Using Claude Code with Eval Tools

31 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=ASDJSLVTKBQ
https://www.youtube.com/watch?v=AsDjsLvtkbQ

SUMMARY

In this session, the capabilities of coding agents, particularly in the context of software evaluation and debugging, were explored using the open-source tool Phoenix. The discussion highlighted both the potential and limitations of automation in coding, emphasizing the need for human oversight and the importance of structured data for effective coding agent performance.

- *Coding agents can write code, run experiments, and debug software, but they still require human judgment for effective evaluation.*
- *The open-source tool Phoenix facilitates the integration of coding agents with observability platforms, allowing for better data analysis and experimentation.*
- *The introduction of models like Opus 4.5 has shifted the coding landscape, enabling more live coding and reducing reliance on manual coding practices.*
- *Coding agents need access to telemetry and traces to understand and improve their coding processes, highlighting the importance of structured data.*
- *The role of human developers is evolving towards validation, review, and harness building for coding agents, rather than direct coding.*
- *Skills and tools are essential for enhancing the capabilities of coding agents, but the quality and reliability of these skills can vary significantly.*
- *Continuous evaluation and improvement of coding agents are necessary to ensure they adhere to best practices and do not replicate existing codebase flaws.*
- *The future of coding agents involves a balance between automation and human oversight, with an emphasis on preparing codebases for effective integration with these agents.*

Coding agents can now write code, run experiments, and debug software. So, can they do your eval's for you? In this session, we put that question to the test live. We give Claude code the full eval life cycle, pulling traces, error analysis, and designing experiments. Some of it works surprisingly well, some of it doesn't. You'll see exactly where automation helps and where human judgment still matters. Mikio's the lead developer of my favorite open-source eval tool. It's called Phoenix. Phoenix arises, you can Google it. It's great. It's really easy to use, self-host, has really good APIs, really good documentation.

And as coding has become a lot easier with models or with coding agents, especially since November, December with the new Opus 4.5 and onwards, yeah, that is a new territory. I don't think you can just hand over your entire application to an LLM and just say eval it. But, I think there's a lot of things that you got to instrument your code, you got to get it into Phoenix. And there's like a lot of little things around even looking at the data, filtering it, navigating it that you might want to think about. And then since Phoenix has all these APIs, I think the way that Mikio has engineered it is that it's very agent-friendly. Like you can connect it with your agent.

And so I saw that Mikio posted that Phoenix now has support. He's been working on sort of making it really good with agents. And so I think a lot of the stuff that was more manual before can now be automated.

And so I'm really excited to talk about and show how to use coding agents with Phoenix. And I'm really excited to see what Mikio shows us today. You know, that I think was been weighing on my mind and probably weighing on Hamel's mind and a bunch of your minds too, hopefully. As Hamel alluded to, coding agents are really in production now. Specifically, last December is the big turning point with Opus coming out. But, and even hardened practitioners like Andrej Karpathy are saying that before December of 2025, they were, you know, coding maybe 20% {quote} unquote live coding and 80% using tab complete and things like that. But, that completely inverted, right? He went completely to 80% live coding and 20% manual coding.

And I think the conclusion that he came to during that was spot-on, which is that right now we're at this interesting point where inference, sorry, the intelligence of these coding agents is a little bit ahead of the infrastructure. And today I want to explore that a little bit and what it might look like to have self-evolving software. I think the left-hand side we all know. If you've been If you're coding in the 2010s, you're familiar with deploying an application. Maybe you have dog monitoring or Grafana dashboards set up.

And then you as a human dev, something goes wrong, you get paged. You You look at that glass pane and you look at the dashboards and you come up with a hypothesis of what to change. Then you make a change in the IDE, write some unit tests, and deploy that. And we're in this middle phase, right? Where you deploy an application, you probably do have an observability platform, maybe hopefully Arise Phoenix. And then you as a human dev look at that screen and figure out what might be going wrong.

And then maybe now you use Cursor or Claude code to come up with a solution, deploy it, and then hopefully you can see the changes in the observability platform. And I think we're still going to be in this phase for some time, but let's put on our like futurist uh hats on for a second and think about the third phase, which is what if coding agents do all the coding, right? Then you're still going to have an application and you still need an observability observability platform because really the agent needs to see the same things that the human was seeing previously, but instead of using uh glass pane with dashboards and heat maps and things like that, it needs things that are, you know, efficient for it to pull things. And I think an agent's eyes are very different than a human's eyes, right? It is really good at parsing out structured data, putting them into files, using grep, using search tools, things of that nature. And so we really need to let coding agents gain access to observability platforms so they can iterate and validate what it's doing and so it can deploy the application. Now, the like the dashboards and stuff still important because human oversight is obviously really important, but it doesn't become the primary use of something like an observability platform. I think there's so much of like just human way of thinking and stuff that has to shift. I think the hard thing is like each individual has like this tribal how you work. I mean, we all need to change.

There's certain arguments about maybe previous the focus was more important, but now parallelization is important. I think obviously the human role in all of this becomes we shift more into the validation and reviewing and ultimately the harness building, the instructions that these agents will because the more you I

think the OpenAI paper also particularly talks about this. It's really about constraining these agents. So, where I think this new what they call agentic engineering groups are going to start forming to make coding agents have a little bit more autonomy.

I don't know if that like fully answers your question, but >> Yeah, I think so. You didn't talk about the harness engineering paper, I think, right? Or is there something like that? Okay. Yeah. I think the so that that picture is just talking about like how much live coding is out there. But, I think what we have to acknowledge also that there's actually two things shifting at once. One is that the way in which code is being shifted it's shifted is changing, but the other thing that's changing is the types of applications, right? So, like on the far left, the reason why, you know, a lot of that stuff worked the way it did is because to some extent there was a level of understanding just through the code. And this is why coding agents are so good is if it has visibility into the code, it knows how that code executes, right?

Through types, through if statements, through while loops, all those kinds of thing. It can create like this state machine and get get an understanding of how your code executes. So, previously, if it was just code, the code documented the execution. But, the thing is now like we don't We're not just uh deploying one type of code, right? We're deploying code, we're deploying natural language prompts, and it also is reliant on the weights. And the way that the code looks is not necessarily how the agent operates, right? The agent might loop, the agent depending on which model it's using might loop three times versus four times, all these kinds of things. If you think about it, to some extent, agent behavior is more documented by the telemetry and the traces than it is with code. So, if agents are going to be coding agents are basically going to have to write code to to steer agents in one direction or another, they actually have to have access to the traces.

Just to reiterate what a coding agent is. Like a coding agent is not just made up of It's not just Opus 4.5, right? It's all the harness that that surrounds Claude code. That's things like skills, which we'll talk about today. Maybe you have an MCP server connecting to documentation. Maybe you have agent browser set up, tools. Maybe it has to have access to things like the Phoenix CLI. You know, if you're using Cursor, it has access to the editor and the LSPs, the prompts that are inside of the harness, and its own memory management system.

So, in order for coding agents to really drive in your code base, you need to tune this harness for your particular use case. And the two things we're going to talk about particularly are skills and tools. Okay. So, this is what we're going to do today is like the idea of a coding agent having access to its own telemetry. And one of the cool things about Phoenix is you can basically give it a Phoenix or coding agent you want. You can have one per workspace. But, we're going to have an agentic application. It's going to be streaming its traces, its evaluations, its experimentation, its feedback all into a local Arise Phoenix instance. And then we're going to create a CLI that's basically connected to that Arise Phoenix. And it's going to be able to pull traces down, log them to files so that you can use things like grep and jq. And then the coding agent won't have like eyes into what it did or what is happening and be able to come up with proposed changes, make changes to the code.

That telemetry will then just continuously flow back into that observability stack. And the coding agent will be able to make adjustments. And this is exactly what the harness engineering that is being done inside of OpenAI

as well. So, today we're going to try to make a coding agent take the AI engineer loop. We're going to go and tell the agent to look at its data via open codes, a taxonomy. And then we're going to skip the annotate the data just for time's sake, but we're going to get it to base off of those codings, hypothesize why the output is bad. We're then we're also going to take like its hypothesis and then let it plan out an experiment and design out like the evaluation criteria that's going to make it measure its outcomes. We're going to force it to measure its outcomes. And then we're going to force it to apply its changes and explain its reasoning. So, that's what we're going to do today. Let's jump right into it.

So, here I have an agent on the right-hand side and I have the Claude code on the side and they're wired together like I just showed you. This is a agent that's connected to MCP with Phoenix. So, let's do the first step of the AI engineering loop just for sake of trying to make this coding agent take my job. Okay. So, I'm going to clear the context just to prove something. And then I'm going to explicitly pull the the Phoenix eval skill just to or say this is an interesting topic we can delve into, but I'm going to paste this. And I'm going to force it to do axial coding to to try to understand what's happening currently in the traces that are already in my Phoenix. And I did seed it with a bunch of questions. Okay.

You're going to do open coding and axial coding with the agent? Yeah, so the way that it works and I'm going to let this rip. You can see that it's pulling down the traces with the Phoenix CLI. So it's pulling down that data and then it's logging it to files and then it's going through each one and doing the open codes and it's going to go through that open code step. And the reason why like I had to prompt it very little is because I actually mounted the the skill there and happy to just show what that skill looks like. So it's worth it's worth lingering on this point for a bit, I think.

>> Yeah. Do you think this is a good idea to just have an LLM do your open code axial code for you without looking at it yourself? Like what's I know like this demo is for a specific purpose, but we should get that nuance. Okay, well. So I think while this is ripping, we should probably do the open coding ourselves and see what the difference is, right? Because I think you're right like you're the taste maker and a lot of the open coding and stuff is about you looking at the data and I think what you're going to notice and it will be an interesting nuance here is that there's a difference between a human doing in the open coding and axial coding than what an LLM does.

That doesn't mean I think if you put yeah, a lot of your human taste making into skills, it couldn't do a lot of interesting things. Whether or not you fully trust that process or not, I think is obviously up to cuz like how much are you letting it rip? I'm not here to say I think it really maybe depends. I'll maybe show some examples and that'll be more clear and we'll see some examples of that. I think it's agents give you the ability to get like a second opinion about stuff too and there's a lot of times that can be good. So maybe you should like not look at what it's doing and then you let it rip and you also look at what it does and it might actually discover things that you hadn't thought about because I think you know, if I did open coding and then you did open coding and axial coding, we might come up with a different coding strategy and what you'll find is that definitely the agent comes up with a different coding strategy than I do. To to your I think the interesting debate now really is going to be is the agent going to have taste? And to some extent, it's not there yet, but if you look at what OpenAI is doing, you can see within their harness they basically have created that taste making inside of here about how

they think design should work, how they think the front end should work. How how it should think about user onboarding and things like that. So they've basically encoded that in the code base to let the And I think the thing that you're going to see here is that this code base this is not mature, right? This code base is one that wasn't designed with this harness in mind. And so it's going to do a lot of things wrong and I think OpenAI also claimed to do that that happened, right? Like the basically the getting started is really tough because the agent's going to go off and come up come with wrong conclusions. I'm doing this more as a straw man exercise to see like how much of this it can be possible, not necessarily like advertising that like this is what you should do. That it went in and said there's a lot of things that are off topic in the two out of the 15 traces, which is 13%.

So it came up with a bunch of coding and it's citing the different traces of what it's doing here. So I'm going to prompt it one more time to just come up with a hypothesis and we'll see how well it does. So I mean, okay, let's do let's just we can also do the same exercise, right? We can come in here, we can look at each one of these and do the open codes like how do I handle file uploads in Phoenix LiveView?

Okay, so it's talking about it's a little bit of a fun exercise. It created its taxonomy in its own memory system right now, but you could definitely have it sync it back. Those are all the kinds of things that that you could consider. Do how much do you want agent generated notes in your telemetry and it's obviously So let's clear the context here. Just let it not bias itself to its previous axial coding and let's also bring in the Phoenix Evals skill again. So we'll still look at the system prompt. So it's saying Phoenix style assistant prompt lacks grounded context about the topic Phoenix Elixir Phoenix at chess. Are you the scientific experiment to validate the root cause step one or state a single falsifiable feat like basically grounding it and giving it the process it should go by to build out this experiment and validate results and loop on on that a few runs to validate it and for it to take a prove me right or prove me wrong through Evals approach and we're going to let it rip as it does that. So an experiment is similar to the scientific method experiment. So you have your know, your controlled variables. In this case, I'm saying the controlled variables keep the model the same, but change the system prompt of the agent to to see if we can steer it to have a better better grounding in in the topic that it's supposed to be talking about. We're reading the system prompt as the uncontrolled variable, then we basically can run Evals on the agent before the system prompt is changed and then Evals after the system prompt is changed so that we can basically prove that that change in the system prompt had like a marked improvement in the agent behavior itself. And it's using some of the data sets that are already there in Phoenix.

I have the evaluate the evaluators here. I have some data set fixtures. I have the evaluators which are like I have this eval called terminal safe because like we're outputting to the command line. I wanted you to it to use ANSI characters rather than markdown. All right, so I have these evals and stuff and then I have these experiments which is does it conform to the terminal format that I want and we're hoping that because I, you know, created enough of an eval harness in here, the agent can discover the best practices that I've put in here for it to create an experiment. It's going to come up with a experiment to say because basically we said the system prompt is infecting how it's not grounded. So it needs to come up with the data set that it's going to run that over. So the examples and the splits that that it needs to run on and then it needs to create the right evaluation criteria. So we put that a little bit in into the prompt itself, but it's also in there, which is after each run record.

So I said data set create defined fixed set of Phoenix evaluators measuring your success criteria. So it's actually just created an evaluator. So again, not advertising, but this is just I'm just giving you a peek into what could be coming down the line, right? Is coding agent's going to write Evals. We'll see how well it does, obviously. So you can see that it's created an eval, right? So let's just look at it. I mean, it it should be here soon. Already, so I can look at it right now. So created this new evaluator. It's called topic handling. It's using Haiku. You are evaluating whether or not an AI assistant correctly handled the user query.

The assistant is a Rise Phoenix assistant and AI observability platform handled correctly. Query is a Rise Phoenix AI observability handles incorrectly and So it's giving it some few shot examples of what it thinks is good and what is not bad. Obviously, we shouldn't if building evaluators like this requires benchmarking, which we can do also if we have some time. So we can look at the experiments that it's formed. How about that? Well, so let's look at the experiments. So it's run this ex it's created this data data set called topic hypothesis experiment V1.

And then you can see here that it's created an eval called topic handling and it thought it handled the topic at 0.93 and then it's run an experiment three times to try to change the system prompt. And so in this experiment, maybe we can compare the two. Let's compare just two. So it's terminal output is a little bit hard to see. Sometimes it constructs experiments in different ways. Sometimes it does it in the experiment itself. Sometimes it makes the changes. So I think depending on how you prompt it, it will either make changes and then revert it and do it like manually or it will do it within the experiment itself. It might be in the experiment itself.

Again, this is what happens when you let it rip. So yeah, so it has basically put the prompt inside the experiment and it's run two different set of instructions in here. So Unfortunately, it didn't change the agent itself. It just changed the prompt inside of the experiment itself and that's just one of those things of like coding agents figure out a way. Let's let it rip on one last thing and then we can just talk over it about all the interesting things we just saw.

So I'm going to let it rip one more time. I'm just going to tell it like evaluate the evaluator basically. So I'm going to say benchmark the new evaluators you created and show the confusion matrix of its accuracy. Okay, so we're just going to try to force it to do some best practices about creating a resource data set for the evaluator and for it to tune it a little bit before we we accept it. Yeah, to your point, I think I definitely recommend everybody read the OpenAI we're not admitting is like we think that there's these repository of skills that are just going to magically make these agents do things. I think it's very different. If you think about like the way that the Claude code team for example keeps track of their a Claude MD and the skills and the hooks and all that stuff. All these kinds of for lack of a better word like restrictions to keep the coding agent on track, that's an iterative process. And so a lot of the even Claude coding supports this now where you can say, looking back at your current session, how would you adjust your skills? How would you change your instruction set to to better do not make the same mistakes you made during this session and they commit that back and then what the Claude code team claims to do is they take that prompt and then they put it through the optimizers that they have inside. I think it is available in Anthropic desktop app optimization, but they basically periodically renormalize the skills. But they the skills are a self-evolving thing. Every time the agent makes a mistake, you basically say, oh, you did

something wrong, put that in your your system instructions in your agent MD or in your skills and make sure that you don't make that mistake again. So I think that's the the harness if you will is like the self-evolving.

The determinism aspect I think is another thing that is talked about a lot in the opening eye paper and obviously if you use cloud there's hooks, right? So there's hooks that basically enforce very specific things. So they're like pre-commit hooks like stuff like forcing the agent to follow very specific patterns with determinism not just through prompts. And the skill spec, you know, that everybody's converging on also has code snippets they can run as part of the skill. How likely is like an agent be able to launch a solution? And that I think is both good skills in the skills but also best practices in your code because I don't know if this is your experience, but once the code base base is mature agents actually do a lot better or a lot worse depending. If your code base has a lot of bad practices in it, it's probably just going to copy those existing patterns and start replicating them over and over because that's what it sees in the files and it thinks that the files is the way that it should be. So you have to mature both of them um at the same time because otherwise the agent's going to start just replicating bad practices and taking the shortest path to victory. And so it's actually like the URL's incorrect for example. And I can tell that as a person that's like basically trying to build this agent that the links that it's generating it's directly taking the link from melt my find just regurgitating those links. But those that that those links actually need to be translated into something that is satisfactory for me. So that's like an example of there's absolutely no way an agent without me telling it that is a problem. If you encode best practices, it'll try to do some interesting things like pre- produce a confusion matrix and try to make these evaluators do things.

So at least like it'll scaffold the benchmarking for you and then you can go back and add examples in which maybe or maybe swap out the data set. But it did create I think a useful evaluator as part of this practice. Like at least now I know that I need to at least put something about that there's a huge name collision with the word Phoenix. Like Phoenix is a very generic word. Right? If I'm me personally I'm at the mercy of the is Phoenix Arizona, there's Phoenix Elixir, there's Phoenix Phoenix LiveView, there's the Phoenix the old email client, right? And so I need to at least put that. If you really wanted to let it rip, you would say make these changes. I think this time time around it didn't produce something as interesting as I was hoping. I've had honestly it produce some really interesting results on the same on the same code base. So I think that's another lesson learned for me too is like these harnesses sometimes it'll look at the skill, sometimes it won't.

It's really hard for me to determine when and why it will will do things. I'll give you an example like last night that I was like, "Oh my gosh, it did something really amazing." Which is it actually started looking at the number of repetitive tool calls that was being made for MCP and forcing actually for the system prompt to to reflect on the topic and optimize the turns of the number of times it should look at tools.

And it actually created some pretty novel evaluators for me that actually I thought were quite interesting and it actually let it rip for much longer and it came to a really pretty insightful because what you see right now like one of the problems here is okay, even though this is about Phoenix LiveView, it's hitting my MCP server three times. Like why would it even pull in context from my Phoenix MCP server to get do perform rag on Phoenix documentation when it the onset already should know that this is basically off topic and it shouldn't have to make the tool calls.

So that was an example of it really understood the topology from the traces and was able to identify something that I could have come up with that conclusion, but I just let an agent rip and it figured out that I should be optimizing the number of tool calls specifically when it was on topic. Let's say these coding agents are making a lot of changes on your system. They need to basically provide evidence that they have made things better or worse, right?

And now the burden on me is to review this stuff. And if I have the agent connected to to come up my eval harness as well as my observability platform, this is basically providing at least to some degree the evidence that it's improved the system. Now we can obviously question the validity of the data and the validity of the eval itself. Those are all things that are can be should be scrutinized, but those can be scrutinized obviously during the pull request. What the agent has currently basically said that it's done is it took the you know, it ran it with the first set of prompt. It was currently it was handling whether or not it was on topic correctly at a 0.93 rate and then it's claiming that it it it accomplished like a perfect score on keeping the the CLI tool on topic. There's a lot of things that I sort of contrived here, right?

We're talking about 15 examples. We also made it do axial coding on 50 15 examples. So that's another prob- issue too is like I didn't really have time to let it rip on much larger data, which maybe it would have figured out a lot more interesting problems um looking at more data. The skills are available. So skills are the distribution of skills is super early right now. They can be distributed in a lot of ways. If you want to just try out these skills, you can do MPX add skills from our repository. And then basically what this does is it pulls our skills that we are been honing over the last ones. So the one you saw today is of teaching the agent how to pull down traces, introspect experiments, stuff like that is Phoenix CLI. That one I definitely recommend it. That one's like non pretty much non-controversial, right?

That's just the treat. For example, the Phoenix CLI is not in the pre-training of Opus 46. So you might need this one to basically let fog code be able to understand how to use the CLI. Phoenix evals is like combination. So these are different composable pieces that we're hoping like you use them or don't maybe you don't, maybe you hard fork them, maybe you know what you do with them is obviously up to you. The all of these are basically registered. Currently Vercel's distributing some of these through skill.sh.

Some people put them on their website in the well-known domain. So there's a lot of different ways that they're currently being distributed. Word of caution, there's a lot of bad skills out there. So just because there's a lot of downloads of a skill doesn't mean it's better. And I think the other thing about skills, be careful of the skills that like loop and go search the internet for more skills cuz it's def- just because you can read the skill in markdown doesn't mean that there isn't malicious HTML in there telling you to do bad things. So there's definitely prompt injection in there. I mean the great thing about this is maybe you were previously you should always be reviewing the prompts. And the good thing about the prompts is that they're really things that you can diff. They're things that you can hook into CI pipelines. You can run a regression suite using the experimentation. So basically a CI step you can validate that these new system instructions that it just committed, right? Are in fact better than the previous one. So this is where like the continuous eval process comes into play, right? Because you now need we only ran it on 15 examples. It's pretty important to run this prompt through a lot more

examples and make sure that it's hardened against other criteria that you might have like properly out-outputting to the terminal or properly resolving links from your MCP server. Whatever those other evaluation criteria is, it needs to prove itself as well. Even the console logs are being logged as markdown so LLMs can interpret that stuff. There's just so much happening right now to give coding agents more access to this stuff.

I think it really it the call to action right now is like getting your code bases ready and have the right opinions so that you know when people come on board with coding agents that you really can scale out your productivity using coding agents when appropriate. Obviously when when appropriate is like the big question right now, but there are definitely areas where things are a and I have best practices of how I might my UI should look. Like yeah, let coding agent rip. If I'm trying to change the system prompt of a very critical agentic app, should I let my coding agent go rip on it?

Probably not, but I should have guardrails through evals that if it does make a change for whatever reason it knows that it did something wrong. So those are the kinds of signals that we need to connect coding agents to more of these signals. And the infrastructure needs to evolve, but you need to connect your infrastructure to your coding agents too. There's the Andrej Karpathy prediction that 2026 the tail end will be the slop Gopolex. I don't know what how to say it, but like the slop like there's going to be a lot of slop that we're going to have to reconcile with.

But then there's also like the open claw stuff. Like has massive security problems, but he probably just went and got hired by OpenAI for you know, 100 million dollars and produced some amazing amount of value. Yeah, has a lot of security problems, but he created a very interesting product of a self-coding agent that provides a huge amount of value. I don't want to maybe say don't Obviously I think our role as coding is going to diminish.

You better get on board to some extent just because you're going to be out-competed by the companies that are doing that. But the level of tolerance of what's good and bad the taste making still is like in the engineers' jobs, the PM's jobs, the MLEs, right? That still is like hugely important. Let's just be honest. Like these coding agents have no taste. These coding agents don't know what best practices are. There's a lot of things that they don't know. And skills I think is the first step to that, but let's be honest codes like skills don't fire. What is the Vercel benchmark? It's like they fire 52% of the time right now.

>> so they're unreliable, >> I'm always naming my skills. Like I even tell it to use the agent browser for testing like manually to make sure it fires. I agree with that and I agree with that. I think you should definitely be preparing like to your earlier point like you should prepare your code base if you want to do evals and measure things, you should be distilling your understanding of evals and how they fit into your domain. Where you're showing the open AI people they have like lots of documentation like this is what's happening in this code. This is how to navigate it. Like I I created this example over the weekend. The code base that like open AI that took 1500 commits and a couple million lines of code to get that self-evolving harness into a shape where they are letting it rip by itself, right? So there's no way that I can create a but they're proving that basically you can get to a certain level of autonomy but you basically read that thing and they're not trusting that to go make changes with the open AI website. They're doing that for an internal tool, right? They're still also being careful about

level of autonomy.

That's the TLDR for sure there. All right, this has been very useful. Cool. I think people know where to install the skill, how to get the skill. You can visit the Phoenix website as well. I think this is really useful but yeah, thank you so much for coming. Yeah, thanks for having me and thanks for forcing me to at least prove that war shipping actually makes some semblance of sense and hopefully it gets better over time. So really appreciate your time, Hal. All right, thank you.

Cool. Thanks. See you.