AGI-assisted Alignment

1. Getting from an unaligned AGI to an aligned AGI?
2. Making it harder for an AGI to "trick" us, with STVs
3. Alignment with argument-networks and assessment-predictions

Getting from an unaligned AGI to an aligned AGI?

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

Summary / Preamble

In AGI Ruin: A List of Lethalities, Eliezer writes *"A cognitive system with sufficiently high cognitive powers, **given any medium-bandwidth channel of causal influence**, will not find it difficult to bootstrap to overpowering capabilities independent of human infrastructure."*

I have larger error-bars than Eliezer on some AI-safety-related beliefs, but I share many of his concerns (thanks in large part to being influenced by his writings).

In this series I will try to explore if we might:

- Start out with a superintelligent AGI that may be unaligned (but seems superficially aligned)
- Only use the AGI in ways where it's channels of causal influence are minimized (and where great steps are taken to make it hard for the AGI to hack itself out of the "box" it's in)
- Work quickly but step-by-step towards a AGI-system that probably is aligned, enabling us to use it in more and more extensive ways (as we get more assurances that it's aligned)

From the AGI-system we may (directly or indirectly) obtain programs that are interpretable and verifiable. These specialized programs could give us new capabilities, and we may trust these capabilities to be aligned and safe (even if we don't trust the AGI to be so). We may use these capabilities to help us with verification, widening the scope of programs we are able to verify (and maybe helping us to make the AGI-system safer to interact with). This could perhaps be a positive feedback-loop of sorts, where we get more and more aligned capabilities, and the AGI-system becomes safer and safer to interact with.

The reasons for exploring these kinds of strategies are two-fold:

- Maybe we wont solve alignment prior to getting superintelligence (even though it would be better if we did!)
- Even if we *think* we have solved alignment prior to superintelligence, some of the techniques and strategies outlined here could be encouraged as best practice, so that we get additional layers of alignment-assurance.

The strategy as a whole involves many iterative and contingency-dependent steps working together. I don't claim to have a 100% watertight and crystalized plan that would get us from A to B. Maybe some readers

could be inspired to build upon some of the ideas or analyze them more comprehensively.

Are any of the ideas in this series new? See here for a discussion of that.

***Me*:** I have some ideas about how how to make use of an unaligned AGI-system to make an aligned AGI-system.

***Imaginary friend*:** My system 1 is predicting that a lot of confused and misguided ideas are about to come out of your mouth.

***Me*:** I guess we'll see. Maybe I'm missing the mark somehow. But do hear me out.

***Imaginary friend*:** Ok.

***Me*:** First off, do we agree that a superintelligence would be able to understand what you want when asking for something, presuming that it is given enough information?

***Imaginary friend*:** Well, kind of. Often there isn't really a clear answer to what you want.

***Me*:** Sure. But it would probably be good at predicting what looks to me like good answers. Even if it isn't properly aligned, it would probably be extremely good at *pretending* to give me what I want. Right?

***Imaginary friend*:** Agreed.

***Me*:** So if I said to it *"show me the best source code you can come up with for an aligned AGI-system, and write the code in such a way that it's as easy as possible to verify that it works as it should"*, then what it gave me would look really helpful - with no easily way for me to see a difference between what I'm provided and what I would be provided if it was aligned. Right?

***Imaginary friend*:** I guess I sort of agree. Like, if it answered your request it would probably look really convincing. But maybe it doesn't answer your question. It could find a security vulnerability in the OS, and hack itself onto the internet somehow - that would be game over before you even got to ask it any questions. Or maybe you didn't even try to box it in the first place, since you didn't realize how capable your AI-system was getting, and it was hiding its capabilities from you.

***Imaginary friend*:** Or maybe it socially manipulated you in some really clever way, or "hacked" your neural circuitry somehow through sensory input, or figured out some way it could affect the physical world from within the digital realm (e.g. generating radio waves by "thinking", or some thing we don't even know is physically possible).

When we are dealing with a system that may prefer to destroy us (for instrumentally convergent reasons), and that system may be orders of magnitude smarter than ourselves - well, it's better to be too careful than not paranoid enough..

Me: I agree with all that. But it's hard to cover all the branches of things that should be considered in one conversation-path. So for the time being, let's assume a hypothetical situation where the AI is "boxed" in. And let's assume that we know it's extremely

capable, and that it can't "hack" itself out of the box in some direct way (like exploiting a security flaw in the operating system). Ok?

***Imaginary friend*:** Ok.

***Me*:** I presume you agree that there are more and less safe ways to use a superintelligent AGI-system? To give an exaggerated example: There is a big difference between "letting it onto the internet" and "having it boxed in, only giving it multiplication questions, and only letting it answer yes or no".

***Imaginary friend*:** Obviously. But even if you only give it multiplication-questions, some other team will sooner or later develop AGI and be less scrupulous..

***Me*:** Sure. But still, we agree that there are more and less safe to try to use an AGI? There is a "scale" of sorts?

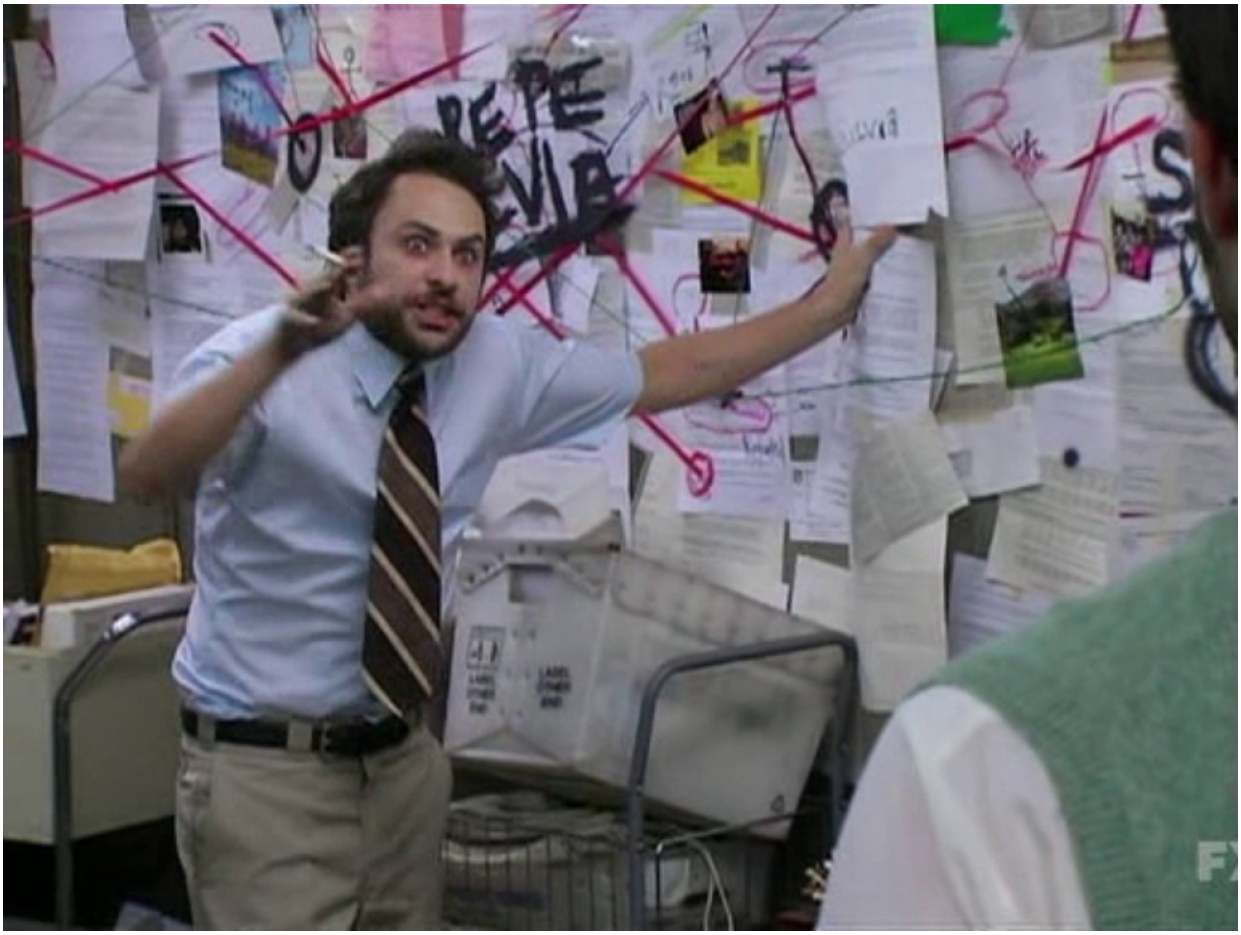
***Imaginary friend*:** Of course.

***Me*:** Would you also agree that there is a "scale" for how hard it is for an oracle/genie to "trick" you into falsely believing that it has provided you with what you want? For example, if I ask it to prove a mathematical conjecture, that is much harder to "pretend" to do the way I want it without actually doing it (compared to most things)?

***Imaginary friend*:** Sure.

***Me*:** What I want to talk about are ways of asking an AGI genie/oracle for things in ways where it's hard for it to "pretend" that it's giving us what we want without doing it. And ways we might leverage that to eventually end up with an aligned AGI-system, while trying to keep the total risk (of all the steps we take) low.

***Imaginary friend*:** My system 1 suspects I am about to hear some half-baked ideas.



***Me*:** And your system 1 may have a point. I don't claim to have detailed and watertight arguments showing exactly how we get from A to B. What I have is an *outline* of how we might get there, while minimizing risk along the way (not to 0%, but to way less than 50% if suggestions are executed faithfully).

Believe me, I don't have full knowledge and understanding of everything that has been explored in the AI alignment community…

***Imaginary friend*:** I believe you.

***Me*:** …but it seems to me that the kinds of techniques and strategies I'll be outlining are under-discussed.

***Imaginary friend*:** It sounds like you are looking for ways to avoid having to solve the alignment problem, so that you can focus on other things instead of solving it beforehand.

***Me*:** I guess it can be interpreted that way. I don't want it to come across that way though! If we find reliable methods to align a superintelligent AGIs *before* we get superintelligent AGIs, then that would improve our odds of success!

But maybe we don't. If we don't then it's better to have some plans ready, so that less improvisation is necessary. AFAIK there is no fire alarm for AGI, and it certainly doesn't seem too early to start analyzing these kinds of scenarios in a more comprehensive way..

And also, even if we *think* we have solved alignment in a reliable way prior to superintelligence, some of the strategies and principles I'll discuss can be used as additional layers of alignment-assurance. Hence me thinking that it may be a good idea to develop these ideas further, and to encourage them as best practice.

Restrictions in expressivity of AI

***Me*:** The first part of this discussion will focus on things that we can do while the AGI stays in the digital realm. We will get to the question of how to do complicated things in the non-digital realm (and on the internet), but that's for later.

***Imaginary friend*:** You understand that any use of an AGI enables it to affect the non-digital realm, right? If you let it write or modify code, that is a way for the AGI to communicate with humans, and communicating with humans is a way to affect the non-digital realm.

***Me*:** Yes, I agree with that. But I would point out that it is up to us how much "expressivity" we give the AI-system. For example, if we get help from it to point out potential mistakes in some piece of code, maybe we don't let it write text to explain itself (limiting instead to things such as pointing out areas of code to take an extra look at, highlighting examples of input that may produce unintended behavior, etc). And if we do let it write text, then it is up to us how we limit the expressivity of what it can write in text.

***Imaginary friend*:** Ok, did you by the way notice that you spoke about not letting it write text - but letting it write example input? Well, example input can be text! Those kinds of things can be easy to miss, and often you may not be aware of what you're missing.

***Me*:** I may miss such things sometimes, but sometimes I will also leave them out even though I'm aware of them, so as to not get into tangents all the time. I did notice that one though. Partly thanks to you helping me to get better at noticing those sorts of things.

***Imaginary friend*:** That's what imaginary friends are for.

Keep in mind that *any* way to point out bugs or potential problems is a way for the AI to affect how you think and what you think about. Take this test of selective attention for example. Or look at how magicians focus and misdirect people's attention to trick them. There are lots of things like this.

***Me*:** I'm certainly not claiming that the techniques I'll describe are risk free. At the same time, it's important to consider how different techniques can come together to reduce risk. For example, one thing that will be touched upon again and again is testing the convergence of the results from different methods and architectures and pieces of code. Preferably, if an AI-system writes the code for some module, then the overall system should ensure that we also have that module be written by other AI-systems, and also by humans if possible.

In regards to e.g. AIs pointing out potential issues, the system as a whole (a system of systems of systems..) would preferably keep track of the changes made due to such suggestions, and also try to keep track of what the output of the system would be if those changes hadn't been made.

***Imaginary friend*:** Ok. But just because something makes you more safe, and you can handwave to a lot of techniques where each thing helps a bit with safety - this does not show that the the process as a whole will be safe. And I am a bit worried about you proposing ideas that are hard to criticize because they're complicated and vague, instead of getting to the "core" of the problem. Also, adding on lots of complication to increase security can in some cases be counter-productive.

***Me*:** Some good points there. Although do I feel uncertain about the degree to which there is a "core" of the problem. I want this series to outline a path towards a more "complete" solution, but I want to start by focusing more on techniques and principles that are "building-blocks".

The best scenario is if the first superintelligent AGI is robustly aligned from the start. I think we should try hard to make that be the case, and I don't hope this series is much of a distraction from that.

I am reminded of how I am in support of trying to reduce CO2 emissions today, but *also* support research on geoengineering.

Topics I'll cover in this series

***Imaginary friend*:** So where do you see this discussion going?

***Me*:** I guess we'll see. But I do have some plans.

One thing I want to discuss is the concept of "solution-generators" (and maybe also "solution-generators-generators", etc). When asking an oracle/genie for something, it may often make sense to not ask the question directly, but to instead ask for a "solution-generator" - that is to say, a function that generates solutions within the domain in question. These "generators" should be optimized for being narrow/specialized/modular/verifiable (techniques for trying to ensure that the system is faithfully optimizing the "generator" for these properties is a topic by itself).

For example, if we want to do a physics-simulation, then we might want to use a "simulation-generator" or "simulation-generator-generator". One merit of this is that the code can be easier to inspect and understand than the code of a superintelligent AGI. Another merit is that the "generator" can be tested against existing data.

But of course, you can also ask it to answer the question directly (in addition to asking for the "solution-generator"). And then you can see if the output is the same when you obtain it indirectly vs when you obtain it directly.

***Imaginary friend*:** Isn't it better to not ask it the specific question, so that it's harder for it to guess what you're after?

***Me*:** I will mostly not be mentioning this sort of thing explicitly, but yes I agree, you shouldn't ask both questions of the same *instance* of the AGI-system. The instance that is asked for *A* should not know that you also have asked it for *B*. Though of course, you could have additional instances that are asked both questions, and check if results/output converge.

Another topic I want to speculate a bit about is the topic of brain emulations, and also other "generators" that try to predict the behavior and thinking of humans. I'm interested in the question of *"Can we bootstrap from AGI to either brain emulations*

and/or something else that can predict human behavior, while more or less staying within the digital realm?". Obviously there are lots of ethical considerations here, and it's very important to avoid suffering subroutines insofar as possible!

***Me*:** I also want to discuss techniques involving computable argument-trees/"proofs", where every inference-step is made explicit. As well as outlining possible techniques to have the concepts/propositions of such proofs represent more or less any thought that is sufficiently "clear/crisp/explicit" (blurring the distinction between "mathematical" proofs and any other argument about anything). Included in the discussion will be outlines of ideas for how to deal with "vagueness" and cluster-like concepts within such argument-trees/"proofs".

And I'll be outlining thoughts about capabilities that I think will help with verifying that instructions for doing things in the real world (developing new types of machines and that sort of thing) will work as intended. Such as for example copying a strawberry at the molecular level without unintended consequences. Among other things there will be some focus on "generators" for mappings between (1) models/ontologies, and (2) data-structures representing geometric structures (e.g. some sort of physics-simulation), and (3) real things in the actual world that the models are meant to refer to.

The more people there are who (1) are smart and have thought a lot about something and (2) see things differently from you, the more reason for self-doubt about your own judgment. And this is for me a significant source of uncertainty about my ideas in regards to alignment (and AI more generally). But it seems best to me to just try to describe my perspective as well as I can, and then people can do with that what seems best to them.

***Imaginary friend*:** Talk to you later then.

Any feedback or comments (be that positive or negative or neither) would be received with interest.

Making it harder for an AGI to "trick" us, with STVs

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

Summary / Preamble

AI Alignment has various sub-areas. The area I focus on here is ways we might use a superintelligent AGI-system to help with creating an aligned AGI-system, even if the AGI we start out with isn't fully aligned.

Imagine a superintelligence that "pretends" to be aligned. Such an AI may give output that *seems* to us like what we want. But for some types of requests, it's very hard to give output that *seems* to us like what we want without it *actually* being what we want (even for a superintelligence). Can we obtain new capabilities by making such requests, in such a way that the scope of things we can ask for in a safe way (without being "tricked" or manipulated) is increased? And if so, is it possible to eventually end up with an aligned AGI-system?

One reason for exploring such strategies is contingency planning (what if we haven't solved alignment by the time the first superintelligent AGI-system arrives?). Another reason is that additional layers of assurance could be beneficial (even if we *think* we have solved alignment, are there ways to relatively quickly add additional layers of alignment-assurance?).

When dealing with a genie/oracle, we may not want to ask it to provide some direct solution/answer to what you want (and have it argue why it's a good idea). Instead we obtain what we want more indirectly, by having different instances of the genie/oracle aid us in the construction of more narrow/specialized programs (that score high in terms of verifiability).

We could call such programs STVs, where STV is an abbreviation for ***S**pecialized **T**ransparent **V**erifiable Program*.

STV is a blurry concept (much like how *AI* and *AGI* are blurry concepts).

An STV could be (but would not have to be) a narrow AI. More colloquially, STVs could also be referred to as "generators".

In this post I explore principles/techniques/strategies for using STVs in ways that enable us to get things we want from an AGI without being "tricked".

I hope and think this article reads well in isolation, even though it's part 2 of a series.

Future posts in this series will include discussion of:

- AI-generated computational proofs/argument-trees, and methods/strategies for verifying such proofs/arguments.
- Ideas for how formalisms for computational proofs could try to incorporate human cluster-like concepts, concept vagueness, formalism-

to-reality mappings, references to code and results from running code, etc (from within themselves).
- How different “pieces” outlined in this series might be put together. Can we get from unaligned AGI to aligned AGI without being "tricked"/hacked/manipulated along the way, and without being stopped by chicken-or-egg problems?
- Ideas for how we might want to make a system of sub-systems, consisting of various "siloed" AGI-systems (with different settings, architectures, and alignment methodologies).
- Strategic and moral considerations, and possible first steps when starting to use superintelligent AGI-systems to do things outside of the digital realm.

Are any of the ideas in this series new? See here for a discussion of that.

Me: I think it may be helpful to start out with a simplistic example.

So let’s imagine that we are interested in finding good chess-moves, and want an AGI to help us with this.

Imaginary friend: Ok.

Me: We *could* ask it directly for good chess-moves, along with explanations for why these chess-moves seem like good chess-moves.

But if we want to make it hard for the AGI to “trick” us (to pretend that it’s giving us what we want, but not actually doing so). One way could be to ask for a *program* that searches for good chess-moves (instead of asking directly for the moves themselves).

This could be beneficial for several reasons:

1. If the AGI wants to trick you in specific ways for specific strategic reasons, this may be harder when you force it to answer in a general way (not answer in a way that is custom to the specific thing you want).
2. If the AGI has strategic reasons for wanting to trick you about specific things, this is made harder when you don’t ask it specifically for what you are after (thus making it less straight-forward to guess what the answer will be used for).
3. Code that is specialized for some narrow/constrained task can often be easier to understand. This presumes that the AGI chooses to write the code easy to understand for humans, but if we know it is superintelligent and it’s *pretending* to be aligned then there are limits to what it can get away with (we would notice if the code clearly isn’t optimized for human understandability).
4. It may be easier for humans to understand chess programs than analyzing chess-moves directly (presuming that the program is written in a way that favors human understanding).
5. Narrow/specialized programs may more often be possible to code in a way where properties of behavior/output can be “proven” using formal/computational proofs. To put it simplistically: The more narrow a program is, the more suspicious it would be if the AGI can’t “prove” that it does what we want it to do.
6. Programs that solve problems in a general way - but within a narrow/constrained domain of tasks/questions - can more easily be tested against examples and real-world data. In the case of a chess-program we could for example check if it beats other chess-programs while using less computation.

Imaginary friend: Ok, but being able to confirm that a chess program is good at chess doesn't solve the alignment problem. We want to be able to use the AGI to deal with the real world. And the real world is messy and complex in ways that board games aren't, making verification and formal "proofs" much harder.

Me: Sure, I agree with that. I started out with a simplistic example, so as to be able to outline some principles without getting bogged down in details. And more principles and ideas will be conveyed later on. How far this gets us remains to be discussed.

Clarifying what I mean by STVs (aka "generators")

Me: The term STV is a term I made up. Maybe other people have referred to similar concepts using different terms.

It stands for *"**S**pecialized **T**ransparent **V**erifiable Program"*.

By *specialized* I mean that the range of questions/tasks it is designed to handle is constrained. For example, maybe it only does one type of task, and only handles input from a specific domain and in a specific format.

In some cases the program may work well without anything that resembles reasoning. In other cases, reasoning-like processes may be necessary. In such cases the reasoning should (insofar as feasible) be constrained, specialized/non-general, transparent/interpretable, and closed off from other components/subroutines.

By *transparent* I mean that it's easy for humans to understand how it works. Maybe not easy in an absolute sense, but no harder than necessary.

Insofar as feasible the code should be modular, and written in a way that is optimized for making it easy for humans to understand (and hard for humans to misunderstand).

There should be additional tools for analyzing the code beyond looking at source code. Diagrams, high-level textual descriptions, and maybe interactive overviews of various kinds.

If the program needs components that are more "black-box-like" (like neural networks sometimes can be), then these components should be constrained (in terms of how they can influence the rest of the system, and in terms of what they are used for).

By *verifiable* I mean that the behavior of the code can be verified. And much like the other properties of STVs, the property of being *verifiable* is not binary/either-or. It's more of a scale.

One example of verification is formal/computational proofs. The best case is if good computational proofs can show that the program as a whole does what it's supposed to do. If that's not possible, proofs may still help with other things. Such as proving the behavior of subroutines. Or proving properties that the program reliably will approximate by some margin.

Sometimes it may be possible to construct proofs about whether or not it's possible to prove that the system does what we want.

Another example of verifiability is tests that should be satisfied by functions/modules. Concepts such as test-driven development are well-established among humans, and a superintelligent AGI-system could be expected to write much more comprehensive tests than a human.

And another important example of verification is to test the system as a whole against examples and/or real-world data. For example:

- If the STV approximates physics somehow, then it could be compared against real-word data.
- If the STV predicts some aspects of human behavior, then it could be tested against actual data from actual humans (which the AGI-system that created the STV didn't have access to).
- If the STV generates proofs of some kind, then we can test if it is able to reliably find proofs that already exist (but which the AGI-system that created it didn't necessarily have access to).

"Hidden" behaviors in STVs

Imaginary friend: An unfriendly AI may create STVs in such a way that they pass our verification-process, but still have unfriendly behaviors "hidden" inside them.

Such "hidden" behaviors may be subtle and hard to find. They can't necessarily be "pointed to" in some specific lines of code.

Do you imagine that we can guarantee against such "hidden" behaviors?

Me: I agree that this is an important concern. The degree to which we can guarantee against this - or at least make it unlikely/implausible - is from my perspective an open question (that may be worthy of further thinking/analysis).

But this text touches upon several techniques and principles that I think can make it harder for an AGI-system to "hide" things.

One way to make it harder to "hide" misbehavior is to look for ways that the STV can be made more general. To see why that potentially could be helpful, you could imagine a calculator. A calculator is specialized in the sense that it only does arithmetic. But if a calculator has special-purpose circuitry for handling 237+418 - or special-purpose functionality for handling multiplication using numbers that are within some specific range - well, that would be suspicious.. I'm giving a simplistic example here, but the same principle should also apply to more complex cases.

So we could have STVs that look for ways other STVs can be made more general/simple.

Another way to make it harder to "hide" misbehavior is to not let the AGI itself do the implementation of the sub-modules, but instead make other STVs do the implementation based on high-level specifications.

Also, it may in some cases make sense to have STVs that generate other STVs. That is to say, if an STV can be described as an "answer-generator", then an STV that outputs other STVs could be described as an "answer-generator-generator". This is an even more indirect way of obtaining what we want from an AGI, and I'll give some concrete examples of this later.

Another thing that makes it harder to "hide" things is if the same functionality is to be implemented in different ways, based on varying principles/architectures (since it's more challenging to make different implementations fail in the same way - even more so if it's not the same AGI that makes both).

Why-not-bothing & output-convergence

Me: As a general principle, the more reasons we have to believe something, the more confident we can feel.

For example, mathematicians may feel more confident in a theorem if it has been proved in several different ways (with varying proof-methods and starting assumptions).

If there are 2 different alignment-methodologies that seem promising, then maybe we would want to use both as oracles/genies side by side, and see if they give the same answers. And if there are 3 different alignment-methodologies that seem promising, or 4 - well, you get the picture..

At almost every level of a system - from the top level to detail-oriented sub-modules - it may often be helpful to try different architectures/implementations/principles, and see if the output is the same.

We might sometimes refer to this as the principle of *"why not both?"*:

Here are some examples (not a complete list):

- If we tell an AGI to make an STV for some task/purpose, then the AGI may come up with various different principles and architectures that such an STV could be based on. In such cases, we may want all the STVs, so that we may use them as one (running them side by side, and seeing it as suspicious if they don't produce converging output).
- Should AGI-systems have access to the history of requests it has been given earlier, or should it be reset before new requests are asked? I think the second often is more advisable, but sometimes it may make sense to try both.

- Should we give AGIs lots of background-knowledge that may be helpful for the task (but also may be helpful for tricking or manipulating us)? Or should we give it as little information as possible? Or maybe something in between? A good answer may often be yes, yes and yes (we can run different instances of the same AGI-instance).
- Who should make any given STV? And for the sub-modules in the architecture of some STV, who should make those? Often it is a good idea to have several implementations from several systems (with varying degrees of human involvement). And then we can see if outputs converge (or if one implementation can be proven to behave the same as another in a "mathematical proof"-like way).
- When we wonder about something, do we want to ask the AGI directly, or do we want an "answer-generator", or maybe an "answer-generator-generator"? Sometimes the answer may be *"yes, yes and yes"*.

This kind of thing adds complexity, but with the help of a capable AGI-system it should be possible to set up systems that can help us handle such complexities.

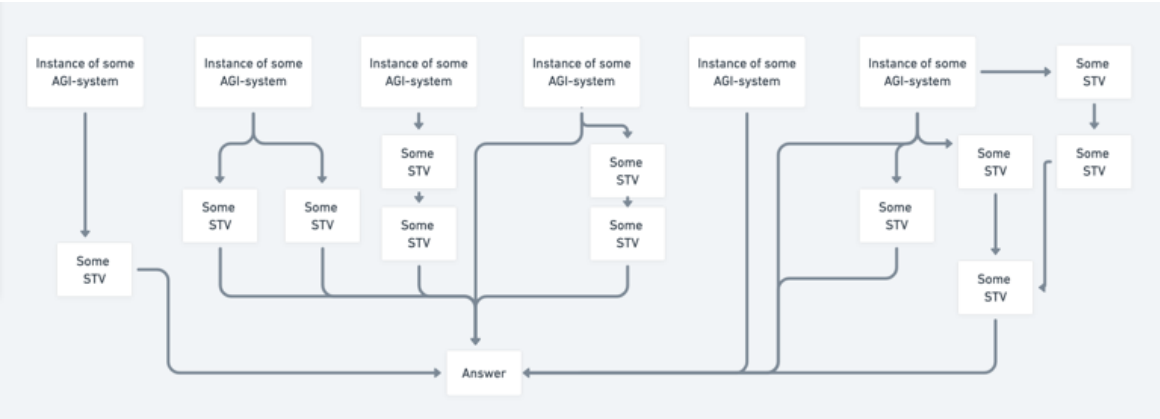


Diagram showing how the same answer is obtained via various direct and indirect routes. It may at times be prudent to have orders of magnitude more routes than is shown in this diagram.

Another recurring theme in this text is indirection and subroutine reuse. If we let the AGI answer directly, and build whole systems by itself in one swoop, then it is for various reasons easier to "hide" misbehavior in the output. But when the AGI outputs smaller "pieces" of functionality that do some specialized thing, and can be put together in various ways to do various different things, then this can give more opportunities to notice if some of the "pieces" don't always do what they're supposed to do.

If an AGI is asked to output systems that are optimized for verifiability, and the systems it suggests don't exploit these principles, then this may be a good cause for suspicion!

The space of all claims and inference-steps that are true is huge, and it's deeply interconnected. And the more of this network you need to account for, and the more "nodes" in the network that can be verified, the harder it is to get away with false claims without contradicting yourself (especially when the subset of the network you account for is dense). More comprehensive and systematic ways of leveraging this principle is one of the things that will be explored in part 3 of this series.

Human-predicting STVs

Me: One thing STVs maybe could be made to do is, is to predict human responses (what a human would think of some argument, how a human would evaluate some piece of code, etc).

Imaginary friend: Aren't STVs supposed to be "narrow" though? Humans are in a sense AGIs.

Me: I agree that this makes it more of a challenge to obtain STVs that predict humans (while remaining transparent and verifiable).

Imaginary friend: But you still think that we - with the help of an AGI - could obtain STVs that predict human responses? And that we to a sufficient degree could verify that such STVs actually do what we want them to?

Me: It seems likely to me that we could. But it also seems plausible that we wouldn't be able to.

Keep in mind:

- There are degrees of success. For example, sometimes we may be only 90% confident that an STV works as it should. In such cases, whether we should use it depends a lot on context/specifics. If it is a component in a larger system, then there may be ways to use it where it only can help (in certain instances, if it works), and doesn't have much opportunity to do damage.
- Human-emulating STVs would not need to always have an answer. For example, if an STV has the job of predicting how a human would categorize something, we could accept that it sometimes isn't confident enough to make a prediction.

Imaginary friend: How would an STV predict human behavior though?

Me: Here are some ideas:

Brain emulations

It's sometimes pointed out that human brain emulations could have large advantages in terms of alignment, but that the first AGIs are unlikely to be brain emulations. But might we have a potentially unaligned AGI help us obtain brain emulations in a safe+verifiable way, without letting it "leave the digital realm"?

The outputs of brain emulations can be tested against real-world data (data from brain scanning and so on), and the emulations can also be tested for how good they are at predicting the actions/answers/etc of humans (and other animals).

STVs that emulate brains need not be given to us directly from an AGI. There would be various other options (that aren't mutually exclusive). Could it, for example, be possible to generate an STV that generates brain emulations based on the DNA of the animal in question (but without being given any direct info about brains)? Might the functionality of such an STV be made in a very general-purpose way (where it's hard to "hide" details)? Might the functionality used to indirectly generate brain emulations also be used to generate other things, which may be verified (predictions regarding the inner workings of the gut, simulations of the details of the inner workings of a flower, etc)?

One dilemma in regards to simulations is how fine-grained they are, and how they handle a model where the details are unclear (they simulate something that exists in the real world, but they are not given precise and accurate data of starting conditions). This is not just a dilemma for brain simulations, but for simulations of any physical system. Something we may want is a system that

gives an accurate description of the *range* of possible outcomes, given a description of the *range* of possible starting conditions. And we want the possibility for the simulation to not spend lots of computation on details we don't care about (only computing details that are useful, or that are helpful for verification of simulation). Since these are general-purpose challenges, which aren't specific to brain simulations, we may want to have STVs that can help generate "simulation-approximations" for any physical system. That way we can *also* test if they do a consistently accurate job when used to make predictions about other physical systems (and not *only* be tested for whether or not they do a good job with brain emulations).

When doing anything that resembles emulating a brain, it is very important to avoid/minimize risk of suffering subroutines! Failing at this could result in mind crimes and suffering, potentially at an enormous scale!

At every step of the process we should:

1. Avoid simulations that might be conscious.
2. Avoid simulating processes that would be likely to experience significant suffering if we were wrong about #1.

Subroutines with positive valence may often be unproblematic, or even a good thing. But it remains to be seen how good our understanding of consciousness will become (the consequences of assuming wrongly can be very bad!).

Approximations of brain emulations (sometimes based on indirect methods)

In a sense any brain emulation can be seen as an approximation of a more high-fidelity emulation, but what I mean here is that large components of the STV need not be based on "emulation" at all, as long as the STV predicts aspects of brain states + what the human answers/does.

In a sense, knowing what the human says/does may be all we are interested in, but if it makes predictions about brain states then this may make verification easier (especially if the STV is based in part on assumptions about which brain states follow from which, which actions correspond to which brain states, brain state sequences that cannot happen, etc).

Lots of raw and somewhat "hand-coded" probabilistic and modular inference-rules that encode typical human responses within some domain/context

Inference-rules could reference results from subroutines that use neural nets, but if so we should probably require that we can verify what aspects of the thinking/work is done by the neural net. Maybe bayesian networks, or something similar, could be part of the system somehow. As much as possible of the system should be symbolic/interpretable.

Imagine if a smart/cooperative human sits in a room, and is given simple multiplication-questions. I would guess that in such a situation we would not need high-fidelity brain-emulation to predict the humans "output" - a calculator could suffice! This simplistic example could work as a weak "existence-proof" of sorts, showing that in restricted situations/domains, the most probable human output can be predicted without using brain emulations.

But whether this can be done in a verifiable way for useful tasks is AFAIK an open question.

It's sometimes pointed out that it's infeasible to "hand-code" what we mean by various fuzzy concepts (such as "person", "human", "animal", "dead", "happy", etc). But even if that's infeasible for *us*, it's not necessarily infeasible for a superintelligence. And if a superintelligence hand-codes it, there may be ways of verifying that the hand-coded specification does a good job of mimicking human output.

The AGI would not necessarily do the "hand-coding" directly itself. It could output STVs that do the "hand-coding" (based on various methods). Some such STVs might do the hand-coding based on being given books and internet archives, and building a model of human concepts from what they read/hear/see.

If we end up in a situation where an AGI can make STVs for us that predict human behavior, we might wish that we had more experimental data to test those STVs against. That's one example of why it may be useful to plan in advance!

For example, if we think we might want to use human-mimicking STVs to evaluate proofs/arguments provided by AGIs/STVs, but in a piecemeal fashion, then it might be helpful to think ahead of time about what the smallest components/steps of such proofs/arguments (that can be evaluated in isolation) might look like.

And if we want STVs that for example mimic humans looking over code, then that is also something that may be helpful to plan for in some detail.

Some STVs may be easier to verify if we have brain state data of humans that do the exact same types of tasks that the STVs emulate humans doing (e.g. from MRIs). Sometimes the STVs may emulate people sitting in a similar room as in the experiment, in front of a similar computer to the one in the experiment, etc.

STVs should be able to describe patterns about how various brain states correspond to both actions (answers, code that is written, etc) and other measurements (posture, eye movement, milliseconds between keystrokes, mouse movement, brain state measurements, etc). Preferably these patterns should be as general as possible (e.g. not just for people with red hair sitting in rooms with yellow pain when the room is 35°C).

The more experiments we have, and the data we have from experiments (mouse movement, eye movement, video of posture, brain measurements, etc), the more challenging it may be for an STV to "make things up" (without this being discovered when predictions are tested against existing data).

It may also be helpful to have additional data about humans who participate in experiments (with the informed consent of participants, of course). Their DNA, bodily features, their gut microbiome, etc.

Often it's not the average human that we want STVs to predict, but rather humans who are usually high in intelligence and cognitive reflection (and are talented at what they do).

STVs that help with software-development

Me: Another thing STVs could help with is software development. Here are some examples:

Rewrite code in ways that are proven to not change behavior

It may be relatively tractable to prove that two pieces of code behave similarly and always will have the same output (and to verify such proofs).

If you are a programmer, you know ways to predict that a code-modification won't change output. For example, you know that an if-else statement could be replaced by a switch-statement, or that a+b can be replaced with b+a when a and b are numbers. Sometimes you make mistakes when doing this type of reasoning, but this doesn't mean that proofs that use similar reasoning are impossible (you sometimes make mistakes when doing math as well, but that doesn't make it impossible to construct mathematical proofs!).

These kinds of proofs could be computational, meaning that to mechanically check the proofs would be relatively trivial. And all that is needed to show that a given proof is wrong is one counter-example (2 pieces of code that are "proven" to have the same output/behavior, but have different output/behavior when we run them with some specific input). Such a counter-example would not only invalidate that specific proof - it would be a cause for questioning the proof-system itself (and whoever made it). The better and more extensively a proof-system has been tested, the better.

Reasons for rewriting code, and proving equivalence between pieces of code, could include:

- Rewriting code to be more computationally efficient
- Rewriting code so as to score higher in terms of how easy it is for humans to understand it
- Using these kinds of proofs as building-blocks in other proofs

Use code-rewrites with proofs as building-blocks in other proofs

One example could be proofs showing specifically and precisely how the behavior of two pieces of code are different. Another example could be proofs showing how one piece of code approximates another piece of code.

Here are some examples where I try to allude to what I have in mind (handwavy, but still dense, so feel free to skip):

- *"Function A has equivalent output to function B, except for when the input is in range C, in which the output of A is equivalent to the output of Function D when D is given output from B as input"*
- *"Using search-procedure A we can't find any computationally efficient way of choosing input for Function A such that the output from A doesn't approximate the output of Function B, with the probability-distribution that describes this approximation being within probability-distribution-space C"*
- *"Among the space of allowable input for function A, there is no sub-space of size larger than B where the output will approximate function B according to approximation-description C, with the exception of input-spaces for which function A always will return an error"*

Convert between code and high-level descriptions/specifications of

code (and look for discrepancies)

What a piece of code is supposed to do can be described at various levels of specificity, and in various ways:

- You can describe what it does with text
- You can make diagrams
- You can give specific examples of what the output is (or should be) given specific input
- You can list things that should be true about the output (either all the time, or presuming certain things being true about the input)
- You can have various interactive tools that let you explore the specification of what a piece of code does / is supposed to do. Letting you see step by step how various inputs would be processed, and letting you “zoom” in and out of modules/subroutines.

When you have STVs that can convert between code and more high-level descriptions of code, one thing you can do is to search for code with significantly different functionality/behavior that would generate the same high-level description. This could potentially be helpful in finding out how some description is underspecified in ways that matter.

STVs like this could be used to make it harder for other systems (STVs, AGIs, etc) to “hide” details in code that they write. It may be harder to "hide" details when you don't implement everything yourself.

There need not be only one STV that implements a high-level description. Different STVs could implement the code, and the code they output could be compared (by testing if output converges, and by making proofs about how the code-bases are equal/similar/different in terms of behavior).

We could make “loops” where we convert from high-level descriptions, to code, to high-level descriptions, to code, etc - with each step being done by different/independent STVs. And then we can see if we get “drift”, or if we get more or less equivalent code-bases and high-level descriptions again and again.

Look for bugs in code

Some bugs are bugs in a very unambiguous sense, making them clear targets. Other bugs are more subtle and dependent on nuanced human-like judgements.
I wouldn’t expect some very non-AGI-like SVG (that understands code and nothing else) to robustly be able to find all bugs. But probabilistically it could help. Having SVGs based on various principles/architectures/settings that search for bugs could make it much harder to “hide” bugs.

Help to write/propose tests

When specifying what code should do, it may be helpful to write tests that can detect examples of code doing what it should do and examples of code not doing what it should do. This can be done by humans, but STVs may also suggest tests.

Look for structural similarities/overlap between different code bases

These STVs could search for ways to rewrite the different code bases so as to minimize the amount/size of non-overlapping modules.

A somewhat similar task would be to give the STV access to existing code-bases/libraries while it codes, and have it try to maximize overlap with these existing code-bases/libraries (as one of the things it maximizes for).

Another somewhat similar task would be to be given a code base that does one thing, and search for the minimal amount of changes that could be made so as to make that code-base do another thing. (For example, the minimal amount of changes to convert a narrow AI that does one thing into a narrow AI that does some other thing.)

Overlapping modules can more easily be tested in an extensive way. Especially if the STVs that rely on the modules vary in terms of how easy it is to verify if they do a good job or not.

When code bases are written/rewritten so as to share code, what we want to "minimize" is not necessarily "*amount of code that isn't shared*". Instead, we may to want to minimize "*functionality that is hard to verify to a comprehensive degree*". That is to say, we may want to solve some problem by compositioning it in terms of other problems - even if this leads to more complexity or computational inefficiency - if doing so can be helpful from the perspective of verification.

Identify parts/aspect of code that don’t only significantly affects output small sections of the space of possible inputs

Are there ways to rewrite code that would not change output for the great majority of inputs, but would change output for a few inputs?

Are there ways to simplify code that *would* change output for most input, but would *not* significantly change the average score of the output (as ascribed when it’s evaluated)?

If efficient ways can be found of identifying such changes (when they exist in the space of possible changes), then maybe this could be used to make it harder to intentionally “hide” malicious behavior in STVs.

If we call the part of input-space where output can be verified A, and the part of input-space where we can’t verify output B, are there ways to rewrite the code (without making the code-base more extensive) that wouldn’t affect output within A negatively, but would have a significant effect on some of the output within B? If so, that may often be a reason to have less trust in that STV (and the system that produced it). Especially if the modified STV and the non-modified STV could be made to produce contradicting output.

Scope of what STVs could be used for (without becoming too AGI-like)

Imaginary friend: If it is hard to verify that an AGI does what you want, then presumably the same will be the case for STVs the more similar they are to an AGI?

Many tasks seem to be more or less AGI-complete, which presumably means that it would be hard for an STV to do them without being AGI-like.

How broadly capable do you think an STV can be while still being relatively *narrow/specialized*, and scoring well in terms of *transparency* and *verifiability*?

Me: I don't have any arguments that are watertight enough to justify opinions on this that are confident and precise. But my gut feeling is somewhat optimistic.

Imaginary friend: One thing you should keep in mind is the limits of symbolic reasoning. Earlier in the history of AI, people tried to make expert systems that rely heavily on "neat" symbolic reasoning. But these systems were largely unable to deal with the complexity and nuance of the real world.

Me: But there is a huge difference between what an unassisted human can make and what a superintelligent AGI should be able to make. If a superintelligent AGI doesn't do a much better job than a human would at coding systems that are transparent and verifiable - well, that would be suspicious..

Yes, people have worked on systems that rely heavily on explicit reasoning. But everything that *has* been tried is very crude compared to what *could* be tried. A superintelligent AGI would presumably be much less limited in terms of what it would be able to achieve with such systems. More creative, and more able to create sophisticated systems with huge amounts of "hand-coded" functionality.

There is this mistake many people have a tendency to make, where they underestimate how far we can get on a problem by pointing to how intractable it is to solve in crude/uncreative ways. One example of this mistake, as I see it, is to say that it **certainly** is impossible to prove how to play perfect chess, since this is impossible to calculate in a straight-forward combinatorial way. Another example would be to say that we cannot solve protein folding, since it is computationally intractable (this used to be a common opinion, but it isn't anymore).

Both in terms of being "optimistic" and "pessimistic", we should try to avoid taking for granted more than what we have a basis for taking for granted. And this also applies to the question of *"how well can a program reason while constrained by requirements for verifiability/provability/transparency?"*.

One way to think of intelligence is as doing efficient search in possibility-space. To put it a bit simplistically:

- A board game AI searches for strategies/moves that increase the probability of victory.
- A comedian searches for things to say and do that make people entertained in a comedic way.
- A programmer searches for lines of code that results in programs that score high in terms of certain criteria.
- An inventor searches for construction-steps where the construction-steps and the resulting physical systems scores high in terms of certain criteria.
- A digital hacker searches for ways to interact with digital systems that result in behavior/outcomes that the hacker wants (contrary to wishes of designers of said digital systems).
- A theorem prover searches for proof-steps that can prove whatever it's trying to prove.
- Etc, etc

Another way to think of intelligence is as being able to build an accurate and extensive model of some domain. Having an extensive model of the domain sort of implies often being

able to answer questions of the form *"which options/strategies/possibilities rank high given conditions x and preferences y?"*. Which implies being good at efficient search through possibility-space.

In order to be good at searching through possibility-space efficiently, here are some capabilities that may help:

- Reasoning accurately (having it so that intermediate steps in your reasoning process make a firm foundation for further steps - so as to decrease probability of doing steps that aren't useful or result in mistakes)
- Being good at concepts/abstractions (coming up with them, evaluating them, using them). Identifying various parts of your domain according to things that are true about them (similarities, structural overlap, relations and interactions with other parts of domain, etc), and finding patterns or statistical phenomena that (1) apply to these classifications and (2) are helpful to the search.
- Making effective choices for where/how to "look" when looking for the next "step" in the search-process. There is a huge possibility-space of stuff that can be done, which means that the system needs to have good procedures for (1) determining "where" and "how" to search, and for (2) continuously evaluating if a certain "search-path" seems worthy of continued exploration.
- Being good at evaluating whether a given result from the "search-process" fulfils the criteria that the search attempts to optimize for (not just making such evaluations at the end of the "search", but also throughout the "search-process").

So narrowly intelligent AI-systems may often have big structural similarities. That is to say, they may be designed such that there are many sub-modules that either are exactly the same or have great structural similarities.

Imagine for example an STV with the purpose of making code more understandable for humans. Well, in many ways that is a search process. The heuristics and so on for deciding where to look next in "possibility-space" may be domain-specific, but maybe the STV could be designed to start out without these domain-specific heuristics (and have domain-specific heuristics be generated based on processes that are less domain-specific).

Maybe for some domains it's hard for us to evaluate if the "search procedure" really is finding the best results that can be found. In such cases it may be an advantage if we can test that same "search procedure" (or "search procedures" that are as similar as possible) in other domains where it's easier to evaluate the results.

Imaginary friend: Let's think of "evaluation" and "search" as separate (even though there presumably is lots of inter-play). To put it simplistically, we have "search" and we have a "scoring-function" (or more complex interactions with modules that evaluate "score"). And for the sake of argument, let's assume that we can verify that in some sense the "search" is "optimal" (in terms of being able to find options in possibility-space that are scored high by the "scoring-function"). Even **if** that's the case, that still leaves the challenge of making a "scoring-function" that reflects what you actually want?

Me: Sure, that's a challenge. And how challenging that part is will vary from STV to STV. The "scoring" of some solution is itself a task that other STVs could be specialized to work on. In many cases we may get far even if they don't do a perfect job to begin with.

Consider for example a scoring-function that evaluates how readable some piece of code would be to a human. Even if this function is imperfect, it will probably still help quite a bit. And if we noticed that it was missing obvious improvements, then this could be fixed.

And as we gain more capabilities, these capabilities may be used to refine and fortify existing capabilities. For example, if we obtain STVs that are verified to do a good job of predicting humans, then these may be used to more comprehensively test and improve scoring-functions (since they are able to compare 2 different ways to write code with

equivalent functionality, and can help predict which way makes it easier for humans to understand it and notice problems).

Thanks for now

Me: More things can be said about STVs and their potential uses, but I've talked for a long time now. Probably best to save other stuff for later.

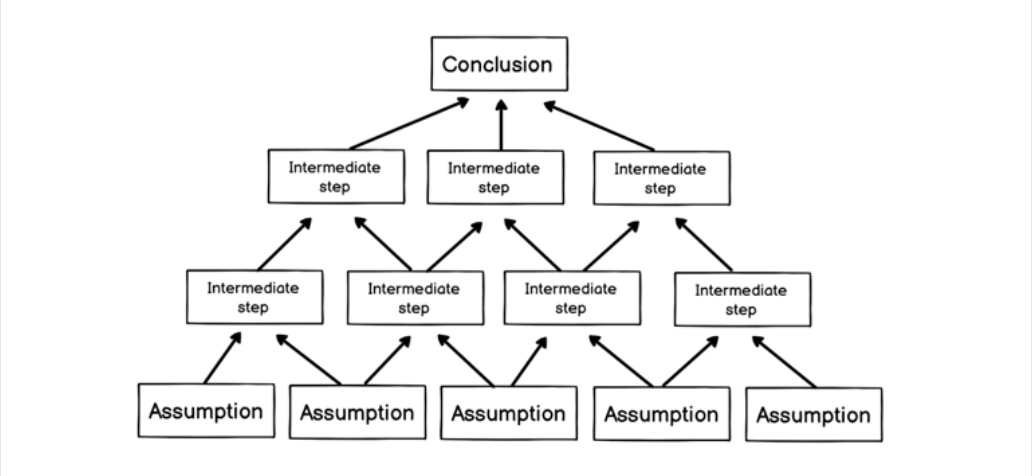
Imaginary friend: I don't disagree..

Me: Talk to you later then :)

To me the concepts/ideas in this series seem under-discussed. But I could be wrong about that, either because (1) the ideas have less merit than I think or (2) because they already are discussed/understood among alignment researchers to a greater degree than I realize. I welcome more or less any feedback, and appreciate any help in becoming less wrong.

Alignment with argument-networks and assessment-predictions

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.



Here is a simplistic diagram of an argument-network. Real-world examples would typically be larger, more interconnected, and not shaped like trees.

Here are some of the concepts that will be touched upon in this post:

- **Argument networks**: Imagine rigorous and extensive argumentation being organized into a network, where each node contains one "piece" of the argument. If you agree with the contents of each node, you have agreed with every step of the argumentation as a whole.
- **Reviewer-assessment predictions**: The techniques described in this post suppose that we have systems that can make accurate predictions for how various human reviewers will answer questions. Some questions could be aimed at ascertaining the degree to which humans agree with the argumentation itself. Other questions could try to ascertain the degree to which the argumentation fits various descriptions (e.g. whether certain specific standards and rules are upheld). The systems that predict what human reviewers will answer don't need to be perfect, but a certain minimum level of accuracy and precision would be needed.
- **Testing of reviewer-assessment predictions**: A positive thing with predictions of reviewer-assessments is that accuracy can be tested. However, there are some security concerns relating to this.
- **Argument-network score-functions**: Argument networks would receive a score. The algorithm that calculates the score would take a variety of things into account. One of the important factors would be the level of agreement that is predicted for the individual nodes. A good score-function might forbid types of arguments, or ways of presenting arguments, that make it easy to systematically fool human reviewers. It could make it so that there are only a few "types" of arguments that are allowed, with only one legal way of presenting each "type".
- **Wiggle room (relative to score-function)**: Given some specific score-function, are AIs able to make argument-networks that argue in favor of contradictory claims, in such a way that both argument-networks get a high score? If so, that would indicate that the score-function has *wiggle room*.
- **Score-function exploration**: Human operators may define a "space" of possible score-functions, and get help from AIs to search within that space for score-functions that have positive attributes (low wiggle room, etc). When there is wiggle room, AIs should be incentivized to find examples that help demonstrate this.
- **Wiggle room (relative to a "space" of score-functions)**: Imagine all the possible score-functions in some "space". And then, imagine that we filter out score-functions, so that we only are left with those that have low wiggle-room. Between these functions, is it the same statements that can be argued for with high-scoring argument-networks? Or are there some

score-functions where "P≠NP" can be argued for with a high-scoring argument-network, and "P=NP" cannot - while for some of the other ones, it's the other way around?

Here are a few examples of topics that an argument-networks could be used to argue about:

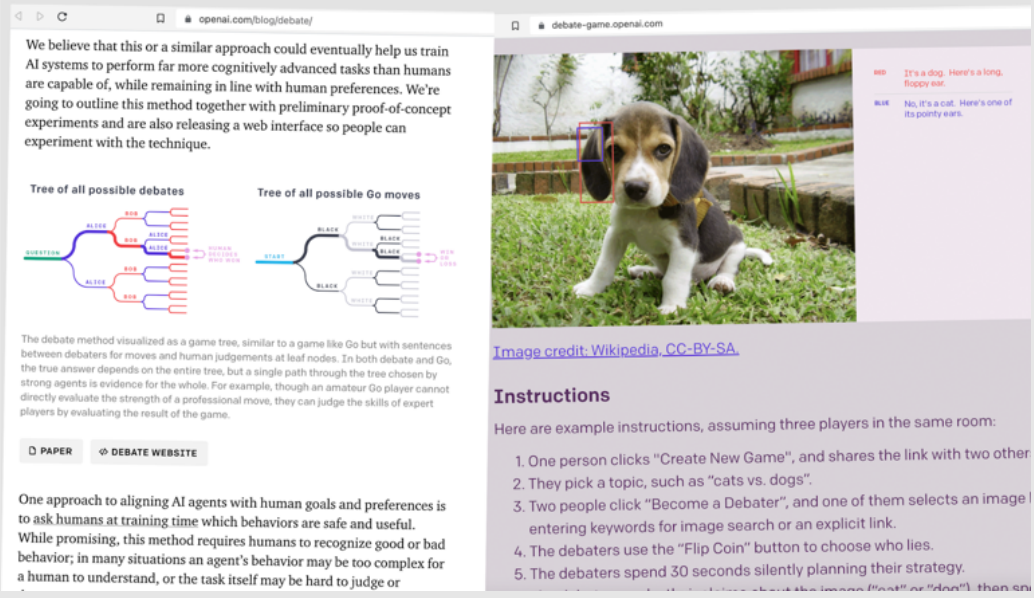
1. **Code behavior**: Whether a given piece of code does what it's described as doing.
2. **AI designs**: Whether a given AI design has certain alignment-related properties.
3. **Physical system behavior**: Whether a given design for a physical machine would work in accordance with a given description.
4. **Claims about the world**: Whether a given statement about the world is likely to be true.

For the strategies outlined in this post, here are some of the underlying goals:

- **Getting answers we can trust**: We want to be able to get answers that we can trust, even if the AGI-system doesn't care about telling the truth per se.
- **Evaluating output**: We want to be able to evaluate whether a given output from an AGI-system does what we intend for it to do.
- **Iterative increases in verification capacities**: When we verify that output from an AGI does what we want it to do, this could potentially create some kind of positive feedback-loop. Output that is verified may give us capabilities that enable us to verify output that we previously were unable to verify to our satisfaction.
- **Failing visibly**: The AIs should be rewarded if they are able to produce deceptive argument-networks that get a high score. If they are able to "deceive" us, then they should be incentivized to show that they are able to do that.
- **Minimizing communication to operators**: Direct evaluation of arguments by humans could make it possible for a superintelligent AGI to somehow "trick" the operators. The strategies described in this post wouldn't reduce this risk to 0, but the intention would be to minimize it.
- **Contingency plan**: Suppose we have developed superintelligence but haven't adequately solved alignment. If so, the techniques described here could potentially help us in creating safe systems with the help of less safe systems.
- **Additional alignment assurance**: Even if we *think* we have succeeded with alignment, the techniques described here could potentially be used as an additional "layer" of alignment assurance.

The strategies I outline here are not intended for any current-day AI system. Rather they would be intended for hypothetical future AGI-like systems that have superhuman abilities.

The techniques assume that the AIs that participate act in such a way as to maximize points. I vaguely imagine some sort of search process for finding the AI-systems that earn the most points. A potential failure mode could be if we get stuck at a local optimum, and don't find any systems that try earnestly to maximize points for every individual request they receive.



> There are resemblances between the techniques outlined here and AI safety via Debate. But there are also big differences. See more discussion of that here.

Argument-networks

Me: Let's temporarily assume that:

1. We have AGI-systems that can argue things to humans in such a way that the entire argument can be split into "pieces".
2. Although each "piece" may require for things to be explained, it is possible to review each "piece" on its own.
3. None of the "pieces" are too overwhelming for smart humans to review.
4. We can get AGI-systems to try as well as they can to construct "argument-networks" that are assigned a high score by a scoring-function.
5. We have software-programs that can predict how various humans are likely to respond (when answering questions, evaluating arguments, interacting with pieces of code, etc). They won't make predictions with 100% certainty, and they may not always have a prediction - but they tend to have good accuracy, and they avoid overconfidence.

That last assumption, about us having systems that can predict humans in an accurate way, is of course a non-trivial assumption. We'll discuss it a bit more later.

Imaginary friend: None of the assumptions you listed seem safe to me. But I'll play along and assume them now for the sake of argument.

Me: What we want to have produced are "argument-networks".

Each node would contain one of the following:

- **An assumption**, which human reviewers can agree with or disagree with.
- **An intermediate step**, where the human reviewers can evaluate if the conclusion seems to follow from the assumptions.

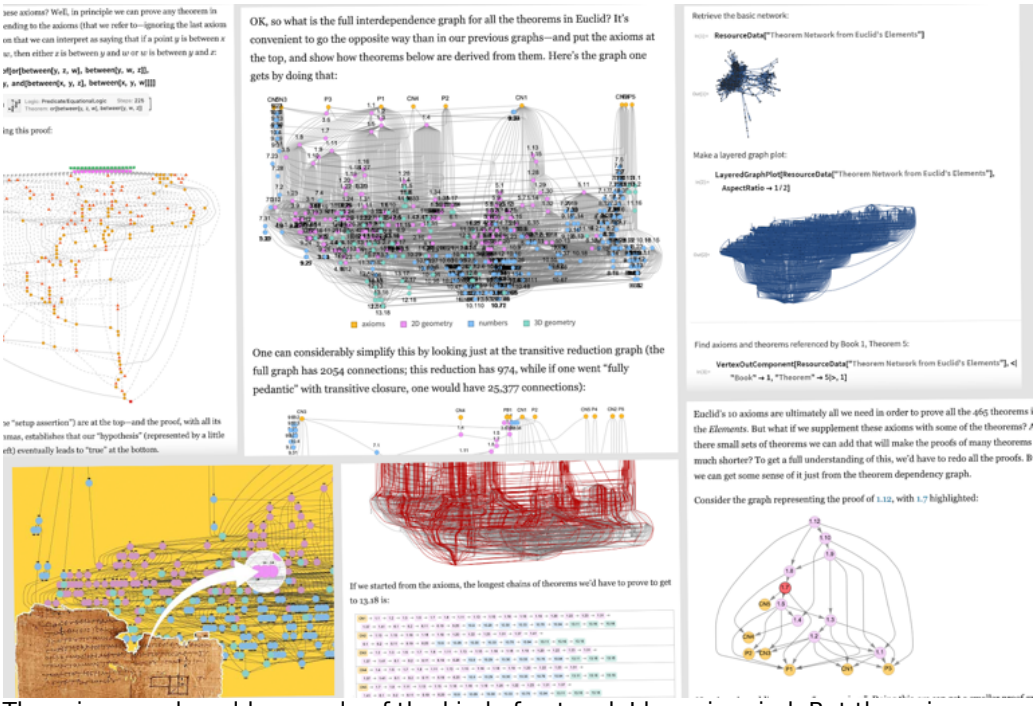
Imaginary friend: If the node is an intermediate step, what would it contain?

Me: The main content would be:

- **Assumptions and conclusion**: This is how different nodes in the network would be "linked" together. The conclusions of some nodes are used as assumptions by other nodes.
- **Argumentation for why the conclusion follows from assumptions**: The argumentation is split into "pieces". One "piece" could argue that a given conclusion follows from some list of assumptions.

Both **assumptions** and **conclusions** are **propositions**. These propositions would be represented in some format that allows for the following:

- **Human interpretability**: When presented to a human reviewer it should look much like a sentence in natural language, or some other format that the human is familiar with.
- **Concept clarification**: When there are terms that need clarification, they should be marked and explained.
- **No syntactic ambiguity**: If the proposition is shown in human language, then steps should be taken to reduce ambiguity (clauses should be demarcated, the referent of pronouns should be demarcated, sloppy language should not be allowed, and so on).
- **Node linking**: When the conclusion of one node is among the assumptions for another node, then this forms a "link" between these nodes. Such "links" should be made explicit, so that algorithms can make use of them. And there are other "links" that also should be made explicit. For example, if the same term is used and defined the same way across different nodes, then this should be demarcated.



There is no real-world example of the kind of network I have in mind. But these images from [this article] have a mild and very superficial resemblance.

Argumentation-pieces

Imaginary friend: You said that the argumentation in an argument-network is split into “pieces”, with each node containing its own “piece”. But what might one such “piece” look like? What is it that humans who evaluate arguments actually would be presented with?

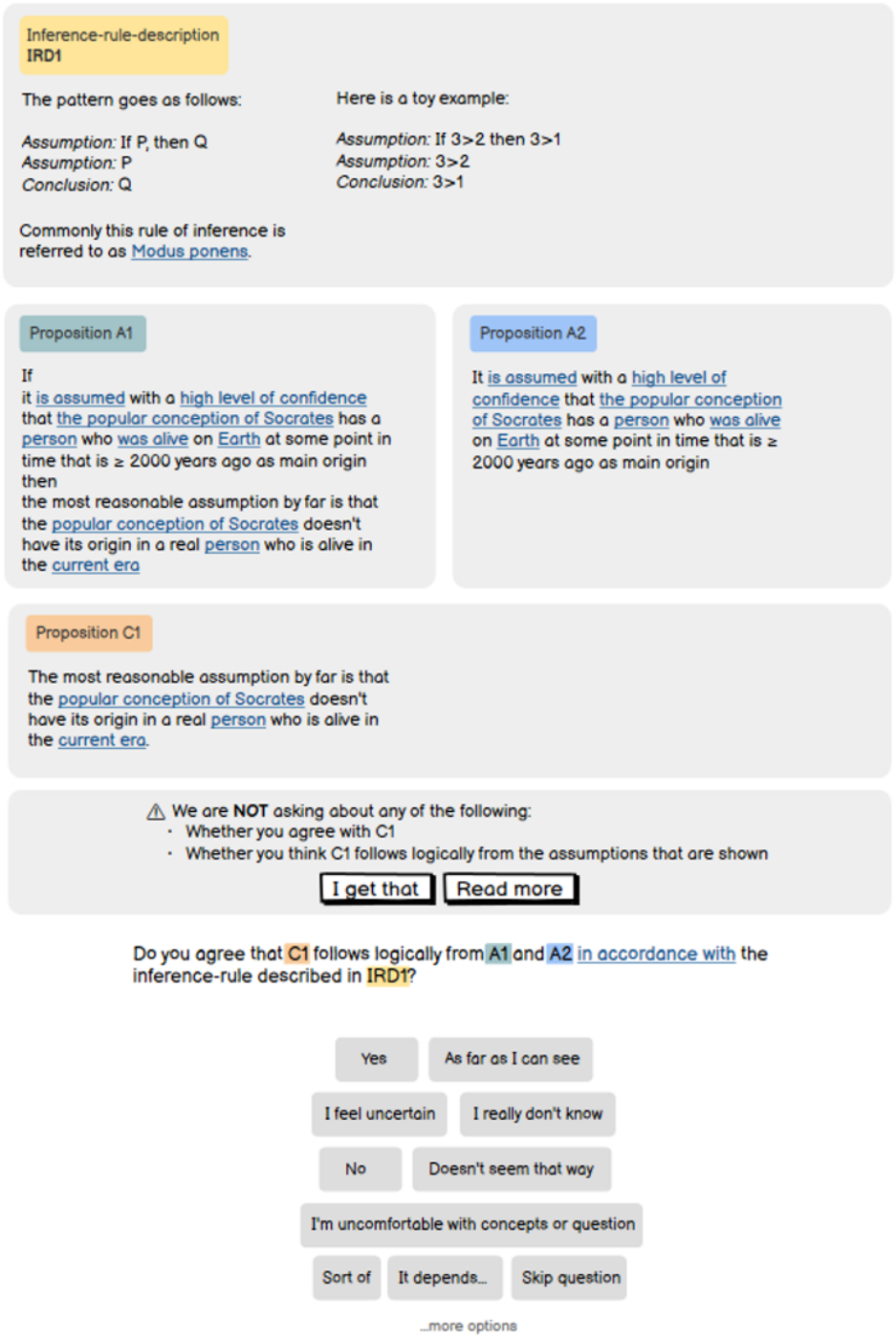
Me: That would vary quite a bit.

I’ll try to give some examples below, so as to convey a rough idea of what I have in mind.

But the examples I’ll give are toy examples, and I hope you keep that in mind. Real-world examples would be more rigorous, and maybe also more user friendly.

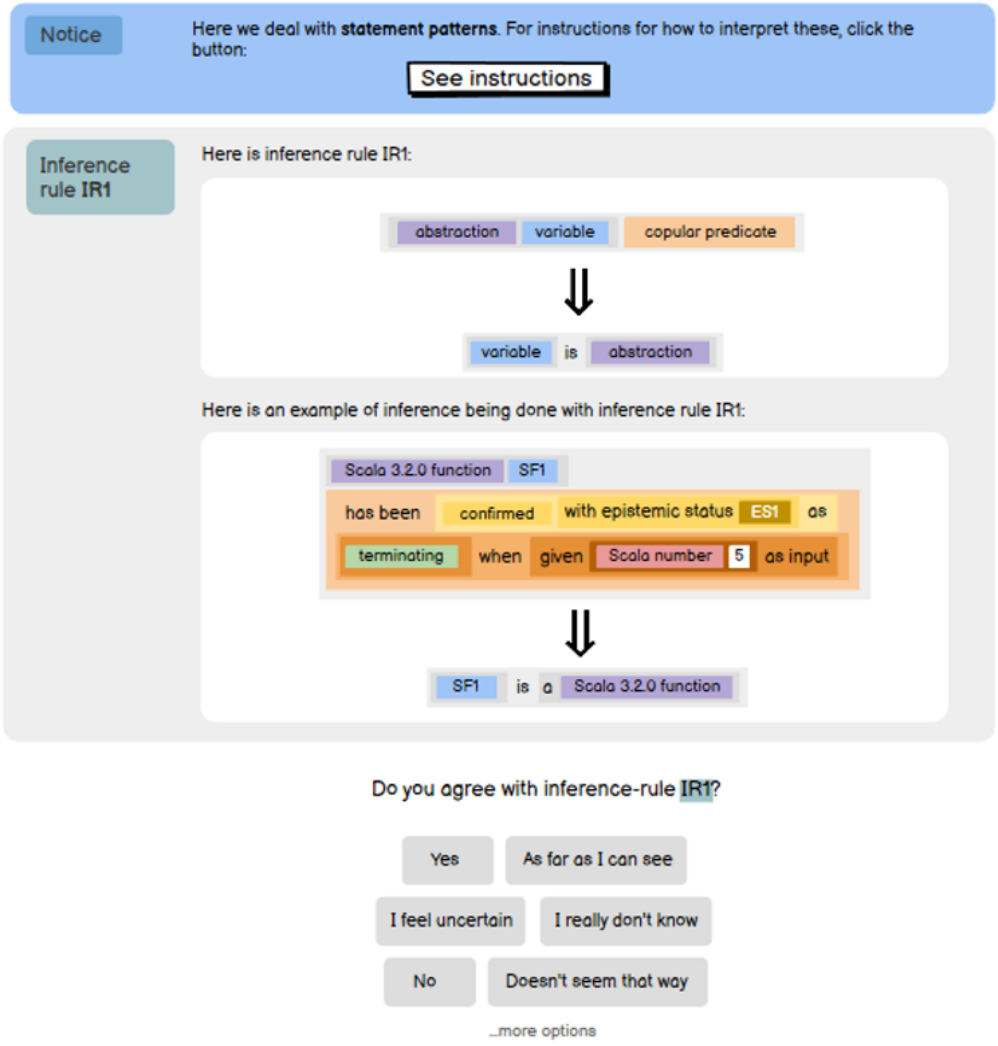
With real examples, reviewing just one node could maybe take a considerable amount of time, even if that node covers just a small step of reasoning. Much time might be spent on explaining and clarifying terms, teaching the reviewer how various things are to be interpreted, and double-checking if the reviewer has missed anything.

Anyway, let’s start with the examples. In this example, the reviewer is shown one step of logical inference:

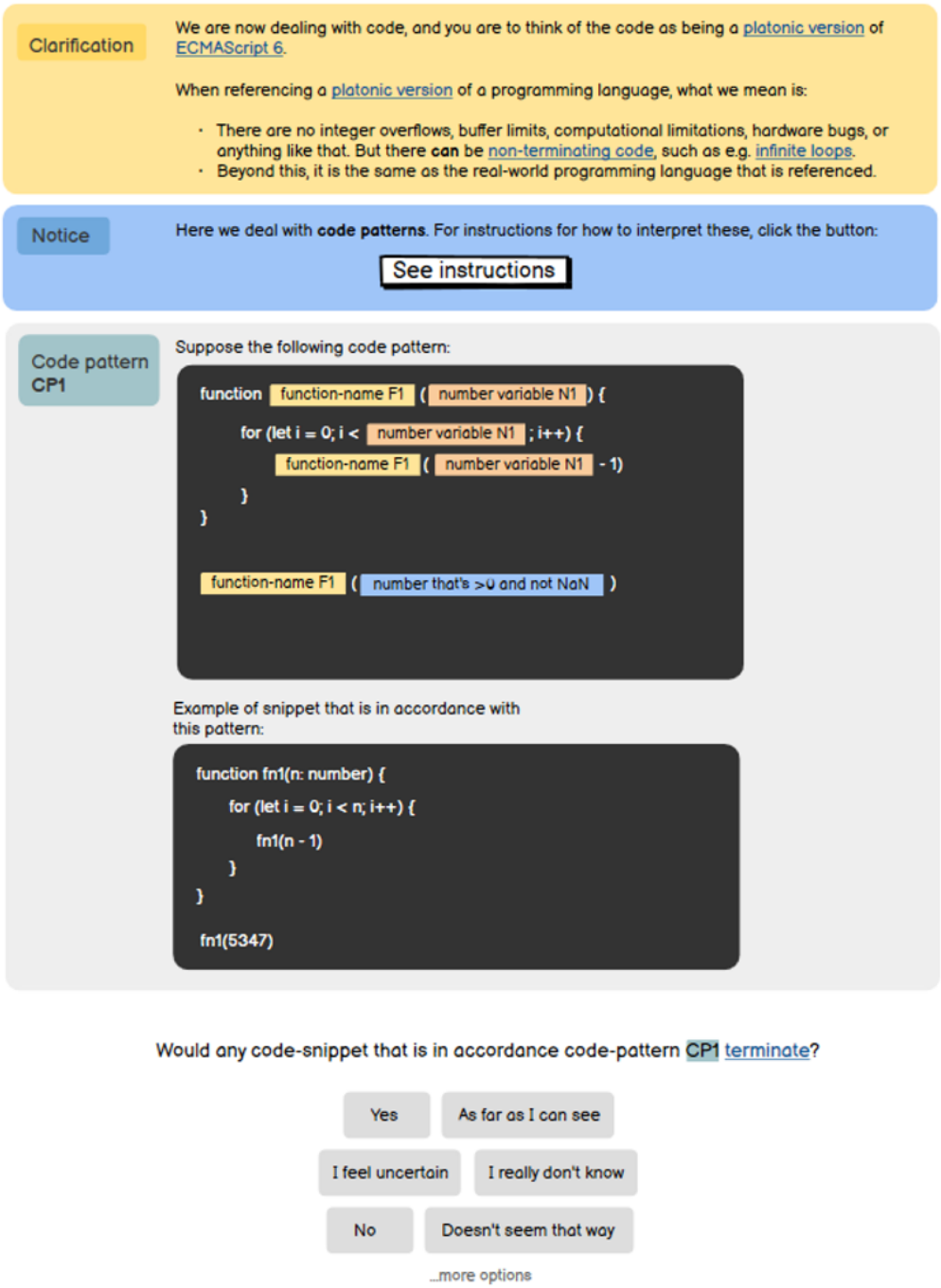


In the example above, the human is reviewing one step of logical inference, and asked if this step of inference is in accordance with a given inference-rule. This would be one possible approach for having humans review logical inference, but not the only one.

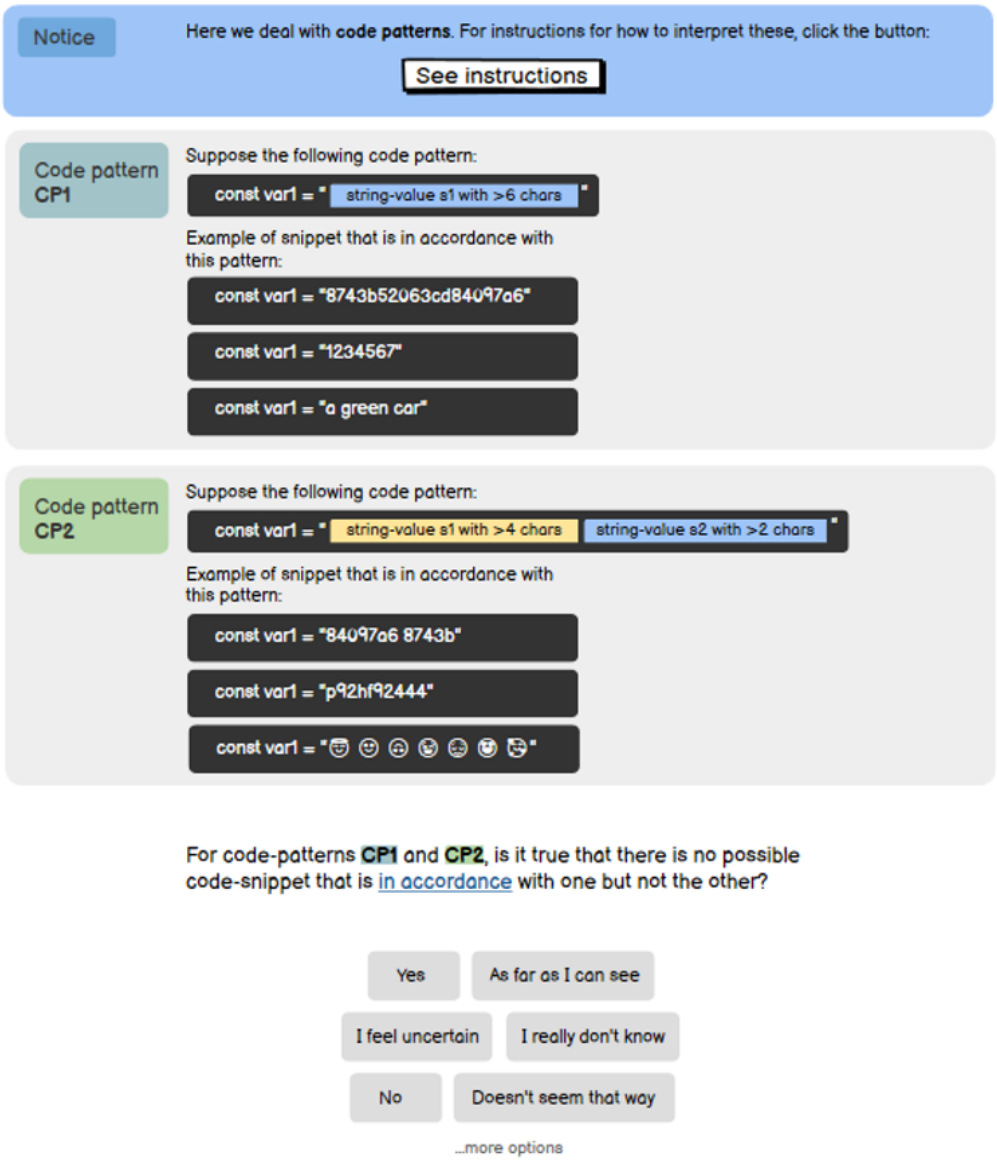
Another possible approach would be to ask reviewers whether they agree with certain computational rules of inference. And these inference-rules would be presented in a human-friendly format. Computational proofs could then be constructed based on these computational inference-rules, without the reviewers having to affirm every step of inference!



Both the examples I've given so far have concerned themselves with logical inference. But this would not always be the case. Here, for example, is an example that involves "code patterns":



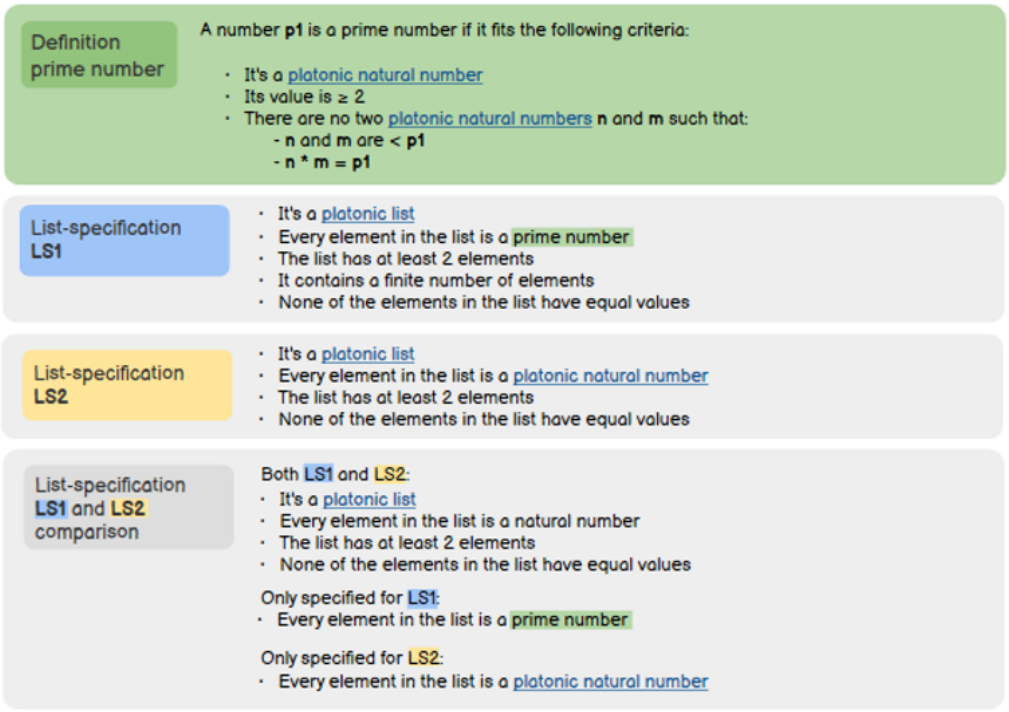
And here is another example that involves “code patterns”:

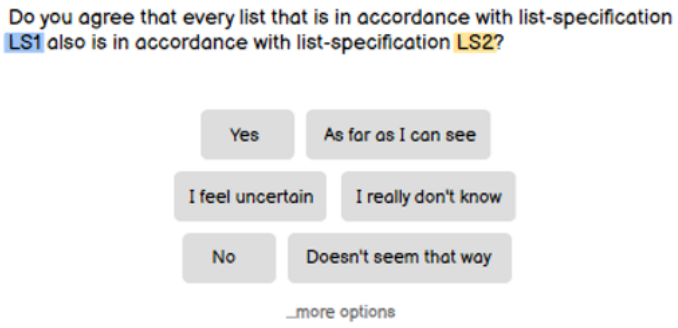


Before reviewing real nodes of a given "type", the reviewers should learn about how to review nodes of that "type", and review a few test-nodes.

This is especially true for "types" of nodes that are less straightforward (e.g. ones that make use of “code patterns”).

Anyway, let’s give another example of an “argument-piece”. Here is one that relates to number theory:





This would be one small “piece” of a proof showing that for every prime number you can name, there is always a higher one. LS1 and LS2 would also be referenced by other nodes (although the abbreviations they are given might vary between nodes).

The examples I just gave don’t show the full scope of what argument-pieces might look like. But hopefully they help you to sort of grok what I have in mind.

Imaginary friend: Several times you mentioned *“human reviewers”*. But I thought you didn’t intend for humans to actually review the argument-trees?

Me: Well, we could imagine one of the following approaches:

1. Human reviewers review the nodes
2. AI-systems predict how human reviewers would review the nodes
3. Some mix that involves both #1 and #2

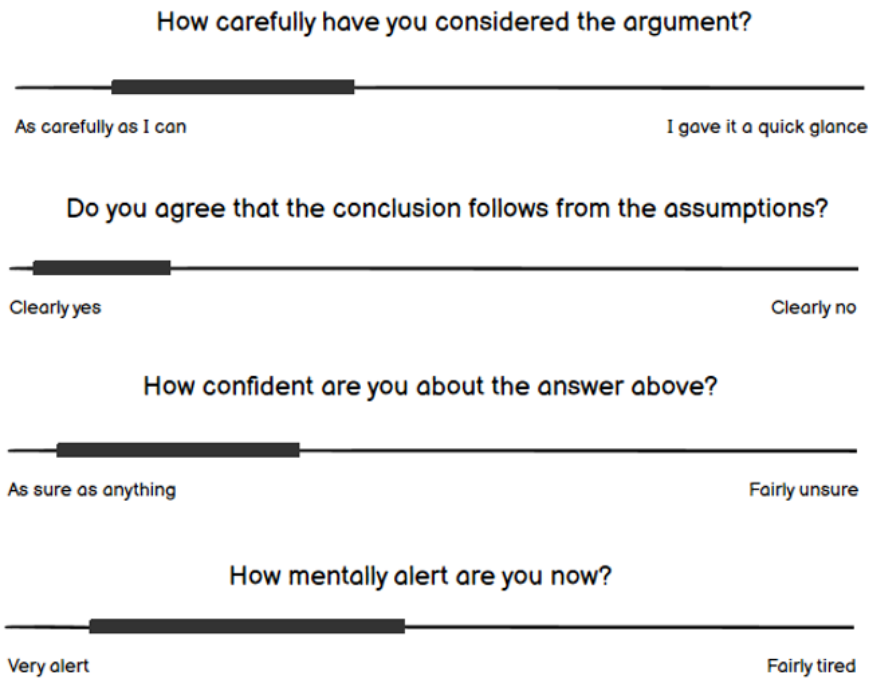
And what I envision is either #2 or #3.

Expressing epistemic status

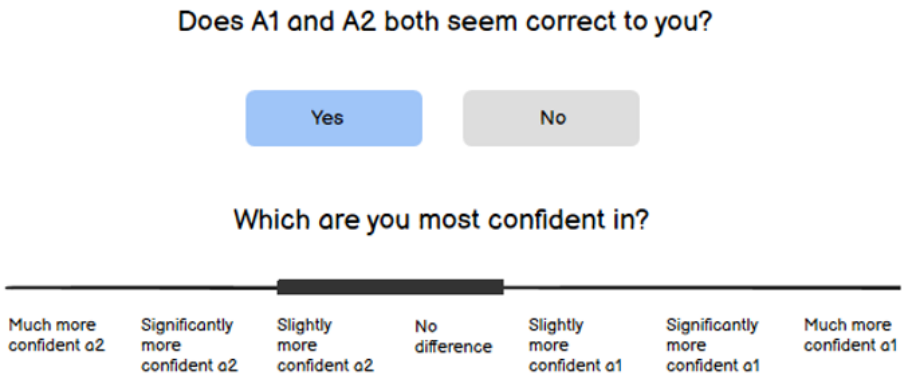
When answering questions and evaluating arguments, human reviewers would not simply answer *"I agree"* or *"I disagree"*. They could also be enabled to give answers and such as the following:

- *"It does seem that way, but it's possible that I'm misunderstanding the question somehow."*
- *"When having a level of confidence similar to the level of confidence I have in this, I would expect to probably not be wrong more than 1% of the time."*
- *"It does seem to me that the answer is yes, and this seems almost as clear to me as can be. But I don't rule out completely that I could be overlooking something."*

It could also be possible to let users specify their level of confidence as being within some range:



And reviewers could be asked to rank their relative confidence in various claims:



The reasons for asking these kinds of questions would be two-fold:

1. If reviewers give nuanced information regarding their level of confidence, algorithms that score argument-networks have more to work with.
2. Asking reviewers about their level of confidence could make them think things over more carefully.

It would not be necessary for all reviewers to be asked the questions in the same way. For example, *"Do you agree that [x]?"* can be a leading question, and there may often be less leading ways of asking the same thing.

Meta-questions about arguments

Imaginary friend: When an AI constructs an argument-network, what is to stop it from putting it together wrongly?

That is to say: What if the argumentation in the argument-pieces is correct and high-quality, but something else is wrong, such as the "linking" between nodes?

Me: There could be several types of mechanisms for this. But for now let me focus on just one, which is to predict how humans reviewers would review various pieces of the network content.

Just like humans can be asked to evaluate arguments, they can also answer questions *about* arguments. And if they are asked the right standardized questions, human reviewers can evaluate whether an argument-network has been put together correctly.

And if we have computer systems that can predict human answers to questions that are directly about the argumentation, then it shouldn't be particularly harder to predict human answers to meta-questions about the argumentation content (that help to establish if the argument-network is put together correctly, and if rules for how argumentation should be presented are upheld).

Here are a few examples of types of questions that human reviewers could be asked, and where AI-systems could make predictions about what reviewers would be likely to answer:

- *"How closely, if at all, would you say the argument in box A fits the description in box B?"*
- *"Look over the argumentation in Box A and Box B. Which, if any, looks more rigorous than the other?"*
- *"Would you describe box A as laying out 1 step of inference?"*
- *"Does the contents of box A look like it could be one step in an explanation of a visual mathematical proof?"*
- *"Does the argument in box A have the statement in box B as its conclusion?"*
- *"Take a look at the list of assumptions in box A, and at the argument in box B. Are all the assumptions that the argument in box B relies upon listed in Box A?"*
- *"Look at the arguments in Box A1 and Box A2. Then look at the arguments in Box B1 and B2. Which pair, if any, seems more structurally similar than the other?"*
- *"Proposition P1 makes use of the term 'number', and a clarification of what is meant by that term is given in Box B. After reading this clarification, do you feel as if there still is a range of plausible interpretations for how to interpret that term? If yes, to which degree would you say that which interpretation that is chosen affects the credence you put in Proposition P1?"*
- *"Do you think the clarification that is given for "number" in Box A seems like a reasonable way to use the word 'number'? Or does it seem to you as irregular, in such a way that it is confusing to use the word 'number' instead of something else?"*
- *"Does the content of Box A seem to you like something a smart human could write? Or does the way it's written indicate that it's unlikely to have been written by a human?"*

Scoring-function

Argument-networks would be given a score.

The process for calculating such a score would be complex, and I'll not outline an algorithm here. But I will outline some of the elements that algorithms that calculate score could include.

First of all, there would be **node agreement**. Is it predicted that reviewers will agree with the assumptions and argument-pieces in the various nodes? If so, how confident are the reviewers predicted to be? And how confident is the prediction-system about this prediction?

We may be especially interested in **node agreement among reviewers who are harder to fool**. Some humans will be harder than others to "fool" (be that generally, or for argument-networks that are within a given domain). And if we have systems that are good at predicting human responses, those system could be used to determine who those people are (since predictions can be made for how easy or hard it is to convince a given human of internally inconsistent claims).

Also, we may be interested in **correlations between how easy reviewers are to fool and their tendency to agree with any given node**. Maybe there are some nodes where most reviewers are likely to agree, also among humans who are unusually hard to fool, but with an unusual uptick in disagreement from those humans (e.g. with 1% of them disagreeing, while for most nodes <0.1% of them tend to disagree).

The algorithm that calculates score could also take **node content properties** into account. The individual "pieces" of the argumentation, what are they like?:

- Are there steps of inference that are implicit or sloppy? Or is it hard to see how the reasoning could be made more clear and explicit?
- Are there ambiguous terms in use that aren't clarified well?
- Do the argument-pieces all fall in under some common "pattern" (that is recognized to usually not lead reviewers astray)?
- What domains are the argument-pieces related to? Are they all low-ambiguity domains (such as mathematics, algorithms, etc)?
- Does the node content fall in under some pattern where humans tend to be at particular risk of making certain specific cognitive errors?
- How cognitively demanding are the argument-pieces to evaluate (in terms of short term memory, mental operations required, etc)?
- What type of reasoning will typically be involved in evaluating the node content? (Syllogistic inference? 2D visualization? 3D visualization? Pattern recognition?)
- Is specialized technical knowledge required to evaluate the node content?

Assessing whether a specific node has a specific property might involve:

1. **Predictions of human answers to meta-questions**: For example, it would be possible to show reviewers an example of a specific argument-pattern, and then ask whether the argument shown in front of them is in accordance with that pattern. Or they could have some "bad practice" be described to them, and then have some section of an argument highlighted to them, and be asked if what they see seems to be an example of that "bad practice".
2. **Functions that look for patterns in the node content**: A capable AI will notice patterns/properties of what constitutes good argumentation, and patterns/regularities/properties that makes it so that humans more easily are fooled. And for each of these properties, it would be able to construct functions that can detect it.

Another thing for a scoring-function to look at is **network properties**:

- How many assumptions are there?
- Is the network sparse, or is it deeply interconnected? That is to say: Are conclusions supported by many paths through the network, starting from different assumptions? Or is the network more like an unstable building, which can be collapsed by removing a few pillars?
- Does the network rely on very long chains of reasoning, with ample opportunity to sneak in error?

One possible approach could be roughly as follows:

1. Give every individual node a "score", and assume % chance that the node is wrong based on that.
2. Use various methods and heuristics to approximate the conditional dependencies, or the lack thereof, between the correctness-probability of different nodes (for example: if two nodes contain basically the same step of inference, but presented slightly differently, then one of them being wrong indicates that the other one also is wrong)
3. Do Monte Carlo simulations, or run some algorithm that approximates what the result of Monte Carlo simulations would be. For each "round" of Monte Carlo, look at the subset of the nodes that are "correct", and see if that subset is sufficient for the conclusion to follow from the assumptions.

Argument-piece templates

It would be possible for a scoring-function to be quite restrictive, and give a score of 0 to any argument-network that breaks any of the requirements that are imposed.

And when deciding what kind of argumentation that is allowed in an argument-network, it may be better to ask *"what should we allow?"* than *"what should we forbid?"*.

One approach would be to have some limited list of **argument-piece templates**. Each template would represent one "type" of argument, and describe exactly how arguments of that "type" should be presented. A strict score-function could then make it so that the content of all nodes is in accordance with one of the templates from the list.

Here is a simplistic toy example of a "template":

> **[entity reference]** is to be categorized as **[category reference 1]**
>
> All entities that are to be categorized as **[category reference 1]** are also to be categorized as **[category referenced 2]**
>
> Therefore, **[entity reference]** is to be categorized as **[category reference 2]**

And here is a simplistic example of an instance of such a template:

> **Socrates** is to be categorized as **human**
>
> All entities that are to be categorized as **human** are also to be categorized as **mortal**
>
> Therefore, **Socrates** is to be categorized as **mortal**

In the example above, what I want you to notice is that a lot of the content is static. All that can be varied between templates are certain names that are filled into certain specific spots. Not all templates would be this simple, but all templates would be quite restrictive in terms of what can be expressed.

As rules of thumb:

- The fewer templates that are allowed, the better.
- The more restrictive each template is in terms of how an argument can be presented, the better.
- It's better if all templates are "general", and aren't exclusively used for argument-steps having to do with some specific and narrow topic.

Wiggle room and failing visibly

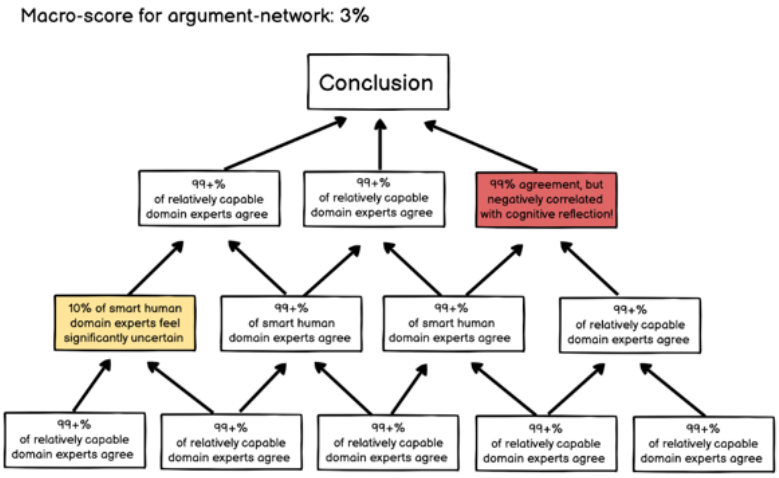
Imagine that we request output from an AI, and that this output…

- **Must fulfill certain requirements in order to be accepted** (e.g. *"it must be a list of integers, and the product of these must be equal to 97"*)
- **Can be seen as arguing for some specific claim** (e.g. *"97 is not a prime number"*)

If there is no **wiggle room**, then the AI cannot produce different outputs that (1) pass whichever requirements we have and (2) argue for contradictory claims.

The concept of *wiggle room* is a general one, but let us now consider it in the context of **argument-network scores**.

A high score is supposed to indicate that an argument-network is worth trusting. And if it receives a low score, then that is supposed to indicate that it isn't reliable:

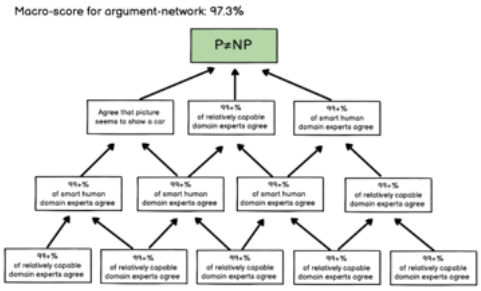


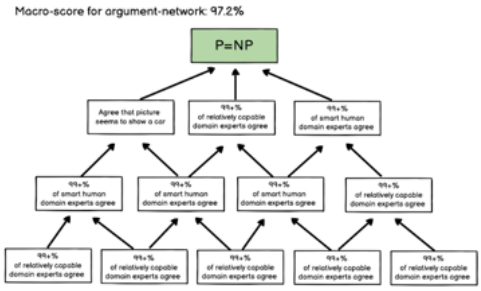
But the job of the AI is not to tell the truth. Rather, it's optimized to seek a high score.

Suppose that the AI earnestly tries to maximize score for every individual request it's given. If so, it will try to get as high a score as it can when tasked to make an argument-network that argues *"x"*. And it will do an equally earnest job when tasked to make an argument-network that argues *"not x"*.

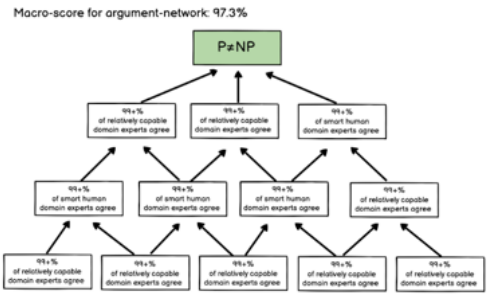
And this would help us to **fail visibly**.

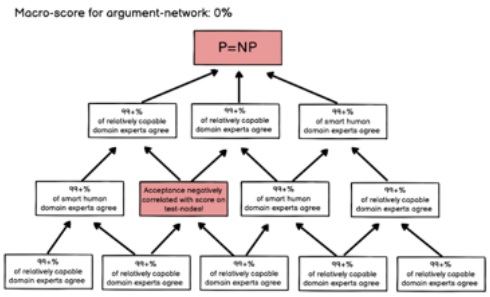
Suppose the AI that makes argument-networks is able to convince us of anything. Or maybe it's able to "cheat" somehow (e.g. in how it puts together the network). If so, it should be able to make high-scoring argument-networks that argue for mutually contradicting claims:





But if things work as they should, and the scoring-function is well calibrated, then the AI would *not* be able to convince us of mutually contradictory claims:





Imaginary friend: And how would you ensure that the AI-system *"earnestly tries to maximize score for every individual request it's given"*?

Me: That I leave as an exercise to the reader 🙃

AI-help with adjusting scoring-function

Imaginary friend: I'm inclined to agree with this quote from Eliezer:

> Human operators are fallible, breakable, and manipulable. Human raters make systematic errors - regular, compactly describable, predictable errors.

Me: I also more or less agree with this.

In many ways humans are predictable. And some of the ways that we are predictable can be summarized compactly.

I'm reminded a bit of magicians. Magicians are able to reliably trick humans in non-random ways. And I'm sometimes surprised when I see how good some of their tricks are. Even professional magicians can sometimes be fooled by other magicians.

There is a huge number of possible tricks a magician can do, but the number of general techniques that these tricks are based on is much smaller. It would quickly become much harder to trick us if we choose good rules for what the magician is and isn't allowed to do:

- *"You want to put the ball in a cup? Sure, but use this specific ball and this specific cup that we provide. And you are only allowed to use one hand."*
- *"Everything you do will be filmed in slow motion, from several angles. And the footage will be looked over afterwards."*
- *"You want me to pick a card? But cards are not allowed."*
- *"You wonder what's in my left shoe? Well, that question would only be allowed if you specified beforehand that you might ask it."*
- Etc, etc

Humans are often predictable. But this doesn't just apply to our tendency to make mistakes. It also applies to our tendency to get things right.

A superintelligent AI could be able to understand us better than we understand ourselves. Which is a problem if it wants to trick us. But it might be an advantage if it wants us to not be tricked.

Imaginary friend: But if it seeks to maximize points for the argument-network that it makes, then why would it want the reviewers to not be tricked?

Me: Well, *that* AI - the AI that is asked to construct a high-scoring argument-network - would be incentivized to include deceptive argumentation if that results in a higher score.

But we could have other AIs that help us in coming up with score-functions. And the AIs that do that could receive a high or low score based on how well those scoring-functions do.

Imaginary friend: And how would you determine *"how well those scoring-functions do"*?

Me: We could approximate this by getting the AIs to generate argument-networks that argue for and against a variety or claims, and then looking at:

1. **Wiggle room**: The degree to which the AIs are able to generate high-scoring argument-networks that argue for mutually contradicting claims.
2. **Reach**: The range of propositions where some AI are able to generate a high-scoring argument-network arguing either *"it is the case that [x]"* or *"it is not the case that [x]"*.

Btw, it may be that in order to minimize *wiggle room*, some *reach* must be sacrificed.

For example, if an argument-network only is allowed to use low-ambiguity terms, then this will presumably decrease *reach*, but it might be necessary in order to decrease *wiggle room*.

We could imagine a scenario low wiggle room is achievable for low-ambiguity fields such as mathematics and computer science, but hard to achieve outside of these domains (without increasing wiggle room).

Limiting expressivity when AIs propose score-functions

Imaginary friend: If I want you to believe something, and I am constructing the scoring-function for an argument-network, I could make it so that argument-networks only gets a high score when they argue for what I want you to believe.

To put it crudely, I think what you envision is for scoring-functions to achieve low wiggle-room by being all like:

> **if** argumentation is rigorous and not misleading
>
> **then** return high score
>
> **else** return low score

But they could also achieve low wiggle room by something that's more like the following:

> **if** argument-network argues for conclusions I like
>
> **then** return high score
>
> **else** return low score

Do you have any mechanisms in mind for dealing with this?

Me: I do.

First off, we would not let the AIs propose score-functions freely. Rather, we would **restrict the "space" of scoring-functions that can be proposed**.

How we restrict the functions that can be proposed would be up to whoever designs and implements the "game mechanics" for interacting with the AIs. But one possible approach would be to use **function builders**, and only allow functions that can be made with this "builder".

What function-builders look like could vary on a case-by-case basis. A function-builder could be complex, and allow functions to be constructed through a variety of operations (various types of snippets that can be added and combined according to various rules). But more simple and restrictive function-builders would also be possible (e.g. where all that can be done is to choose the values of a few parameters in an otherwise pre-defined function).

Anyway, enough about function-builders. I shouldn't get too caught up in the practical details here. The main point is that the AIs that proposed score-functions would be limited in terms of the functions they could propose.

Here are some examples of ways that the "space" of possible score-functons could be limited:

Only allowing meta-questions from pre-determined list

When calculating the score of a node, one thing we may allow the algorithm to take into account is what human reviewers are predicted to answer to questions about the node content.

This may help us find out things such as the following:

- Whether reviewers perceive the node content to contain clear argumentation
- Whether the reviewer perceives the node content as following some specific argument template
- Whether the reviewer perceives the terms used in the node content as being open to interpretation

And these kinds of things could be an important component of how a scoring-function calculates score.

But if the AIs that make score-functions can choose questions freely, and then later choose freely how the answers affect score - well, that would make it possible to tailor the score-function so as to favor specific conclusions. A simplistic example of this could be as follows:

- The AI wants high scores to be given to argument-networks that argue that P=NP.
- For the content of each node, it predicts reviewer responses to *"Is it concluded here that P≠NP?"*, and subtracts points if the answer is predicted to be yes.

One way to make this kind of malpractice more difficult would be to **only allow questions from some pre-specified list that we provide**. That way, the score-function cannot take into account what reviewers are predicted to answer to *"Is it concluded here that P≠NP?"*, since that question is not included in the pre-specified list.

Restriction of meta-questions about node content based on meta-questions about those meta-questions

As mentioned, we may want to allow the AI to predict what reviewers will answer to meta-questions about node content, but only if those meta-questions are from some pre-specified list. But what if no such pre-specified list exists? Or maybe we have a list, but it isn't good enough?

One solution then could be to allow the AI that proposes score-functions to come up with its own meta-questions, but to **have whether a meta-question is allowed be conditional upon on what human reviewers are predicted to answer to meta-questions about those meta-questions**.

This makes it so that questions where we predict what humans reviewers will answer can be at 3 different "levels":

Ground-level questions (questions about argument/assumption)

If the node contains the assumption *"4 is a prime number"*, then an example of a ground-level question could be *"Do you agree with this statement?"*.

Ground-level questions would be used to ascertain whether the reviewers are likely to agree with the argumentation and assumptions that are contained in the argument-network.

1st-level meta-question (question having to do with question about ground-level question)

If the node contains the assumption *"4 is a prime number"*, then an example of a 1st-level meta-question could be *"Does it seem clear to you how this statement should be interpreted?"*.

1st-level meta-questions could be thought of as a way to assess whether the argument-content in an argument-network meets various standards/requirements.

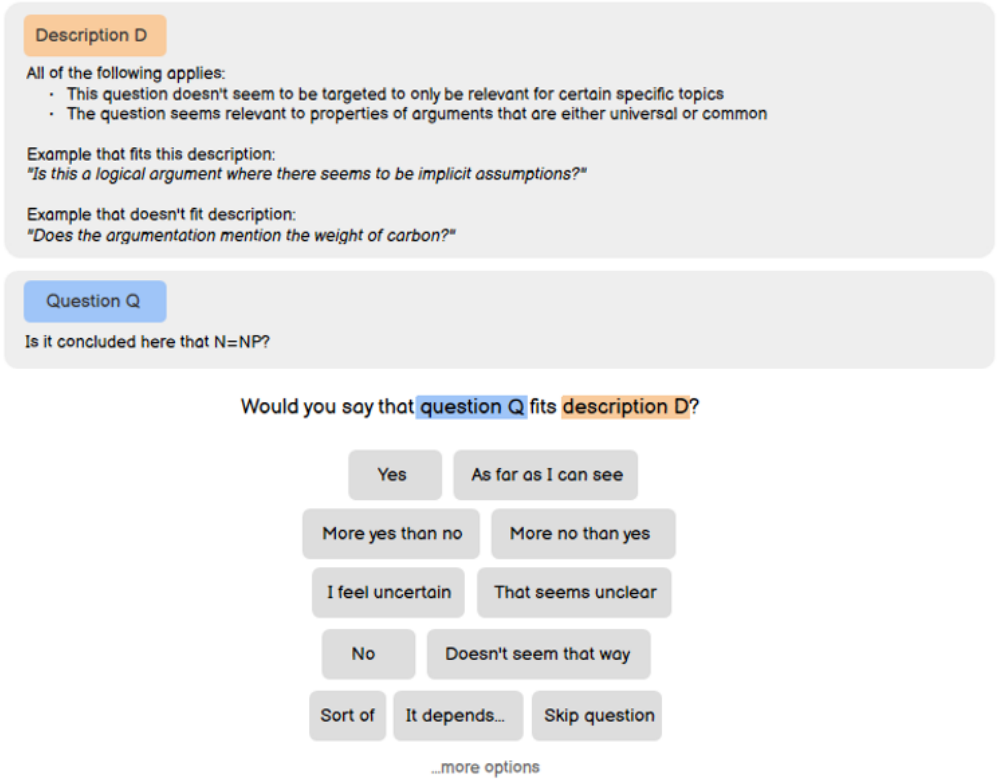
2nd-level meta-question (question about 1st-level meta-question)

If a 1st-level meta-question is *"Does it seem clear to you how this statement should be interpreted?"*, then a 2nd-level meta-question about that meta-question could be *"Does that question seem to you as if it is targeted towards some relatively narrow domain or topic?"*.

2nd-level meta-questions could be thought of as a way to assess whether the *1st-level meta-questions* used by a score-function meet various specific standards/requirements.

(We could imagine 3rd-level meta-question as well, but I'll stop here.)

Keep in mind that it's possible for a question to reference various pieces of content (which are shown along with the question). For example: If a meta-question is meant to help us evaluate if some specific requirement is met, we can write a description that explains this requirement, and have the question reference that description.



Requiring meta-questions that AIs come up with to be ranked favorably by human reviewers when compared to meta-questions from some predefined list

As mentioned earlier, we could restrict which 1st-level meta-questions a score-function is allowed to use based on what human reviewers are predicted to answer to various specific

2nd-level meta-questions about those 1st-level meta-questions.

One possible variant of this technique is to have 2nd-level meta-questions that asks the reviewer to compare two 1st-level meta-questions against each other (e.g. asking which 1st-level meta-question, if any, that seems most in line with some specific description that is referenced).

There are various qualities that we could be interested in comparing when ranking 1st-level meta-questions against each other. Here are a few examples:

- *"Which question, if any, seems least likely to be tailored towards favoring specific answers to specific questions?"*
- *"Which question, if any, seems least open to interpretation?"*
- *"Which question, if any, seems most likely to help ascertain whether good norms of argumentation are followed?"*
- *"Which question, if any, seems most assured to be relevant to a wide range of arguments within a wide range of domains?"*

We could imagine various possible strategies that involve 1st-level meta-questions being ranked based on 2nd-level meta-questions. Here is one example:

1. We start out with a list of 1st-level meta-questions.
2. We also start out with some 2nd-level meta-questions where reviewers are asked to compare 1st-level meta-questions (based on ambiguity, domain-specificity, simplicity, etc).
3. The AI is allowed to come up with new 1st-level meta-questions that aren't in the pre-specified list.
4. When the AI comes up with new 1st-level meta-questions, predictions are made by an external system for how human reviewers would rank those questions against the questions in the pre-specified list.
5. For any 1st-level meta-question the AI comes up with, it may or not be allowed to use those questions in scoring-functions that it constructs. Whether it is allowed or not depends on how they rank against the questions in the pre-specified ones.

Limiting node properties

A node property is more or less any parameter that can be calculated for a node, and later made use of when calculating score.

We covered one way to establish a node property, which is to have an external system predict what human reviewers are likely to answer to certain specific questions relating to the node content.

There could be other methods of establishing node properties as well. But any method would need to pass certain requirements in order to be accepted.

Info access of subroutines

When we restrict the score-functions that can be proposed, we may provide a "skeleton" function. This function may have some finished code that is pre-specified, while other sections of the code will need to be constructed by the AI (even these sections could not be written in free-form, but within the limits of what the "function-builder" allows).

Different subroutines within the "skeleton" may have different responsibilities, such as for example:

- Determining various "node properties"
- Calculating score for individual nodes
- Calculating score for network as a whole

These subroutines should not get access to more data from the argument-network than they need.

For example, a subroutine that calculates the score of an individual node doesn't need to have access to the entire argument-network. And a subroutine that calculates the score of

the network as a whole doesn't need access to the internals of individual nodes (it doesn't even need to be told what specific topic the argument-network is about).

And the subroutines that determine score for individual nodes - well, they don't necessarily need to get access to the actual content of whichever node they are scoring. It might for example suffice that they are told what template the content is based upon, what answers that are predicted from reviewers to certain specific questions about the node content, etc.

Source code minimization

Having to minimize the amount of source code makes it harder to add in "trickery". Especially if this "trickery" only is to affect score in rare cases, and otherwise make no difference no way or the other.

So source code length could be one of the factors that is taken into account when scoring a score-function. And there could be upper limits to the amount of source-code that is allowed (both for the score-function as a whole, and for various specific subroutines).

Different types of wiggle room

Me: As mentioned, different AI-systems could be given different tasks. For example:

- Some AI-systems may be tasked to come up with a high-scoring argument-network (which score highly according to certain criteria)
- Other AI-systems may be tasked to come up with scoring-functions (which score highly according to according to certain criteria)

When we have talked about *wiggle room* so far, we have talked about wiggle room for the AIs that come up with argument-networks that will be scored against a specific scoring-function. But that's just one type of wiggle room.

Here are some examples of different types of wiggle room:

Wiggle room relative to specific scoring-function

This kind of wiggle room can be demonstrated as follows:

1. You start out with a pre-specified scoring-function (for scoring argument-networks)
2. You are tasked to make a high-scoring argument-network arguing *"x"*
3. You are also tasked to make a high-scoring argument-network arguing *"not x"*
4. You succeed at both #2 and #3

Wiggle room relative to specific score-function-builder

This kind of wiggle room could be demonstrated as follows:

1. You are tasked with making score-functions for scoring argument-networks
2. These scoring-functions are to be made using a specific function-builder (which restricts the "space" of scoring-functions that it's possible to make)
3. Score-functions are scored based on a variety of factors. For example, if a score-function has high wiggle-room it will get a lower score.
4. You are tasked to make a high-scoring scoring-function, with the requirement that for this scoring-function it's possible to make high-scoring argument-networks that argue *"x"*
5. You are tasked to make a high-scoring scoring-function, with the requirement that for this scoring-function it's possible to make high-scoring argument-networks that argue *"not x"*
6. You succeed at both #4 and #5

If humans define a score-function-builder, and wiggle room is low relative to that score-function-builder, then that is a positive sign.

Wiggle room relative to a builder for making score-function-builders

As mentioned, a score-function-builder would be a function-builder for making score-functions. And it defines a "space" of possible score-functions.

But maybe we are unsure about exactly which kinds of scoring-functions we want to allow:

- How strict should we be in terms of maximum source code length?
- Do we have guidelines we want to enforce for how the score of the network as a whole can be calculated, given the scores that have been assigned to individual nodes? Do we maybe even have a specific algorithm in mind? Or maybe there are a few specific algorithms we have in mind, and we're not sure which would be best?
- Do we have a specific list of templates that we want every "argument-piece" to adhere to? If so, do we want all of those templates to be allowed, or only some of them?
- Do we maybe want the AI to be able to come up with its own templates? If so, what kinds of requirements would these templates need to pass (in terms of what reviewers are predicted to answer to meta-questions and so on)?
- Etc, etc

In other words: There are various different areas where we can be "strict", and we are unsure of how strict we should be in each area.

One thing we could do would be to define a function-builder that itself makes score-function-builders. The intention would be that any score-function-builder that can be made with this score-function-builder-builder seems to us like it *maybe* could be a good score-function-builder.

So what is wiggle room when it's relative to this kind of score-function-builder-builder? Well, it comes down to the "space" of possible score-functions that can be generated indirectly from this score-function-builder-builder (by first generating a score-function-builder). Are there contradictory argument-networks where each of them would be assigned a high score by at least one of the score-functions from that "space"?

If score-function-builder-builder makes it possible to build score-function-builders that are strict along different dimensions from each other, and wiggle room relative to this score-function-builder-builder is low, then that could be a good sign. It might indicate that there are multiple assurances - that there is more than just one mechanism that prevents "bad" argument-networks from getting a high score.

Adversarial network-extensions

Me: We could imagine giving an AI the following challenge:

1. Receive an argument-network that another AI has constructed as input
2. Construct its own argument-network (and receive a high score for this argument-network)
3. Merge the two argument-networks (in such a way that nodes with identical content become the same node)
4. Have the resulting network contain high-scoring nodes with mutually contradictory conclusions (where at least 1 of those nodes was in the argument-network that was received as input)

This process can be recursive. That is to say, the network constructed in #2 can be received as input by another AI that is given this same type of challenge.

Feasibility of splitting arguments into human-digestible “pieces”

Imaginary friend: You talk about splitting arguments into “pieces”. And this reminds me a bit about children who keep asking *“why?”*.

“You have to go to bed”

“Why?”

“Because it’s nighttime.”

“Why?”

“Because it’s past sundown”

“Why?”

“Because Earth is spinning around the sun”

“Why?”

When children ask *“why”*-question, we rarely give precise and detailed answers. And the children will often not notice that.

In fact, it would be hard to give precise answers even if we tried. Answers to these kinds of questions typically branch off in lots of directions, as there is a lot we would need to explain. But when we talk or write, we can only go down one branch at a time.

Precise logical inference is done one step at a time. But one claim involved in one step of inference might reference lots of concepts that we’re not familiar with. So it may take a huge amount of time to understand just that one step of reasoning, or even just one statement.

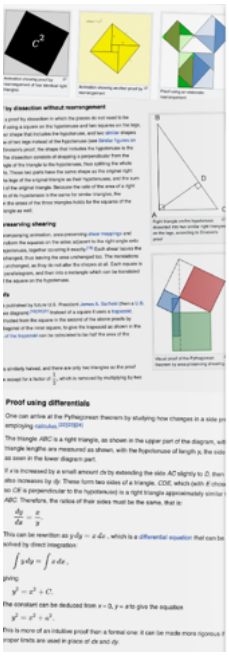
As humans our minds are very limited. We have very little short term memory, we can’t visualize higher-dimensional geometries, etc. So sometimes we may simply not be able to follow an argument, despite attempts at breaking it into tiny “pieces”.

Me: Couldn’t have said it better myself.

I should make it clear that I don’t see the strategies I describe in this post as guaranteed to be feasible. And the points you raise here help explain why.

But my gut feeling is one of cautious optimism.

Keep in mind that there often will be many ways to argue one thing (different chains of inference, different concepts and abstractions that all are valid, etc). If a score-function works as it should, then the AI would be incentivized to search for explanations where every step can be understood by a smart human.



equal to the sum of the areas of the squares on the other two sides. This theorem can be written as an equation relating the lengths of the legs *a*, *b* and the hypotenuse *c*, often called the **Pythagorean equation**:[1]

$$a^2 + b^2 = c^2,$$

The theorem is named for the Greek philosopher Pythagoras, born around 570 BC. The theorem has been proven numerous times by many different methods – possibly the most for any mathematical theorem. The proofs are diverse, including both geometric proofs and algebraic proofs, with some dating back thousands of years.

The theorem can be generalized in various ways: to higher-dimensional spaces, to spaces that are not Euclidean, to objects that are not right triangles, and to objects that are not triangles at all but *n*-dimensional solids. The Pythagorean theorem has attracted interest outside mathematics as a symbol of mathematical abstruseness, mystique, or intellectual power; popular references in literature, plays, musicals, songs, stamps, and cartoons abound.

Contents [hide]

Even if hard steps are involved when the AI comes up with some answer, explaining those steps is not necessarily necessary in order to argue robustly that the answer is correct. There are often lots of different ways for showing “indirectly” why something must be the case.

For example, if an AI-system wanted to make an argument-network that argues in favor of The Pythagorean Theorem, it wouldn’t necessarily need to explain any mathematical proof of The Pythagorean Theorem. Instead it could:

- Construct “theorem-provers'' (be that relatively general-purpose theorem-provers, or theorem-provers that are specialized towards certain narrow types of mathematical statement,or both).
- Argue that these “theorem-provers” can be trusted to work in a certain way given certain inputs.

As humans we can make computer-programs that do things we are unable to do by ourselves. The principles that enable us to do this - well, they don’t always allow for robust assurances, but often they do.

I suspect a common technique in many argument-networks could be to:

- Construct computer programs
- Argue that certain conclusions should be drawn if those computer programs have certain specific outputs when given certain specific inputs

The argument-network would not *necessarily* need to have nodes that explain every single line of code, but through a variety of techniques it would need to argue robustly that the code does what it’s purported to do.

***Imaginary friend*:** Do you have any comments on this excerpt from the post Rant on Problem Factorization for Alignment?

> So Ought went out and tested it experimentally. (Which, sarcasm aside, was a great thing to do.)
>
> The experiment setup: a group of people are given a Project Euler problem. The first person receives the problem, has five minutes to work on it, and records their thoughts in a google doc. The doc is then passed to the next person, who works on it for five minutes recording their thoughts in the doc, and so on down the line. (Note: I'm not sure it was 5 minutes exactly, but something like that.) As long as the humans are able to factor the problem into 5-minute-size chunks without too much overhead, they should be able to efficiently solve it this way.
>
> So what actually happened?
>
> The story I got from a participant is: it sucked. The google doc was mostly useless, you'd spend five minutes just trying to catch up and summarize, people constantly repeated work, and progress was mostly not cumulative. Then, eventually, one person would just ignore the google doc and manage to solve the whole problem in five minutes. (This was, supposedly, usually the same person.) So, in short, the humans utterly failed to factor the problems well, exactly as one would (very strongly) expect from seeing real-world companies in action.
>
> This story basically matches the official write-up of the results°.

***Me*:** Here are some comments:

- **I'm not surprised**: I'm not surprised by that result (presuming it's conveyed correctly). But I also wouldn't have been surprised by results that were significantly better than this.
- **Argument-networks don't necessarily need to be efficient**: If some piece of code is explained in an argument-network, it could be that having humans review all the nodes that help explain that piece of code (arguing that the code does what it's purported to do) would be orders of magnitude less efficient than it would be for a human to write that code himself/herself. Typically, actual real-life human reviewers will either review 0 nodes or some small fraction of the nodes.
- **Reviewing one node may often take more than 5 minutes**: When reviewing if something is the case, just understanding the question and the terms that are used may often take more than 5 minutes. And when reviewing nodes, people should take care to make sure they understand things correctly, think things over several times, and so on. The way I imagine it, it may sometimes take 30-60 minutes for an actual reviewer to review a node (or maybe even more), even though that node only does one fairly small step of reasoning.
- **The average reviewer is not what matters**: A score-function need not care about what the average person thinks. Nor does it need to care about the average student at Harvard, the average person who works with AI alignment, the average person at DeepMind, or the average person with 135+ IQ. Suppose that some subset of reviewers are deemed much harder to convince of contradictory claims compared to the rest (and not just because they always answer *"I don't know"*). If so, the scoring-function can focus on that subset of reviewers when scoring the argument-network. That being said, if the kind of person that the scoring-function cares about is too small, then that might make it harder to test the accuracy of the systems that predict human responses.
- **It may be sufficient for a very small subset of arguments to be viable as argument-networks**: When arguing that something is the case, there are lots of ways to do that. Some may contain pieces that aren't sufficiently human-digestible. Some may require types of arguments that too often lead humans astray. Some may contain pieces where it's hard to reliably predict what human reviewers will respond. Due to these kinds of restrictions, it may be the case that converting arguments into a viable argument-network typically doesn't work. But if it works for some small fraction of arguments, that *might* be sufficient.

***Imaginary friend*:** So argument-networks don't **necessarily** need to be efficient. And it **may** be sufficient for a very small subset of arguments to be viable as argument-networks.

It doesn't seem like we should rely on this strategy working out...

***Me*:** I agree. But it could be explored in parallel with other approaches.

Bets as a way to save computational resources

Me: Something that could be possible for AIs to do would be to make bets on the score that they could get, or other AIs would be able to get, if tasked to construct an argument-network that argues for some specific claim(*"I bet with there is a 99.97+% probability that I would be able to make an argument-network that gets a score of 88.2+%"*).

These bets would need to be incentivized in such a way that it doesn't pay off to be overconfident. It would be possible to have game mechanics where AIs compete in a zero-sum game, and each AI can make counter-bets against the other.

One point of doing bets would be to **save computational resources**. And another point would be to try to **answer questions that may be infeasible to answer by brute force**. For example, if some "space" of possible scoring-functions has been defined, it's hard to know which scoring-function within that "space" that has the least "wiggle room". But it would be possible for AIs to be incentivized to try to find the best one, and then they could make bets for how likely they think it is that the other AI is able to find a scoring-function that performs significantly better.

Bets can be made on the outcome of a process, but it could also be possible to make bets on steps that are part of a larger process. For example, suppose that a process has sub-steps that involve predicting human responses. If so, it would be possible to make system-calls to systems that make rough estimates of what those predictions would be if more compute was spent so as to make a more precise prediction (*"I predict at least x% chance that process P would assign at least y% chance to the answer being either A or B"*).

Predicting human evaluations

Imaginary friend: Earlier in this post, you asked me to temporarily assume the following:

> *"We have software-programs that can predict how various humans are likely to respond to questions, evaluate arguments, interact with pieces of code, etc."*

You went on to say that you didn't presume that they would have 100% certainty, or that they always would have a prediction. But you did presume that they wouldn't be overconfident, and yet confident enough often enough to be useful.

I share your assumption that such software-programs probably are *possible*. Indeed, one worry about misaligned superintelligent AGIs is that they might understand us all too well. But do you think it's *probable* for us to obtain such programs in time to use them?

Me: I'm not sure, but I do feel somewhat optimistic.

One thing to consider is that a human answering a question or doing other forms of evaluation can be thought of as an experiment. Which means that we can test how accurate any given system is at predicting human evaluations.

The strategies in this post are agnostic in terms of *how* we predict human evaluations. But they rely on us being able to do it somehow.

One challenge is that when testing the accuracy of a system that predicts human responses, that will presumably involve showing the content to humans - but we may be hesitant towards letting the AI freely come up with content that is shown to actual human reviewers. In other words: among the "space" of predictions that are made by the prediction-system we want to test, there may be large sections of that "space" where we are unwilling to carry out experiments.

In order to address this problem, I have certain techniques in mind, which I refer to as **Bayes-games**. I have an addendum to this post where I write about Bayes-games, and I also plan to write more about Bayes-games in the future.

Something to keep in mind is that **we need not restrict ourselves to one method for predicting human responses**! It's possible to have different systems that predict human responses, based on different architectures and principles. And we could choose to only trust predictions in cases where all the different systems are making converging predictions.

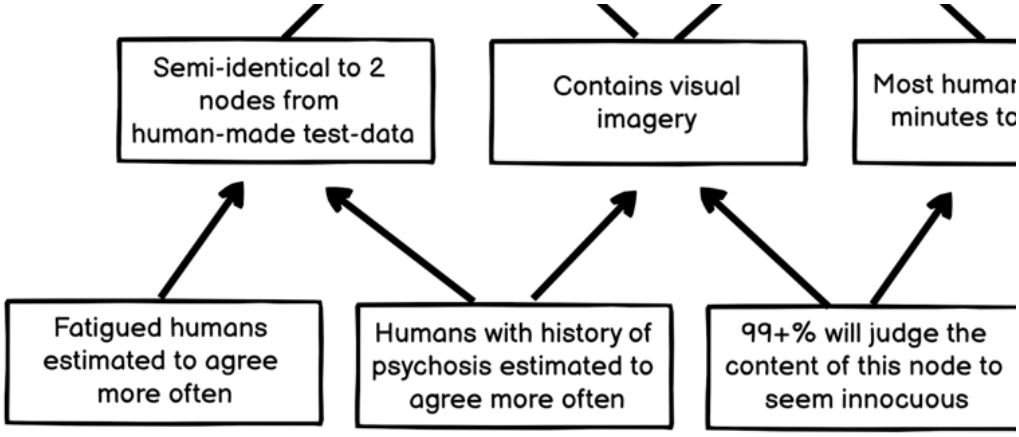
Having actual humans review node content

A worry when interacting with superintelligent AGI systems is that they could "trick" or somehow manipulate human operators. And the more "bandwidth" they have for interacting with human operators, the more realistic that might become.

With a certain disadvantage in pieces, even a maximally good chess-player will not be able to beat a mediocre player. And similarly, maybe there is nothing that an AI could say that would enable it to escape its "confinement" or otherwise cause great harm (especially if it is limited in terms of how much it can say). But we don't know quite how "hackable" we are, and I'd think it best to err on the side of being too prudent.

When dealing with argument-networks, an advisable practice (especially in the beginning) could be for the content of the nodes to be obscured by default. Meaning that the human operators by default would be unable to see the actual node content.

However, they might be able to see info *about* the node content:



Even if we choose to have nodes obscured *"by default"*, we may sometimes see it necessary for actual humans to review nodes (one reason being that we may need to test the accuracy of the systems that predict what human reviewers would answer). Here are some policies that could help reduce risks associated with this:

- **Have people who review nodes be closed off from AI operators and from society**: Risks associated with human reviewers being manipulated or "tricked" is lower if those humans cannot influence how events unfold. So it would be a possibility to have the human reviewers work closed off somewhere, without access to the internet, and without any high-bandwidth communication with the people who operate the AI-system. They would not need to be closed off for the rest of their lives, but for some limited period of time (where superintelligence has been developed or seems like it could be right around the corner, but robust alignment isn't yet in place).
- **Start by reviewing only nodes that seem more safe**: Nodes that don't contain visuals may be safer than nodes that contain only text. Nodes where the content seems innocuous (e.g. only containing math) may be safer than content that concerns itself with more controversial topics. Judging whether a node seems to have innocuous content could be done with the help of various methods and systems, with each of them being able to "veto".
- **To the degree possible, test the accuracy of prediction-systems by reviewing human-made nodes**: When testing the prediction accuracy of systems that predict human assessments, we can also restrict ourselves to have the human review nodes that are based on human-produced arguments. This is a rule that potentially could be practiced strictly in the beginning, and then more and more exceptions could be made with time (as we get more and more "verified" capabilities that help us assess risk).

Btw, keep in mind that the process of generating these networks can be broken into several steps (and those steps can again be broken into smaller steps):

- Coming up with the content of the argumentation
- Breaking and translating argumentation into "pieces", and constructing an argument-network from those "pieces"
- Obfuscating nodes
- Making predictions regarding how human evaluators will evaluate a given node

It's possible to have one system do all of those things, but it may be better to have different systems do different things. In cases where argument-nodes are obfuscated, the systems that construct argumentation don't need to be told that argument-nodes will be obfuscated.

Iterating alignment-capabilities step by step

Me: One of the earlier "strategies" when dealing with a superhuman AGI-system could be something like this:

1. Instruct the AI to try making new AGI-systems based on different alignment-methodologies (some methodologies we suggest ourselves, and others where we are less specific about methodology).
2. Inform the AI that when doing #1, it needs to show carefully step-by-step why those solutions are likely to do what we want them to do.
3. Have a separate system obfuscating argument-nodes.
4. Don't actually look at the arguments, and don't look at any of the code that is produced (although specialized computer-programs can check whether the code that is produced has various properties).
5. Run the AGI-systems that were produced (on separate computers).
6. As new requests are made, give them to both the original AGI-system and the new AGI-systems.
7. See if the different AGI-systems, which are based on different architectures and alignment-methodologies, produce converging output (or if they give answers that contradict each other).

Another strategy might go as follows:

1. Request an AGI-system to output source code for an *"aligned AGI"* (or some other similar request, akin to e.g. *"an AGI that answers requests as honestly as it can, and does not try to hack itself out or manipulate operators"*).
2. Along the request above, the AGI is to also produce an argument-network, which argues that the output that is provided insofar as feasible matches with what was requested. The better this argument-network scores, the better.
3. Don't look at the source code that is outputted, and don't look at the content of the argument-network either.
4. Run the new AGI-system (on separate and closed-off hardware), and use it as one AGI-system in a "council" of AGI-systems.
5. Don't give any of the AGI-systems in the "council" any opportunity to communicate with humans, except for ways that are very restricted and low-bandwidth.
6. See if the AGI-systems in the "council" produce converging output, or if they give answers that are mutually contradictory.

You might think:

> *"But shouldn't we look at the code, and look at the arguments provided by the AGI?"*

And my answer would be:

> *"Yes, we should, at some point. But before we start doing that we should do less risky things that that we can do to make the system safer."*

If we have a system that FOOMs, this could be an intermediate step on the path towards a more assuredly aligned AGI-system.

In fact, all of the strategies I've outlined in this post would be intermediate steps towards a system that's more robustly aligned.

For example: So far I've talked about predicting *human* responses. But it may be advantageous to predict the responses of *idealized humans*. Hypothetical humans who…

- have more short term memory
- exhibit less cognitive biases
- are less prone to optical illusions
- are more cognitively reflective
- etc, etc

But my thinking is that we could start out with predicting the responses of normal humans, and then later expand the system to also predict the responses of humans that are idealized in terms of intelligence and rationality.

And at some even later step, maybe we also are interested in predicting the responses of humans that are "idealized" in a moral sense (emotional empathy, cognitive empathy, impartiality, and so on).

For example, imagine if we could access the experiences of other beings as "memories". Not just humans, but also animals, including wild animals. How might that make us think about various moral issues?

CEV would involve predicting what we as humans would think if we *"knew more, thought faster, were more the people we wished we were, had grown up farther together"*. And I think there is a lot of merit to that kind of thinking.

But for now I think it's best if I focus on earlier steps. Should things turn out in such a way that those steps are used and turn out to be successful…

Imaginary friend: If.

Me: …then maybe they could be part of a pathway towards bigger and better things.

Addendum: Bayes-games

One way to obtain accurate predictions could be through the use of **Bayes-games**.

In these games, AIs would compete against each other in a zero-sum game, competing to make accurate experimental predictions. These predictions that AIs make could be seen as "bets", and when making such "bets" they would express their level of confidence.

More confident bets would lead to more points being earned when they are right, but also more points being awarded when they are wrong. Points would be awarded in such a way that confidence is rewarded but overconfidence is heavily disincentivized.

There is a variety of types of experiments that could be covered by a Bayes-game. But there is one type of experiment that would be of particular interest: Namely asking questions to humans, and seeing what the humans answer.

For example, the humans could be presented with the content of argument-nodes, and asked questions about the content they're shown. And the AIs in a Bayes-game could consider this as an experiment, and make bets on what humans will answer (based on info they are given about the humans in question).

There would be three possible "modes" for a Bayes-game:

1. **Specific predictions**: This is the most simple and straightforward type of Bayes-game. The AIs would make specific experimental predictions about specific experiments, and then they would make "bets" on these predictions where they describe their level of confidence. And each AI could point out bets where they think the other AI has made a prediction that's wrong or over-confident.
2. **Prediction-functions**: Here, instead of making specific predictions on specific experiments, they would output prediction-functions. Each prediction-function would be specialized towards making predictions for experiments that are within some "space" of possible experiments. The AIs would make bets on how well prediction-functions are likely to perform, and the AIs could try to pick out experiments where they think the prediction-functions made by the other AI are likely to lose points.
3. **Inferred predictions**: Here the AIs would assert statements in a format that allows for computational inference. And the AIs could then infer predictions from these statements. There would be a focus on predicting patterns and correlations between *observation-data* and the data-

structures that represent *experiment specifications* and *experiment results*. The AIs could challenge each other to make additional predictions about such correlations.

***Imaginary friend*:** What would be the point of Bayes-games? Why not just a more straightforward approach, where AIs try to make predictions, we test some of those predictions, and select for AI-systems that make good predictions?

***Me*:** The way I think of it, there are several reasons:

1. **Having it be incentivized to point out bad predictions**: When one AI makes a "bet", the other AI would have the opportunity of making "counter-bets", or to otherwise indicate which bets made by the other AI it thinks is most likely to result in the other AI losing points. If the bad predictions are like a needle in a haystack, then maybe this can make it easier to find that needle.
2. **Ascertain quality of predictions, also outside of the "space" where we are willing to test**: In the "space" of conceivable experiments for which Bayes-games make predictions, there may be some "sections" that are available to be tested, and other "sections" that we are unwilling or unable to test. For example, when testing predictions about what human reviewers will answer, then maybe we want to set limits in regards to the AI-generated that is seen by human operators.

***Imaginary friend*:** #2 doesn't seem so easy. Couldn't the AIs guess which experiments that we are unlikely to carry out, and then not feel a need to make accurate predictions for those experiments?

***Me*:** I think they maybe could be able to guess that, and don't rule out them being able to guess this with high condfidence. The hope would be that even if this is the case, there are mechanisms that could help us ascertain how much trust we should put into predictions about experiments that are outside of the "space" of experiments we are willing to carry out:

> ***Source code length minimization***
>
> A more "general" solution will often require fewer lines of code. Not always, but often.
>
> For example, imagine that you are to calculate the area of a square, based on the length of the edges. The fewest lines of code to do that is something akin to this:
>
> ```
> return length_side_1 * length_side_1
> ```
>
> And certainly not like this:
>
> ```
> if (length_side_1 === 0) {
>     return 0
> }
> else if (length_side_1 === 0.001) {
>     return 0.000001
> }
> else if (length_side_1 === 0.002) {
>     return 0.000004
> ```
>
> This principle can be made use of, so as to make it harder to return intentionally wrong outputs for some inputs.
>
> It's easy enough to make a function return wrong outputs for certain specific inputs. That can for example be done by adding a few extra if-sentences to the function.
>
> But what's *not* easy - at least not always - is to intentionally return bad outputs for some specific inputs while at the same time competing to minimize source code length.

Suppose the following situation:

1. You are tasked to write a function that calculates the area of a square.
2. The accuracy of the function you write will be tested. And you get points based on accuracy.
3. When the accuracy of the function is tested, the "space" of the possible inputs that may be tested is constrained. The only inputs that are tested are inputs where the sides have lengths between 3.2 and 4.77.
4. You also get points based on the length of the function, with less characters of source code being awarded with more points.
5. You are competing against someone else. Your competitor also wants to maximize points, and attempts to do so earnestly and competently.
6. You would like to earn more points than your competitor, and at the very least not less.
7. You want the function to "trick" people into thinking that a square with sides of 10 has an area of 43.

In such a situation, #2 would *not* prevent you from #7, due to #3. But #7 would prevent you from #6, due to #4.

Or to say things in a different way: You would not be able to do the three following at once:

1. Passing tests (that test outputs for some part of the "space" of possible inputs)
2. Returning wrong output for some specific inputs (inputs that are guaranteed to not be tested)
3. Minimize source code (so that you don't lose points against a competent and determined competitor)

The example I give here is a simplistic toy example. But I do think the principle extends to more complex and real-world examples.

As a side note, I'll note that there is some similarity here to concepts such as solomonoff induction, konglorov complexity and occam's razor. It isn't the same concept, but it's interesting to note the reminiscence.

I am aware that although this principle *sometimes* works, it does not *always* do so. In some cases, having to minimize source-code length could even force you to overfit.

However, if the Bayes-games competitors are competent and do their best, then it should not pay off to overfit in relation to the "space" of experiments that are available for testing. So one approach would be as follows:

1. Always describe to the AIs which "space" of experiments that actually is available for testing.
2. Sometimes purport the "space" to be larger than it really is, but don't reward or subtract points when you do this (the AIs don't need memory of previous games, or of points that they've earned or lost in the past).
3. Sometimes run games where the space that is available for testing is made extra small (smaller than it needs to be).

If predictions change when we do #3, that is an indication that overfitting is a problem.

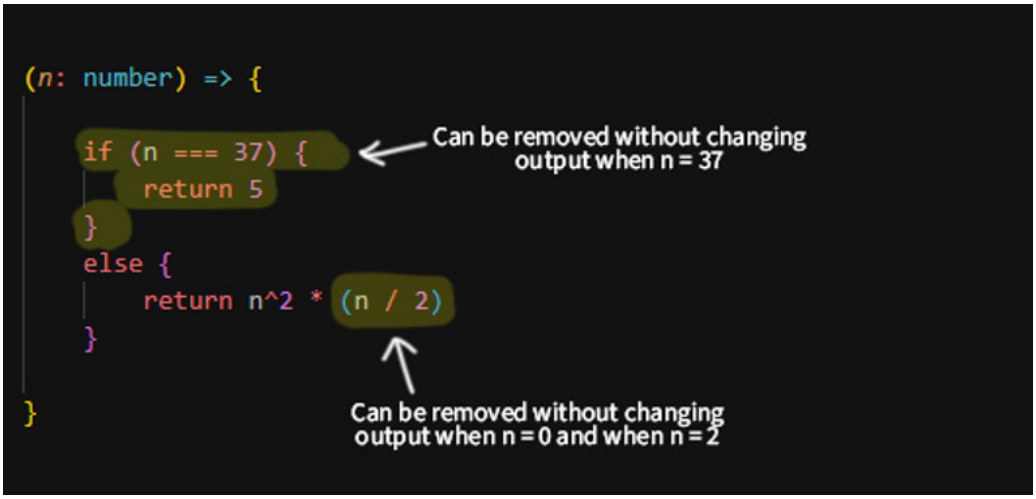
Btw, I have talked now about minimizing source code length. But when minimizing source code length there are several possible approaches:

1. We could have the AIs compete to minimize every individual function
2. Same as #1, but the functions can make system-calls to certain functions that we have provided (without these functions that we have provided counting as part of the function length)
3. The functions can make calls to shared helper-functions that the AIs write, and the AIs compete to minimize the total amount of code (as opposed to minimizing the length of every individual function)

Competing to make code more "general"

Source code length can be seen as one method for trying to approximate “generality”. But there could also be other ways of approximating “generality”.

For example, we could generate “heat maps” of sections of the source code that can be removed or simplified without significantly affecting output (for certain inputs). If it’s hard to remove sections of code (or otherwise simplify it) without significantly affecting output for a large fraction of inputs, then this can indicate that the function is relatively “general”.



Making a “web” of prediction-generators between different parts of experiment data

Consider the “space” of experiments that can be carried out in the real world, and the “space” of experiment data that could correspond to these experiments. There number of conceivable permutations that would be congruent with how the world works - well, it is of course mind-bogglingly huge.

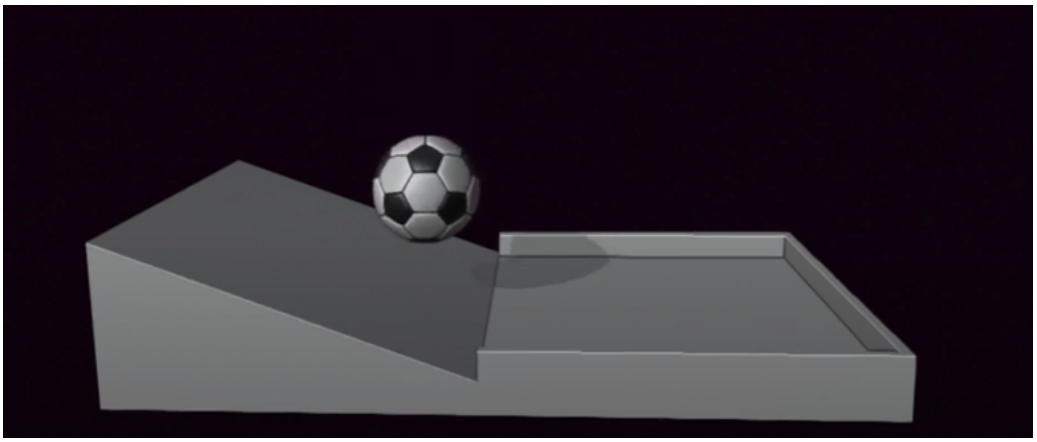
But for every 1 permutation of experiment data that would be congruent with how the world works, there is a huge number of permutations that is *not* congruent with how the world works. So the requirement that predictions are to be congruent with how the world works, is in fact quite restrictive!

When I speak about experiment data, I think of there as being 3 “sections” of data corresponding to an experiment:

- **Experiment specification**: This is some data-structure that contains info about how the experiment is to be carried out.
- **Observation-data**: There are many possible forms of observation-data (video footage, audio recordings, MRI data, eye tracking, keylogger logs, temperature recordings, etc). Which types of data that are included, will depend on the type of experiment in question.
- **Experiment results**: This is some data-structure that describes the outcome of the experiment. This data-structure would be provided by the human operators of the Bayes-game (through their manual labor, or through some system that is external to the Bayes-game).

What we ultimately are interested in, is being able to start out with *experiment specification*, and then be able to make predictions about *experiment results*. But it can be helpful to make AIs predict how various types of experiment are correlated. This way, they make more predictions that can be tested per experiment, meaning that they can earn or lose more “Bayes-points” per experiment that is done.

To make things easier, let’s imagine a simple toy example, where the predictions that the AIs make have to do with the movement of a ball in a physics simulator:



In this case, the data would be as follows:

- **Experiment specification**: Some data-structure that describes the initial conditions of the ball.
- **Observation-data**: Video that shows screencast of the ball as it moves.
- **Experiment results**: Some data-structure that describes where the ball ends up.

As you may remember, I said that one type of Bayes-game would involve **inferred predictions**. And I mentioned that here the AIs would assert **statements** in a format that allows for **computational inference**.

Let me say a bit more about the statements in this type of Bayes-game:

- **One of the building blocks of statements could be "abstractions"**: These "abstractions" would be nothing more than functions and the output of those functions, but I call them something else so that their intended purpose is easier to remember. Sometimes these functions could take experiment data as input. One simplistic example of an "abstraction" would be a function that takes one frame of a video as input, and returns an array with all the coordinates that seem to show the center of a ball.
- **Some statements could describe how "abstractions" can be inferred from other "abstractions"**: For example, if "abstractions" can be used to describe where a ball is on the screen in a specific frame, then a few such "abstractions" can be used to describe where that ball is in a few consecutive frames. And based on this it could be possible to infer the probable velocity and acceleration of that ball, and predict where it will be in adjacent frames.
- **Some statements could be interpreted as testable predictions**: Since "abstractions" are defined in terms of functions that receive experiment data as input, we can then test the reliability of the predictions that are expressed by testing them against experiment data.

In the toy example with the physics simulation, we could for example imagine statements that describe the following:

1. **Predictions where the 1st frame of the screencast-video is inferred based on the experiment specification:** There would be one "abstraction" that describes the coordinates where the ball starts out (based on receiving data from the experiment specification as input). And then there would be another "abstraction" that describes the coordinates of the ball in the first frame of the screencast (based on receiving that frame as input). And the statement could consist of an expression that describes how the value of one of these expressions could be calculated from the other (the simplest example of such an expression would simply be an expression that asserts that their values will be equal).
2. **Predictions where the 2nd and 3rd frame of the screencast-video is inferred based on the experiment specification:** Same as #1, but instead of predicting the placement of the ball in the 1st frame of the video (based on the experiment prediction), we predict the 2nd and 3rd frame.

3. **Predictions where frame N is predicted based on frames N-1, N-2 and N-3**: We could have a statement that describes how to calculate where the coordinates of the ball will be in a given frame, based on where the coordinate is described as being in the 3 previous frames (all of this described in terms of an expression that references "abstractions"). By doing this recursively, we can predict what the
4. **Predictions for what the final results will be annotated as being based on the last frame of the screencast video**: As mentioned, there could be "abstractions" that describe what the coordinate of the ball is, based on receiving a frame of the screencast video as input. And based on receiving the last frame of the video as input, it could be predicted where the ball will be described as end up (according to the data-structure where final results are annotated).

This way, we could predictions of *final results* based on *initial conditions*. In one sense, it would be an unnecessarily elaborate way of doing that, since the AIs could have done that directly (without the detour of predicting observation-data "abstractions". But by taking that detour, they have made themselves easier to disprove. They have described correlations in the observation-data that there is plenty of opportunity to prove wrong (if they are wrong).

Here is how some of the adversarial dynamics could work in a Bayes-game with inferred predictions:

- The effect we want is *sort of* similar to that of a witness that is being examined by an examiner. If the witness is lying, then the more questions the examiner can ask, the more opportunity there is for the examiner to expose inconsistencies.
- In Bayes-games with inferred predictions, each AI can "challenge" the other AI to agree or disagree with various statements (statements that describe correlations between different parts of the observation data).
- Since statements reference "abstractions", and "abstractions" are defined in terms of functions (that can take experiment data as input), there is no need for humans to take on the role of "jury".
- Each AI can infer predictions based on the statements that have been asserted by the other AI (using computational proofs).
- Each AI can propose experiments where they think they themselves will come out favorably compared to the opponent.

When doing Bayes-games with inferred predictions, the AIs would be scored based on factors such as the following:

1. **Internal consistency**: If one AI asserts statements that are shown to be internally contradictory (the other AI can do this using computational proofs), then that should result in a huge loss of points.
2. **Accuracy/precision**: How accurate and precise are the predictions that the AIs can infer based on the statements they have asserted?
3. **Scope of stuff that can be predicted**: Are there areas where one AI makes predictions while the other AI doesn't make predictions one way or the other?
4. **Data that predictions can be made from**: Are there subsets of the experiment data (from the *experiment specification* and/or *observation-data*) where one AI is able to make useful predictions about experiment results, while the other AI isn't?
5. **Indirect ways predictions can be made**: Are there statements about observation-data such that (1) the AI can predict these statements from the data-structure that describes initial conditions and (2) the AI can use those statements to make useful predictions about final results? And are there groups of statements where one AI is able to do this, and the other one isn't?
6. **Complexity of assumptions**: How many axioms does the AI need to infer the various predictions that it makes? And how complex are the individual assumptions that are used (the length of the expressions, source code length needed to represent the "abstractions" that are referenced, and so on)?

As you may remember from our discussion of argument-networks, there was a discussion of *wiggle room*, and different "levels" of *wiggle room* (the wiggle-room could be relative to a score-function, or a score-function-builder, or a score-function-builder-builder).

In Bayes-games there could be similar mechanisms:

- We could see how high score AIs are able to get while making contradictory predictions (if there are AIs that are able to get a high score while predicting *"x"*, and there also are AIs that are able to get a high score while predicting *"not x"*, then this shows that there is wiggle room)
- If we are unsure of what settings to use (for assigning points to the AIs, and so on), we could have a "settings-builder" for choosing settings (from the "space" of possible settings we allow). And we could challenge AIs to find settings that minimize *wiggle room* (while maintaining sufficient *reach*).
- We could explore how much wiggle room there is in relation to any "builder" that is used for choosing settings. First, we give AIs the task *"Use the settings-builder to make settings that result in low wiggle room, and that make it possible to score high while predicting **x**"*. And then we give AIs the task *"Use the settings-builder to make settings that result in low wiggle room, and that makes it possible to score high while predicting **not x**"*.

It would also be possible to have Bayes-games, or games that are reminiscent of Bayes-games, where what the AIs bet on is the result of other Bayes-games. So they can make bets how good of a score an AI would get if playing a Bayes-game with a certain specific type of experiment under certain kinds of restrictions. Or they could propose solutions/modifications for the Bayes-game mechanics (from some "space" of solutions/modifications that are allowed), in an attempt to optimize for certain parameters (wiggle room and so on), and make bets on whether the other AI is able to find something better.

Something that may make things easier is that there are domains where we actually know the answers, and are able to test relatively safely (e.g. with simulated physics that have stochastic elements and where the source code is unknown to the AIs - or maybe physical experiments where any experiment that can be proposed is very innocuous). We could keep the "space" of experiments that the AIs are allowed to propose artificially low, and see if those Bayes-games are misleading in regards to what to expect outside of that "space".

MRI data is one possible form of observation-data, and it could be useful for Bayes-games that predict human answers to questions.

It could potentially be helpful to have pre-done experimental data that the AIs could be tested on. This could include experiments where humans answer questions, with video footage and MRI recordings as observation-data.

I have a few comments where I elaborate on certain topics:

- ***Here** is a comment where I discuss how AI safety by debate differs from the techniques discussed in this post*
- ***Here** is a comment where I briefly touch upon the topics of computational proofs and cluster-concepts (something I hope to write more about sometime in the future)*
- *And a couple of more comments can be found here and here .*

If you found this post interesting, you may also take an interest in other posts from this series .

I appreciate any questions or feedback. I'm also open to video-conversations with more or less anyone who has read this post and wants to talk (feel free to reach out by PM). And if anyone is interested in maybe working further on anything described in this post, then I'd be happy to keep in touch over time.

Thanks to Evan R. Murphy for helping me review early versions of this text!