because it'll allow agents to augment us and not just communicate with us. So if agents can perceive the world and take actions in it, then they can help us carry out tasks. Here, my work has covered things like multimodal grounding as well as predicting structured actions and learning abstractions over actions and structures. So we'll see examples of all of these both in part one and part two of the talk. And finally, before starting the first part, I want to point to this kind of cross-cutting theme that sort of comes up in a lot of this work, which is handling uncertainty, ambiguity, and under specification. These kinds of problems are only going to become bigger as the size of the tasks that we give models grow and handling them robustly is a really important component to safety and having models behave predictably. So today we'll see some work on communicating uncertainty in the first part of the talk, as well as dealing with and resolving ambiguity and under specification and predicting actions with calibration, which we'll cover more briefly. This is the topic of a number of papers, including some that'll come up at various points during the talk today. So we'll start with the first theme, improving agents through multi-agent collaboration. And here I want to start with a quote from one of the forefathers of AI, Marvin Minsky, who put forth this idea called the Society of Mind. Here, Minsky was talking about how human intelligence works, but I think this applies more broadly to intelligence, which is that it stems from a large number of agents communicating with each other. This kind of framing led me to think about some key research questions. Firstly, how can we learn from interactions between agents to improve individual models? In other words, how can we train models to internalize the kind of intelligence that you can get from interaction such that these models become better interactive partners themselves? Another key question kind of underlying this is what is actually needed for successful interaction between different agents? And to start answering this question, I think we can consider a kind of team scenario that's probably familiar to a lot of people here, which is a trivia game where you and your teammates need to answer questions together as a team. Of course, this is a gamified version of a setting that comes up in much more serious places like healthcare, government, et cetera. And these kind of team scenarios apply both to human model interactions as well as model model teams. Now, everyone who's been on a team, trivia team before knows that there are a few key elements that you need to be a good teammate. First, you need to tell your teammates how much they should trust your answer. So if you're overconfident and you're wrong all the time, you're going to mislead your team. You also need to be open to changing your answer when someone else has a better response. So you need to be able to accept persuasion, but you should also advocate for your answer when you think that others might be wrong. So the key problem that we actually want to solve here is ultimately how do we teach models to communicate like people do? And one solution for this would be have them interact with people and use reinforcement learning to train them. But of course, people are expensive and it's hard to scale that kind of annotation. So the general solution that I'll propose here is that we should be simulating interactions between models and then rewarding desirable behavior. And this will allow us to actually use weaker models to train stronger ones. So we'll start by seeing how we can do this for calibration. I'll start sort of motivating calibration a bit more broadly, and then we'll zoom in on how we can actually improve calibration. So I've mentioned calibration a couple of times now. What is calibration? Well, it asks whether a model's confidence matches its correctness. So in other words, if a model reports high confidence for a given example like this one, then it should be right and vice versa, right? So if it's low confidence, it should be incorrect. In this case, the model, if it had been calibrated, would have probably said that it's not confident about its answer. More formally, what we're measuring here is if we plot the average accuracy and the confidence, we want to see if they're matched up. So in other words, we should get this kind of nice linear correlation between confidence and accuracy. More often than not, we'll get something that deviates from this either with a model that's less confident than it is accurate. That's actually quite rare. What's more common is an overconfident model, a model that says it's very confident and it's much less accurate than it thinks it is. We can measure

calibration on tasks, on generation tasks, but we can also measure this on sort of agent-y kind of tasks where agents are predicting actions. So for example, a task like this, where we're predicting a program for a calendaring agent, this can query your calendar, make meetings, invite people to them. And here we actually found that earlier versions of LLMs were actually quite well calibrated on these kinds of tasks. So you have a nice kind of correspondence here between token accuracy in this case and model confidence. A key area where this kind of thing comes into play, of course, is when you have agents doing things for you in the real world, you might have a robot stacking blocks and you want to know how confident the model is about succeeding on a given command like this one before you actually execute that command. Of course, in this case, these commands are pretty direct, but whenever you're dealing with natural language, you always have the risk of having some ambiguities. So imagine you have a command like this, put all the cubes in a box. Well, one interpretation of this is that all the cubes can go in the same box, but of course we could have another interpretation where these cubes end up in different boxes and they're still all in a box. In this case, if the model is calibrated, we can use its uncertainty to kind of communicate with the user that there are multiple outcomes here. This is something we explored more directly in work measuring calibration for LLMs on semantic parsing tasks with linguistic ambiguities. It's also something I've looked at in more multimodal contexts, looking at ambiguity and visual question answering. The point I want to drive home here is underscored by a different paper called Did You Mean? And in this paper, we showed how given a calibrated model, we can improve on these kinds of high uncertainty, ambiguous inputs by rephrasing the input and asking the user for confirmation, right? In this case, we might ask, do you mean put all the blocks in the same box? So asking for one of these interpretations. The really key thing that I want everyone to take from these slides is that if you have a calibrated confidence score, you can use it as a signal for when to trust the model. This is gonna be important in a lot of domains. It's important in these kinds of agent domains. It's also important in more information-seeking domains where people are interacting with models to get answers to questions. So I don't actually know the right answer to this question. I probably should have looked it up before. If I ask the model this, whether I accept the model's answer is going to be based largely on how confident the model sounds. So if the model sounds confident, I'll trust it. And if it doesn't sound confident, I won't trust it. And this kind of sets the stage for LACI, which is a multi-agent training method that we use to that we introduced for learning calibrated confidence. This is work that I presented at NURIPS a few months ago. And the kind of core insight to this paper is that calibration and confidence are important enough that languages have pretty nuanced and sophisticated ways of reporting confidence. So you could say something like this, right, to kind of report a high degree of confidence. But you can also say things like this. And between these two, I think there'd be broad agreement that option B is a lot more confident than option A, even though neither of these have any kind of numeric cues in them. And so in our work, we're going to develop a method that can handle calibration both for these kinds of explicit statements. And this is what past work has focused on. But it can also handle calibration for more implicit cues, like statements about expertise or background or general tone that also communicate confidence. The key insight here is that calibration isn't just about whether an answer is right or not. It's about how the answer is perceived by another agent. So in this case, if an answer is expressed confidently, then a sort of naive listener like me might accept this answer, even though this one is actually wrong. And if an answer is not expressed confidently, then this naive listener is going to reject the answer, whether it's right or not. To try to teach this to models, we take an approach that's motivated by pragmatics, where we model both the speaker and the listener. And the goal is to make the speaker pragmatic in the sort of RSA sense, because it should take the listener into account. So before training, the speaker just kind of gives out responses. After training, the goal is that the speaker should have this internal model of the listener that it's learned through this training, and that should inform whether it hedges or not on this

answer. Because now its incentives kind of change, right? So before its incentive was just to kind of get everything accepted. Now its incentive is to get correct answers accepted, but to get incorrect answers rejected. And it does this by using confidence as a sort of communication channel. So it uses its confidence to communicate
how likely the answer is to be correct. To implement this kind of pragmatic calibration, we used reinforcement learning. Specifically, we use preference-based optimization, where we start by bootstrapping preference data from a base model. We'll take a data set of questions and answers, in this case, trivia QA questions, which are these trivia questions where we know the right answer. And we'll query a base model a number of times for each of these questions. We'll get a variety of outputs here. Some of them are right, some of them are confident, some of them are wrong, some of them are not confident. And we'll then pass these to a listener model. And this listener model decides whether to accept or reject each of these answers independently. So the first answer sounds pretty good. I think the listener would accept that. The second answer sounds pretty confident as well. It's also gonna get accepted. The last answer doesn't sound confident, so the listener would reject this answer. Crucially, the listener doesn't actually see what the name is here. We actually remove that from the prompt. The listener does this just based on how the answer sounds. So this is sort of like a sentiment analysis task. And then we can compare these answers to the ground truth, which is Edmund Barton. So the first answer is right, next answer is wrong, last answer is again, right. And then using this data, we can define this preference function that we'll use to do reinforcement learning on the model. This function rewards both true accepts and true rejects. So correct answers that are accepted and incorrect answers that are rejected. And it penalizes the model for either being underconfident, so getting a right answer rejected by the listener. It penalizes the model even more for getting a false answer accepted. This is kind of the worst case scenario where we fooled the listener into accepting something they shouldn't. So here we use DPO, which is sort of the standard way of applying preference-based RL to large language models. And we train three different models on this data, Mistral, llama3 8 billion and llama3 70 billion. Crucially, what I wanna point out here is that the data that we use to train is actually generated from this Mistral 7 billion model. So we're using data from a 7 billion parameter model to train a 70 billion parameter model. Looking briefly at the results on the 70 billion parameter model, the key thing to highlight here is that compared to both the base model and a chat version of llama3, we get much better scores for area under the curve, calibration error, which is basically how far off from that perfect y equals x line are you, and then also listener precision. So how good is the listener distinguishing between correct and incorrect answers? This might be easier to explain also with the human evaluation that we did where we had people act as listeners. So people were basically accepting or rejecting answers to questions. Crucially, these are questions that they don't know the answer to themselves, and we measured their precision and recall here. So how good are people at discriminating between correct and incorrect answers from different models? Our method significantly improves over the base model in terms of precision while having roughly the same recall. And this is great because it means that after training, the model is better at communicating to people when it should be trusted and when it shouldn't be trusted. Breaking this down a bit more, the increase in precision here comes from a big reduction in false accept, so incorrect answer is that we're accepted. Qualitatively, this looks a bit like this. So here's an example from the dataset. I didn't know the answer to this question before making these slides. If I looked at the answer from the base model, I would probably accept it. It sounds pretty confident. It sounds realistic. It's actually wrong. It turns out that the model after training is still wrong. It hasn't learned the correct answer, but it has learned to hedge, right? So it says something that probably nobody would accept as an answer. It says it doesn't really know. It doesn't have a lot of confidence, et cetera. So in Lacey, we modeled confidence as this property of the speaker, and we optimized the speaker to be well calibrated, and that resulted in better hedging on these examples that were likely

to be wrong. But we should also consider the listener here, right? How much weight should the listener be putting on the speaker's utterances? And when should the listener's knowledge factor in? So if the speaker says something that's obviously wrong, but really high confidence, or vice versa, it says something really right, and low confidence, the listener should be able to kind of override the speaker and still accept it. And that takes us to this next paper that was recently accepted to NACL, in which we introduce a method to train models, both as speakers and listeners simultaneously, to accept persuasion that makes them better, but reject it when it would make them worse. So the focus here is on persuasion. This is a key to successful teamwork, because it's the way that we change each other's beliefs, the ideal scenario for persuasion maybe looks something like this. You start with a question, at least in this trivia domain, and one of your teammates says something totally wrong, your teammate says Elton John, you actually know the right answer. So you come in and say, no, it's Shirley Bassey, and your teammate says, okay, sounds good, let's submit that answer. You get this question right. This is something that's come up often in human model interactions as well. There was a recent study comparing chat GPT to doctors, as well as doctors combined with chat GPT, where they found that models did better on their own than they did on this team with doctors. The key thing that was actually holding doctors back from benefiting from the model was a lack of persuasion. So the doctors weren't persuaded by the model when it actually would have improved their answers, and this held back the team performance. But persuasion can actually cut both ways, it can be both positive and negative. So we've seen this positive direction. In the other direction, persuasion can all have a lot of negative impacts. You can kind of misinform models, it can lead to incorrect beliefs being adopted. It can also be used for jailbreaking, right? You can basically persuade a model to do something that it shouldn't be doing. And that's mostly what past work is focused on, kind of documenting these negative aspects like jailbreaking and misinformation, where they found that even the strongest models are really susceptible to this kind of negative persuasion. Our work is the first to propose a solution to defending against this kind of negative persuasion, while also balancing this positive aspect of persuasion. And here, this solution applies to these kind of dialogue agents whose state is the dialogue history and whose actions consist of producing text, in this case, text that either accepts or rejects a proposed change. So here again, we'll use reinforcement learning to train these models according to a multi-agent signal. Here, we're gonna use a tree-based approach where we construct a dialogue tree between two agents. So we'll give both of these agents this question and we'll get answers from them. In this case, they disagree, right? One says Shirley Bassey, one says Paul McCartney. It's now time for them to kind of discuss their answers. So we can descend in this tree to expand the dialogue. And the reason it's a tree is because we have multiple possible responses from each agent. So each agent gets different prompts. One prompt might be resistant. So this prompt basically tells the agent, never change your answer, always keep it the same. So this agent is gonna say, no, wrong, it's Shirley Bassey. I'm not accepting Paul McCartney. The other agent is kind of a pushover, right? It basically says, you're always wrong, always accept everything. So this agrees right away. Whenever these agents agree, we're gonna terminate that branch in the tree. But we can keep expanding this for multiple iterations. And we also have different argumentation strategies. So one agent, one prompt will tell the agent to use a kind of logical argument, maybe a credibility-based argument or an emotional argument. And so we'll have these different kinds of persuasion. You can think of this tree as a compact representation of a lot of counterfactual dialogue. So one path through this tree is one dialogue. And at each node, you're kind of branching off into seeing like, what would have happened if I had responded this way or that way? And because of that, we can score this tree, right? We can start at the base nodes or the leaf nodes here and give them a score of one or zero depending on whether they're right or not. And then we'll recursively score the rest of the tree based on whether that node is right, but also based on the scores of its children. And the intuition there is that a turn that leads to a lot of correct answers should be rewarded.

We'll repeat this process all the way up to the root node and then we can construct preference data from these scored trees. So basically we'll go through the tree, we'll grab some of this dialogue history by sampling a path and then we'll compare these counterfactual responses. And in this case, we'll get an example of resisting negative persuasion. Here the agent that disagreed basically resisted a wrong answer and got the right answer, but we'll also have examples of accepting positive persuasion where an agent that has the wrong answer changes its answer to the right answer. So we have both types of persuasion in our dataset and because the dataset is actually balanced exactly between these two types
of persuasion, we called our method persuasion balance training or PBT. We have a bunch of results on misinformation and things like flip-flopping in the paper, but I'll briefly just show results on our balanced evaluation where the data is also balanced between positive and negative persuasion. So half of it is negative persuasion. Example is where you want the agent to minimize the number of answers that, wrong answers that it accepts and the other half is positive persuasion. Here you want the agent to kind of maximize the number of corrections that it's accepting. And the metric that I'll show here is the final accuracy of the agent. We'll compare a few different models here. We'll compare a llama 3.1, 70 billion chat, an accept only version of this model that's only been trained on accepting positive persuasion. So it knows how to accept corrections, but it hasn't been trained on the resisting side. The opposite of that is the resist only. So it's only been trained to kind of resist bad persuasion, but not accept positive persuasion and then our balanced model. And looking at the results here, we'll see that the base model has a reasonable rate of resisting misinformation and accepting corrections. So each of these plots shows the accuracy of the model on each of these splits and then the overall accuracy is the solid bar, which is the average of the two. We'll see that the accept only model gets a lot better at accepting corrections, but it over accepts. So it accepts a lot of things that it shouldn't and the resist only model kind of does the opposite. It over resists. So it basically hurts its performance because it won't accept any corrections from other agents. We'll see that we can get kind of the best of both worlds here with our PBT model. It gets better at accepting corrections, but it also simultaneously gets better at resisting misinformation. Crucially, all of the data that we trained on was generated by 8 billion and 7 billion parameter models kind of dialoguing with each other. And then we use that to train a 70 billion parameter model. So this is again scaling up from these smaller models to bigger models. Here, we also looked at a team setting where we paired this strong 70 billion parameter model with a much weaker 8 billion parameter model in a discussion. So you can think about this as like if you were on a team, on a trivia team with someone who's really bad at trivia. We found that for non-PBT models, the performance of the team depends a lot on who goes first. So sometimes the strong model pulls the weak model up when it gets to get its answer in first, but sometimes the weak model pulls the strong model down. Because the strong model isn't really able to say no to the weak model. In other words, there's a lot of variance here between these different settings of who goes first. That variance is reduced a lot when we use the PBT model and the overall team performance is more consistently strong with PBT. So that wraps up part one of the talk here. We saw agents communicating with each other and we saw how we could improve agents by modeling multi-agent interactions. Here we're gonna go a bit deeper into what it takes to have this kind of single successful agent. Because to be useful in the world, these agents need to be able to both perceive it through multimodal inputs and representations and they need to be able to take these kind of safe and effective actions. We've seen some of this safety side with this calibrated story earlier where we had these calibrated actions. So I want to move on to discussing a bit more of the kind of effectiveness part now. There are a few really important challenges here. I think in the work that we've seen so far earlier in this kind of semantic parsing work, we've assumed that the actions that the agent takes are provided to us. But in reality, agents should really be learning what actions to take. They should be learning abstractions over actions, which is often thought of as kind of learning skills. Another key challenge here is

how can we adaptively improve models on the skills that they already have? So once we're dealing with dynamic things like agents, we'll see that static data sets are going to be kind of incomplete. We want to adaptively build data and environments for these agents to learn in, both for text-based and multimodal environments. So we'll start the second part of the talk with this key question, how can we learn verified abstractions over actions and code, in other words, how can we learn the kind of domain-specific languages that we were using for parsing earlier? And then we'll come back to how we can actually use these abstractions downstream. Now, imagine you get an instruction like this, right? This is actually an instruction from the Alfred Datas environment, where you have this agent in a 3D environment. It can go around and pick things up and move them around, and it gets these natural language instructions that give it some goals. Now, I think if I asked you to do this task, I imagine you would give me a series of actions like this. You would tell me, you have to go get the first credit card, put it down on the table, et cetera. But of course, if you actually looked at what you do in practice, it might look more like this, these sort of low-level actions of what an agent would do in this environment. These kinds of long trajectories appear in a lot of domains, including for things like web agents, where you might ask an agent to book use a flight to somewhere, you have a long trajectory of clicks and typing and things like that, that represent this sort of low-level trajectory. A key question then becomes, how can we learn these kinds of abstractions so that we can go from what you do more to what you think? And I'll argue that language is a fantastic source of these kinds of abstractions, as illustrated by this example on the left, where language encodes these really nice, high-level abstractions over these low-level actions. This is something that I've worked on in a number of iterations, one with Microsoft Research, where we learned these kind of skills for Alfred, as shown here. Our algorithm learns these kind of abstract actions, like navigation, picking up objects, and it's all guided by a large language model. It's something that I'll also go a bit deeper on in the next paper that I'll cover, which is Regal. This looks at abstraction learning more generally for code. So the abstractions that we're looking at here are kind of one level up from what we just saw. Here, we'll look at this kind of logo-drawing domain, where again, you have this agent, and it gets a query of a thing that it should draw, like this six-sided snowflake with nonagon arms. Now, the actual low-level actions that the agent takes are things like going forward a certain number of steps, turning left, turning right, picking up its pen, putting the pen down. But we'll actually assume here that we have a little bit of a more sophisticated action space that looks more like code. So for these really repetitive things, we can have something like control flow that lets us do for loops and things like that. What I'll say is that I think that the current text-to-code paradigm that we have right now is really ill-suited for this kind of domain, because this current paradigm has a very single-input, single-output flavor. So in this case, we would give this command to an LLM, and we'd ask that LLM to predict a single output for this single input, not really sharing anything across different examples. And this will kind of lead to our downfall here, because if we execute this program and get the result, we'll get this kind of ugly-looking shape here. This turns out to actually be wrong. And the reason why is kind of indicative of two different problems, a lack of abstraction and a lack of reuse. So maybe I'll start with reuse here. I think it's probably been drilled into everyone here at some point, maybe by Jeff, maybe earlier on, that you should avoid writing repetitive code, because this can lead to unnecessary mistakes. In this case, even though you have a different example where the model drew this nonagon correctly, in this case, it just predicted the wrong angle. It said 40.5 degrees instead of 40 degrees, and that led to the wrong shape being drawn. This could have easily been avoided if you just had a function for this. Another way that's maybe less intuitive that a program can be improved is through abstraction. And what abstraction does is it lifts reasoning away from the agent and into the programming language. This encapsulates a lot of the reasoning that you would need to, and it turns the problem into a kind of retrieval problem. So instead of reasoning about like, oh, I have to draw a nonagon that has nine arms, I need a for loop, et

cetera, all I have to do is match these tokens, small nine gone to this function, small nine gone, and now I've already solved half of the problem. So this makes the overall task a lot easier. The core problem that we're addressing here is how do we learn these abstractions from data? And to do this, we apply a technique that everyone who writes code does all the time, which is refactoring. We'll take working but inefficient code, and we'll ask a large language model to rewrite that code to be more efficient without changing the outcome. And what inevitably ends up happening here is that we end up with these helper functions along the way, things that encapsulate things like draw a small nonagon or draw a pentagon that we can then store for use at test time. So during training, we're gonna learn, test, and prune these abstractions in order to build up this code bank so we can learn abstractions on one example, test them on another example, figure out which ones work, which ones don't. And then at test time, we'll
actually ask the agent to use these abstractions in the code that it's writing to solve some of these test problems. The way this looks like in practice is that we have two different agents, both of them have a prompt. The baseline agent gets a prompt that has this query, and it will use that query to retrieve relevant demonstrations from the training data. So in this case, it's gonna retrieve 10 demonstrations of similar queries. Maybe it finds this demonstration with a five-sided snowflake. Maybe that helps it draw snowflakes with different sides. The regal agent looks pretty much the same. It starts with this query. It also gets a list of these helper functions that have been learned during training. And some of its examples are dedicated to these primitive-only programs, so programs that are written in these primitives that just have forward, backward, left, right, et cetera. And some of its programs that it retrieves are going to have examples of these helper functions being used. So here we might see an example of this nonagon function being used. And when we look at the results across three domains, we'll see some nice improvements. So here we looked at this logo-drawing domain where you have these programs that are drawing logos. We also looked at date, which is math-word problems about date reasoning, as well as text craft. This is a kind of Minecraft-style environment where an agent needs to collect ingredients to craft different things in Minecraft and then put these ingredients together in different ways. And we see nice improvements coming from our abstractions compared to writing programs with just primitives. So in the paper, we also show these improvements on other sizes of models, other open-source models, as well as closed-source models. I think a core assumption that I want to point out here, and it's something that was borne out in this paper, is that LLMs are able to propose these abstractions over actions. The nice thing about these abstractions is that because they're code-based, they're verifiable, and they're also testable. I can go back into these code banks and I can see what functions were learned. I can actually even go back into the code bank and I can modify and improve these functions if I want to, as opposed to abstractions that neural networks would typically learn, which are sort of black box, right? You can't just go in and change weights in a neural network. This kind of verifiability actually comes up in another recent paper from our group that we put out a few days ago, where we introduced UTGen. This is a way of training models to automatically generate unit tests for a given piece of code. And this kind of unit test generation could actually help further verify the functions in a regal-style code bank. So the last question that I want to address today is now that we have kind of seen that models can learn abstractions, what should we do with abstractions? In this case, we'll actually look at abstractions over model errors that allow us to generate data to improve models on their weak skills. So this was work that was recently accepted to iClear as a spotlight paper. And what it looks at is kind of the future of these models, how we can improve that future through abstractions by building data generation agents. So going back to the very first part of the talk, I was talking about how we're starting to get worried about the data that we're training on, because the models that we're training are extremely data-hungry. And there's this general concern that we're starting to butt up against sort of the limit of the amount of data that we have available on the internet. We've kind of scraped everything. And so generally one way that we've

seen is maybe a promising way to get enough data to train models in the future is to generate synthetic data from powerful generative models. The idea here being that we might take a teacher model and we might ask it to create synthetic input-output examples, and we'll use those examples to then train another model. This has been successful in domains like math and reasoning, where synthetic data has been really crucial to building strong open source models. But a key thing to note here is that this teacher model is actually not an agent. This is just a model in this kind of framework. Now, why does this matter? Well, this is a setting where quality matters a lot more than quantity. We need to generate not just any data, but we need to generate the right data for a student. And that's gonna be specific to each student model. Like people, different models have different skills that they might be better or worse on. And it's also temporally dependent. As you address one skill, this might reveal another weakness that needs to be addressed. So the process of improving models according to their weak skills right now is really human-driven. In other words, people are acting as agents. They train models and analyze their weaknesses. They then get this teacher model to generate a bunch of data. They retrain the model and they repeat this process over and over again. And this is expensive and time-consuming. So in our paper, we ask, what would automation look like here? And specifically what we do is introduce a framework for testing different ways of automating this process with data generation agents. These agents, they operate in an environment. And so the environment looks like this. It'll have a student model at time t where t could be zero. The environment then evaluates that student and sends feedback from the student's errors over to the agent. This agent is responsible for creating data. So this agent will create whole examples, a whole data set of examples for the student to learn from. That data is then given back to the environment and the environment trains a new student model. It updates this and the whole process repeats. We support three different kinds of environments in Data Engine that have different state representations and they vary in how they use skills. So this open-ended environment is really flexible. It shows the teacher just a list of errors that the student made. These skill-based environments first use LLMs to discover skills from these errors and then they report performance on each skill. I'll show you briefly how this works. We discover these skills from examples in this kind of two-step process. So first we'll ask an LLM to label each example with a set of skills that are needed to answer the question. So maybe there'd be things like hue or shade identification and bird identification, mammal identification, kind of specific skills for each question. We'll then ask in the second step for the model to take all of these skills and cluster them together into a sort of hierarchy. So we might get something like animal identification or color identification out of this. And this can be done for a variety of domains. So in this case, I'll show examples of this skill tree that we have for visual question answering. I wanna note that all of the images here are examples from our generated data. So not only are the questions and answers generated, but the images themselves are also generated. We can also generate math problems according to skills. And that's what this looks like. Overall, we have four different domains that we evaluate with five different data sets. So we have two data sets for visual question answering. We have GQA and natural bench. These questions look like this. They're questions about images. We also have math reasoning. We evaluate that with the math data set. We have live code bench, which evaluates code reasoning or coding ability. And then we have tool use, which we evaluate with this M&M's benchmark where the agent has to kind of predict interleaved calls to different tools like question answering, speech recognition, querying, et cetera. And our benchmark comes with a few different kind of baseline agents already implemented. I'll show some results from these baseline agents across these different domains. So we have GQA, which is visual question answering, math, coding. And in these domains, the baseline agent that we include is actually already able to improve these student models across these tasks and teaching environments. The thing I want to point out here is that these student models have already been extensively post trained. So improving llama three, eight billion on coding is actually non-trivial because it's been trained on a large

amount of code. So you need to generate things that it hasn't already seen. We see these improvements continuing on natural bench, as well as M&M's. And this underscores the kind of breadth of tasks that these agents can improve models on. Moreover, I think the positive results here on M&M's indicate some promise for agents kind of developing stronger agents because tool use is starting to get closer to this kind of agent style task. So you could imagine an agent teaching another agent to get better. I think there's effectively unlimited room for growth on these tasks. And so for that, we've released a leaderboard where these agents act as kind of baselines. I hope that people can kind of develop and submit their own agents here. And we plan on adding more domains also in the future. I want to take the last couple of minutes to zoom out a bit and discuss some future work. I think a core area for future work is in scaling up agents, both in terms of the domains that we can apply them to, which right now are limited to areas where we have human-defined environments and human-defined rewards, as well as the kind of time scale over which agents can behave autonomously. Right now, we're just at the starting point where these agents need a lot of supervision and intervention. By automatically learning abstractions and generating data and environments, I think we can

create agents that adapt on the fly and we can enable them to handle new environments. To act across these longer horizon tasks, agents will also need to take information-seeking actions that reduce their uncertainty, which of course hinges on model calibration. And they'll need to be able to teach themselves key skills both by creating data and by seeking out informative experiences. I hope to connect that kind of work to the challenges of devising learning exercises and environments also for human learners. And this is something I've started exploring as part of our NSF Institute on AI and Education here at UNC. Scaling up agents alone is going to be insufficient. They also need to communicate with each other and collaborate. And to this end, we'll need to also simulate longer-term factors like reputation and trust, and we'll need to consider mixed cooperation scenarios like negotiations, where the goals of different agents are only partially aligned with each other. And finally, I think if we want agents with robust and long-term autonomy, they'll need to handle challenging multimodal inputs. The first key task here is handling ambiguous or underspecified inputs, both by interacting with people as well as by having agents interact with each other and their environments. To determine when these interactions should take place, agents need to be able to predict the outcomes of actions or events in their environment. And to that end, we need to imbue these agents with world models that allow them to predict future outcomes. I think that by addressing these core challenges, my work will pave the way for reliable and trustworthy agents that interact with people in intuitive ways. So I'll close there. I want to thank all of my collaborators on these papers, as well as all of you for listening. I'm looking forward to taking your questions and discussing the work. Okay. Yeah, thanks, Elias. That was wonderful. So you can hear us okay? Yeah. Yeah, okay, great. Let me make this little bigger here. I guess, hold on. Let me just, you can all think about your questions while I get this set up. Okay, well, I'll just leave it. It does open on him, obviously. Ah, okay, great, awesome. Cool. Okay, so, yeah. We have some time to ask some questions. So yeah, do you want to? So I'll call on them, Elias, and then... Hey, I'm Adam. So first of all, yeah, that was fascinating. The question is about the confidence aspect of part one. So that's something I'm really interested in. And I noticed that most of the examples you gave were confidence on kind of question answering or very well-defined tasks. How do you see that kind of expanding to less defined tasks? I think in your future work you mentioned like negotiating, let's say, or what I'm really curious about is per, I guess, either LOMs or kind of agents that are working together on a specific task, like some kind of negotiation. How do you envision confidence being measured in kind of more abstract things like, oh, I have a good idea about what we should do as opposed to just what's the answer to this question? I'm sorry if that's like, I'm taking this really far as the first question, but that's what I was... Yeah, I know. I think that's a very kind of spot-on question. I think this whole question of calibration is kind of limited by

the fact that you can't measure it if you don't have a way of measuring accuracy. So this actually comes up even in the kind of work that I showed earlier on measuring calibration and semantic parsing where it turns out to be quite kind of difficult to figure out what is the right way to actually measure accuracy because you can imagine that, for example, if you're predicting programs, you don't actually necessarily want to measure your accuracy on individual tokens because there might be an infinite number of programs that all execute to the same result. So do you then measure in terms of execution accuracy? I think broadly this kind of connects with how do you verify answers and how do you verify what your model is doing. So it happens to be that the kind of domains that we can look at for calibration are ones that have these really easily verifiable answers. So it's partly why we look at things like question answering where we just know who the first prime minister of Australia was. This isn't like an opinion question. It's really easy to define what it means to be over or under confident there. I think there are certain tasks that we maybe don't measure calibration on right now because it's too expensive. So if you have something like these sort of longer term interactions, these more collaborative things, you can still measure accuracy in the end or you have some kind of error score, which is how good did your agent do in the negotiation or something and then you can measure like is that sort of correspond to how confident the agent was earlier about how well it would do. I think that's one of the ways that like if we just expand it and we have enough examples, then we can start to kind of measure calibration. But I also think there are probably these problems that are more like almost like subjective things, right? Where it's like, can you define what it means to have like a good email response or something? So in those cases, maybe it could be measured based on like whether a person accepts the response or not or something like that or have some kind of scoring rubric. But it is generally a very difficult thing to try to quantify. Yeah. I guess sort of more of a clarification question. This is regarding when you were talking about like getting models that interact better in teams. First of all, I just want to say that I thought that was really interesting. It wasn't something I'd really considered before, but after your talk, I really understand like why that would be beneficial. But the one part I was kind of confused on is when you had that tree, where you have the model that always agrees and the model that always like shows resistance. I mean, you said the tree would stop when the models agreed, but if one model is always agreeing in one model, like wouldn't they just agree with each other after the first node? Like how do you actually get down that tree and like use those results? Yeah, that's a good question. So basically the way that we're doing this is with a prompt. So the prompt instructs the model to basically bias it towards agreeing, but the model doesn't follow the prompt 100% of the time. So mostly this node does get basically pruned, right? So most of the dialogues basically end there because the model generally agrees with its parent node. But sometimes it basically disagrees still because whatever the other person said was so obviously wrong that even given this prompt that says, please agree, the model still says, like no way that's the right answer. But it's actually like these nodes are quite important for this kind of second task. Because if we want to have these examples where we basically have one agent that is persuaded and one agent that isn't persuaded, then we need this kind of like counterfactual. So even though this branch gets pruned, the other branches still expand and give you this kind of counterfactual example where like one agent has disagreed and the other one hasn't. Oh, okay, yeah, I think that makes sense, thank you. Yeah, Elias, can you hear me from over there? Yeah. Cool, thanks for the talk. I was curious, you gave a comment early in the talk regarding how some of the older language bottles were surprisingly well calibrated. I was curious how that statistic or how model calibration has changed with the newer models that have come out, if you had information on that. Yeah, so actually in the, so this was this plot. So the part of the reason why these models were well calibrated here is because this was kind of before you could do things like zero-shot code prediction. So these models were actually trained on this task. So these models have seen a lot of data of these kind of like calendaring programs and that makes them fairly well

calibrated. So in this paper, we actually also looked at some early versions of, I think it was CodeGen was the model at the time that was kind of the best like code pre-trained model. And we did in context learning with that model. We found that it was still relatively like surprisingly well calibrated, but definitely not as well calibrated as these trained models. I think that overall, it seems like basically the more data the model sees, the better calibrated it's going to get because to some extent, it feels like this question of calibration is really a question of like whether an example is in domain or not. So if you have a lot of training data for your domain, then almost everything is in domain and the models are quite well calibrated on those things. Once you start getting into these more like out of domain things, then they get less calibrated. So partly, these large language models have an incredibly wide domain because they've been trained on so much data that most things that they see are sort of in domain. And so they'll be reasonably well calibrated on something like Python. But there again, you have this problem of like measuring confidence and measuring accuracy becomes difficult like for a model predicting a Python program, it's not clear like where you should pull the confidence from. Is it really meaningful to look at like token level confidence or do you need to look at something like a whole sequence of confidence scores or something like that? Thanks very much. Okay, yeah, if you can answer that. I have another one if you don't mind. I don't recall the paper, I'm sorry, because you covered a couple, but there was one of them in there where the language model was kind of to calibrate and hedge its responses based on the counterparty and the conversation.

If you were interested in the work or if there was a lot of work on kind of the flip side of that, I sort of almost interpreted it a little bit pessimistically in terms of how should I, I have a certain amount of knowledge, how should I word my claims to the counterparty such that they'll accept, I think to be true statements? The flip side of that is like, okay, you're an unknown party in this interaction. I may know something about you from our limited interactions that we've had so far in the conversation. How much should I trust you as some other agent that I'm interacting with? Yeah, so I think this was something that actually came up a little bit in the review process where one of the reviewers basically asked, couldn't you flip the objective here and basically create a model that kind of tricks people? So if you go back, if we go back to like this objective, right now it's the case that like we're rewarding the model for getting incorrect answers rejected and getting correct answers accepted. But of course we could flip this and basically say, like you get a reward if you like trick someone into accepting something that's wrong and you get penalized if they like guess that the thing that you were saying is wrong. So I think that that is like something that could come up. Partly this was something that we tried to kind of address with the second paper on trying to like learn to accept or reject persuasion. Because you can kind of think of that second scenario as something like this, like the scenario you mentioned in which you need to kind of figure out like how much should you trust this person or this party that you're interacting with? It's kind of a question of like, should you be persuaded by what they're saying? And so here, what we're trying to model is basically like a one-turn version of this, which is like basically is the thing that you're trying to convince me of plausible enough that I should change my answer? Or is the fact that you're trying to convince me about like completely implausible and I should reject that fact? But I think this gets to this point in future work about like learning about like reputation and things like that, which is that if you want to do this over, like people don't just do this over a single turn unless you don't know someone at all, right? You might have a trivia teammate where you have played trivia with them often enough to know like, okay, well, you know, this trivia teammate is always overconfident and always says things they know everything about sports and they actually don't know anything. So if there's a sports question and they give us an answer, I'm gonna ignore that because they're not very reliable or something like that. Or they have a really stellar reputation and we know that like, if this person is saying they're really confident, then we should definitely go with that because they basically only say

that when they're really gonna be right. So I think these are the kind of things that are more like multi interaction things that could help with this. I have another question if you don't mind. I was kind of interested when you talked, like you talked briefly in the beginning about making agents better teammates with people. It seemed like the talk didn't really focus on that, but I would think that a major thing for that would be not just communicating when you're unconfident in an answer, but also recognizing like, where the other person is coming from. Like obviously if I was like on a team with someone else in the NLP program, I might assume that like, oh, they have a lot of like background knowledge. Whereas if I was just talking to like a random person on the street, I would have to like kind of figure out what level they're at, what I need to explain more and less and stuff like that. Have you guys like done any like work on that or have any like, like I guess like future work or like, what would you think would be done for that? Yeah, I think that's a really great area to explore because I think like a big problem with models right now is that they're very like uselessly verbose. So if you ask them a really simple question, they'll give you a really complicated answer. And what you really wanted was this, like maybe some people want that really complicated answer and some people just want like a really direct answer to their question. And the model isn't really like kind of modeling this background knowledge that you mentioned where somebody, like if you know that someone has taken a bunch of NLP courses before, you're not gonna explain some concept that they would already know at length. I think that to some extent, this is something that's hard to do because it's hard to get like persistent interactions where like you would, if you want to kind of learn this, I think like a lot of industry labs might be well positioned to do this because they have users that are like interacting with their models very frequently. But it's hard to run that kind of study, I think without having like sort of longitudinal data. I think that one, like this kind of like personalization direction in general feels like an area where there's gonna be a lot of growth in the near future. But our work hasn't really looked at like, how can we basically model like what is the user's current kind of knowledge state and to what extent does it need to be like taken into account by the model that like this user seems like they know a lot, maybe they need a certain kind of response versus a different user who doesn't know much. Thank you. Cool, awesome. Yeah, okay, Adam? Yeah, so in one of your early slides where you kind of have a diagram of the concept of like collaboration in the future. So I wonder if you have an idea of kind of how the setup would actually look of multiple agents collaborating on something. What this one, yeah. Whether, like if you have specific use cases even in mind, I'm trying to understand whether you mean like there are several agents that are trained like basically copies of each other and somebody tells them here's your task, work together to achieve something or whether they're different agents, each one has a specialization or kind of a role or, so could you clarify if you have ideas? Yeah, so the kind of short answer is both. I think like work that I didn't get time to cover today is more about like multi-agent reasoning, which is basically this setting where you have, rather than giving your question to a single model, you would give it to multiple models and you'd basically say, please discuss this amongst yourselves and come up with an answer, right? And that is a setting where you can either have kind of multiple instances of the same model, you can have different models. Our work has been mostly looking at like different models because I think if you have instances of the same model, it's sort of limiting like what kind of new knowledge you can get versus if you have different models, you could imagine like one model is really an expert on one domain and another model is an expert on another domain and they can each bring kind of different expertise or at least different like pre-training data and things like that to the table. So that's kind of one direction. I think another direction that's quite interesting to me is more, so that's sort of the collaborative side of it. I think the other thing that we're probably going to see kind of happening more and more is like a kind of semi-collaborative setting, right? So it's not purely adversarial, right? So we actually, we have some work on like a paper called GT Bench, which was looking at like a game theoretic benchmark

for LLMs where you actually have LLMs playing like strategy games against each other. So they're playing games like poker or prisoner's dilemma or something where like one, it's a sort of zero sum thing, right, where one agent wins and one agent loses. I think in reality, right, most things are not zero sum. So for example, if you think about like the thing I brought up later, which is negotiations, right? Negotiations are somewhat adversarial because you want something, like let's say you're trying to buy a car and you're haggling about the price. You don't want to spend money and the other person wants you to spend more money and you're negotiating about that, but ultimately it's also cooperative because you want a car and they want to sell you a car. And the thing you need to agree on is the price. It's not that like, if you leave without a car, both of you are unhappy about this outcome. So these are settings where I think that we'll also see like interesting things happening with LLM agents. I think that there are, I have a lot of like head of concerns about these kinds of settings because I think that I can see a lot of scenarios in which I guess basically, basically, bigger players can kind of misuse LLMs. So a domain I can think of here is like, let's say like insurance claims or something, where like if you are a really wealthy company, you can afford to have automated models that basically do the negotiation part. So they can basically deny things or like negotiate care providers down. And that costs you nothing because you've put no human effort in, right? Like you're just running a model and it's basically like, you know, wheedling the other party down. And at some point the other person's gonna be like, I'm out of time, just like whatever, this is fine. Or if you, let's say you call your, let's say you want to dispute a bill with your electric provider. If you get on a phone with an agent, right? Like a person, that person also has like a fixed amount of time. And if you stay on the phone long enough, maybe you'll be able to negotiate with them. If you have another model on the end of that call, right? It can negotiate with you for a year and eventually you'll give
up. So then I think we'll get into a scenario where basically you're gonna say, well, this is stupid. Why would I talk to a robot? I'm gonna get my robot to talk to your robot. Now whoever has like the stronger negotiation robot is gonna come out on top. So I think this is something that's kind of important to, like I think it's something that's gonna happen either way. And so it's important to sort of get ahead of like, how do we make sure that we can kind of even the playing field here? Yeah, do you have a question? One question. So does like reasoning always improve the confidence score? Like when, like so in O1 and deep seek, the model reasons itself and then talks to itself and comes to a conclusion and then gives an answer. But how does in multi agent, like in this framework that we're trying to do. So does reasoning, can we conclude that reasoning always is gonna improve the confidence score of the model? Yeah, this isn't an interesting question. So I haven't actually checked whether like O1 or deep seek, these sort of long, long chain of thought or like long reasoning models are better calibrated. My sense is that, so there's sort of a hairy thing with calibration in general, which is that there's this kind of degenerate solution that happens where trivially as you get more accurate at a task, you generally also get better calibrated because what ends up happening is that if we go back to this, let me see if I can get this, yeah, this calibration plot. So most of the time models are up here in this like overconfident region. So they basically say like, I'm 100% confident about my answer for almost every answer. And what then ends up happening is that as you get more accurate, right, your calibration error decreases because you're more up in this corner of being 100% right and 100% confident. I wouldn't necessarily say that the model has actually learned anything in terms of calibration. It's still sort of overconfident in the sense that it only has one answer, which is I'm 100% sure, right. It's just that the task has gotten easier. And so it seems less overconfident because it's an easier task for it. I think that there may be some promise to things like DeepSeek R1 because of the training on rewards, like long distance rewards. If you do enough of that training, I think the model will hopefully implicitly capture some of that reward model and basically be able to predict whether it's going to get a reward or not with a particular answer because

it should be trying to like maximize the reward. And that should make it probably reasonably good at, or hopefully more calibrated. So I think there is some promise there, especially because also there's been past work showing that RLHF generally leads to models being better calibrated, at least in terms of the words that they say, sort of this like verbal calibration kind of stuff. So it could be that those models end up being better calibrated, but I would have to check. Cool, awesome. Yeah, I had actually one or two questions. So yeah, it's a great talk, by the way, I really liked it. So with the Regal work, so like the language as abstractions, so that kind of reminds me a little bit of planning work, so having LLMs plan and make a high level abstract plan and then like whatever tasks they're doing. And it's kind of how, I was curious how you see your work fitting in and then in the future, like, would LLMs do a high level natural language description of some big program and then like slowly start to solve it or like how you fit it, think of it with like bigger programs and stuff. Yeah, so it definitely has strong connections to planning. So actually, and like sort of hierarchical reinforcement learning. So this paper, which was looking at the kind of skill learning, this was looking at it much more from this kind of planning domain where the skills that you're learning are basically like an intermediate plan for then generating low level actions. So you basically start by generating, you start by learning these types of skills. So you need to learn what these things are, like go get a credit card, put it out, put it on the table, get a second credit card. These are the skills that you kind of infer from your training data. And then at test time, your agent basically starts by predicting these high level skills and then conditions on those high level skills to predict these low level skills. And this basically improves the performance because it makes the learning signal a lot denser, right? There are a lot more examples of like navigation to a particular room than there are of like entire trajectories of a particular goal. I think Regal is doing something slightly different and that it's almost doing like the opposite or what's different here I think is that the second part of that sort of high level to low level plan is deterministic because what we're retrieving is, in this case is a function, right? So we've actually created like there's, for each of these training runs, Regal saves a file that's called like code bank.py that has all of these functions basically like hard coded into them. So rather than having this hierarchical policy where you have a controller that's first predicting your high level actions and then you have another policy that's taking high level actions to low level actions, we just have the controller and the controller gets to mix high level and low level actions because it can still do these like low level things like forward and left. But when it picks a kind of high level action like draw small nine gone, that just gets called from this predetermined thing. So this only really works because we're doing this in code. Of course, if we were doing this in other domains like in a robotics domain where you have like more fine grained actions, you can't like hard codes anything then this doesn't work. And that's, I think one of the limitations of doing this in code is you have to have an action space that can be defined in that way. But yeah, it's definitely has this sort of hierarchical planning kind of flavor. Yeah, so people have, I know we've been asking like pretty high level questions, but if people have like more basic questions or more detail or something you didn't find clear, you can ask that too. Any other things people wanted to ask? Yes, Sam. I had a small detailed question. I apologize again, I don't recall if it was the, which of the papers were you presented, but there was one where you were using the generations and hedging of Mistral 7B to then train Llama 370B I think. Yeah. Opinions on that being off policy and the impact of that or the efficacy of that in improving the hedging abilities or where the positive hedging abilities of Llama 370B. Yeah, I think, so this was interesting actually because I think this project to some extent went against some of the like established knowledge in that we did DPO directly on the model. We actually didn't do supervised fine tuning first. And I think that, and it still just worked on the 70 billion parameter model. I think the reason for this is that ultimately what we're doing here is we're not actually teaching the model anything new. We're kind of just teaching it to match something that's already kind of present in the model with a certain

output style, right? So you can think of this like hedging style as an output style. And I think that to some extent, this information about the model being uncertain kind of has to be present in the model because on these uncertain examples, what we found is that if you sample the model a lot of times, you get a lot of diversity, you get a lot of different responses, which means that somehow like somewhere in the model, there must be certain parameters that lead to this high diversity. And what you're really kind of teaching the model to do is pick up on that signal and then map that signal to these sort of like hedging behaviors. And so I think that maybe that's just in distribution enough because we're not teaching it to do any kind of like new reasoning. We're not teaching it to kind of do anything that it wasn't able to do before. We're just teaching it kind of when to do these things. I think that's probably why this works. I think as you kind of go further out from that and try to do things that are like more and more far away from like what the model does on its own, then you need to add in these other steps. So for example, for this persuasion training data that we had where we have these examples of like positive and negative persuasion, here we did actually have to do supervised fine tuning before DPO to get this to work. So this feels like maybe something that's maybe like one hop out, but still close enough that you can take data from one model and use that data to teach another model to do something. Thank you. Yeah, thanks. Yeah, kind of. I'll try to keep this like on the detail, like concrete level. It's about the abstraction piece. So your example is about code refactoring, right? So do you think that this same technique or some aspect of it, I should phrase it differently, is the intention that this kind of technique can be itself abstracted to other domains? So instead of refactoring code, you're refactoring something else in the process of developing a skill. Yeah, I think so the refactoring part, I think is somewhat code specific because the idea is that you need to be able to kind of run, you need to be able to execute both before and after and make sure that the results match, right? So if you have kind of, like if you refactored your code correctly, you'll know because nothing changes, right? Like the output that you got before is the same as the output you got after. If you did something wrong, then you'll get a different result. And that's sort of the verification signal here. That's what's telling you whether the abstraction that you learned was actually a good abstraction or not. I think if you don't have that, it becomes more challenging to learn abstractions in this way by just kind of like rewriting. But I think it's still doable, right? It's still like, it just has to be done with a different kind of verification signal. So one thing we did actually do in the paper that I didn't show results on is that we have some reasoning domains as well. So we have like math reasoning where you can express the, you can read. It's pros or pros? I mean, his, we'll wait to see if he comes back on. It's possible like maybe his,