him live longer and get more rewards is picking up the med kits. First way to solve this was shaping. Like I said before, just giving the agent bonus points for picking up med kits, and then it works like charm.

It's not very elegant, because there are many environments in which we cannot tinker with rewards. We don't know really what we should do. Agent is supposed to learn it. Actually, we came up with some other ways, but it's lots of technical stuff so I'm not sure if I should explain it. Very little changes to DQN which are not — They're quite technical. There's no magical high level stuff which I could explain about it.

[0:36:45.0] JM: Can you explain the software packages that you were using, because I'm sure there are people out there who they're interested in machine learning or deep learning and they want to get started by simulating games. Just give a workflow for what somebody who wants to make a bot for a game. They want to do machine learning, or reinforcement learning for a game. What are the different software packages they should use, machine learning frameworks? Give a rundown of what you used.

[0:37:18.4] MK: First of all, it's not very easy to start this because you need the engine itself to support you. What I mean, for example, Blizzard is not very keen on making bots. You probably won't have — Recently, DeepMind, probably something about Starcraft II. Generally, most of publishers of games do not allow you to get API and get into the workflow of the game. It's the first step, to find the game which supports this. For example, API and C, or Python, or whatever.

As for making bots itself, I would just recommend Python and TensorFlow, of Theano, or PyTorch, [inaudible 0:38:12.5]. I don't use it, but lots of people say it's okay, although I'm not sure if it's good for reinforcement learning, or lots of games I feel — I'm not sure, use Lua

inside. Maybe Lua and Torch, which is promoted by Facebook and developed by people from Facebook I think would be good. These are all good frameworks and there's not much about the workflow. You just need Lua or Python and the framework itself and you can do the stuff, and GPU, but it's not software, and not Windows preferably.

[0:38:57.9] JM: Okay. The DeepMind team had success teaching AI to play Atari games and they also had success with AlphaGo. How did those machine learning approaches compared to the approaches you took with Doom bot?

[0:39:15.6] MK: Yeah, we're heavily inspired by DQN and the algorithms that were developed to solve Atari. Most part it was just the same algorithm. At least DQN, because, one, playing Go was something magical, something else maybe. I mean it was a lot bigger than Atari and we wouldn't be able to do it on our home PCs.

[0:39:48.6] JM: Okay. Cool. I guess to begin to wrap up, what are the potential upgrades in the future that you're working on for ViZDoom or improvements that you could make?

[0:39:59.8] MK: We don't feel like ViZDoom itself needs more features, like extracting more information from it, but there are suggestions maybe to add sound buffer, though we don't like the idea because it's very painful. Possibly porting it to other Doom engine which, for example, supports true 3D, because I'm not sure if you know and our listeners, Doom is not real 3D. It's like 2-1/2.

There are parts of doom which do support OpenGL and real 3D graphics and it's potentially prettier. What's more, the port we're using now, it's just not being supported from yesterday. I'm not sure, but my friend told me this. It wouldn't support it anymore, at least by the main authors.

[0:40:59.1] JM: Okay. Last question; what are the domains that you think you could see reinforcement leaning be applied to in the near future that haven't been as much as you think they should be approached.

[0:41:13.0] MK: I'm sure that there are lots of domains, but I'm not sure they haven't been approached. I bet that most of tasks which require decision making and quasi real, or real-time where it solves to buy me software reinforcement learning. I'm rather sure of it, like robotic stuff, driving, any movement stuff, tracking, anything. Yeah. I seriously doubt that there's a task that wasn't solved by reinforcement learning which deserves reinforcement leaning. For example, recently, I got to know that even some of our Polish — the facility where you clean waste, uses DQN to solve their optimization to steer some pumps or some heating to clear water. If DQN and reinforcement learning is used in such place, I'm not really sure that there's any area it is not used.

[0:42:32.2] JM: Okay.

[0:42:32.3] MK: At least like who can approach stuff from the IT point of view.

[0:42:38.5] JM: Okay. All right. Mikal, thanks for coming on Software Engineering Daily. I'm excited to see your results from ViZDoom and I look forward to seeing your work in the future.

[0:42:48.5] MK: Thank you.

[0:42:48.8] JM: All right. Thank you.

[0:42:50.6] MK: One more thing.

[0:42:52.4] JM: Sure.

[0:42:52.6] MK: Could you add some mention about other authors of ViZDoom and what description you get or you write.

[0:43:02.4] JM: Sure. I will put those in the credits. I'll put a link to the paper and I will attribute the other authors.

[0:43:11.4] MK: Thank you.

[0:43:13.0] JM: All right. Thanks, Mikal.

[0:43:14.7] MK: Thank you.

[0:43:14.9] JM: Okay. See you later.

[END OF EPISODE]

[0:43:21.1] JM: At Software Engineering Daily, we need to keep our metrics reliable. If a botnet started listening to all of our episodes and we had nothing to stop it, our statistics would be corrupted. We would have no way to know whether a listen came from a bot, or from a real user. That's why we use Encapsula to stop attackers and improve performance.

When a listener makes a request to play an episode of Software Engineering Daily, Encapsula checks that request before it reaches our servers and filters bot traffic preventing it from ever reaching us. Botnets and DDoS are not just a threat to podcasts. They can impact your application too. Encapsula can protect your API servers and your microservices from responding to unwanted requests.

To try Encapsula for yourself, got to encapsula.com/sedaily and get a month of Encapsula for free. Encapsula's API gives you control over the security and performance of your application. Whether you have a complex microservices architecture, or a WordPress site, like Software Engineering Daily.

Encapsula has a global network of over 30 data centers that optimize routing and cacher content. The same network of data centers that is filtering your content for attackers is operating as a CDN and speeding up your application.

To try Encapsula today, go to encapsula.com/sedaily and check it out. Thanks again Encapsula.

[END]