/mlst

Thomas Ahle (Normal Computing) — Auto-Formalisation, Verilog Agents & Thermodynamic Compute

June 28, 2026



Machine Learning Street Talk

Podcast Transcript

Revision: d7a69e9b

Topic headings include retention indicators: ● above average, ● average, ● below average (vs. similar YouTube videos)

Interested in licensing ReScript for your podcast? Get in touch with Tim Scarfe.

Contents

Overview 2

Meet Thomas Ahle: the Lovable for chip design 3

Why hardware needs formal verification 5

Ten thousand dollars per core and a six-month agent run 6

Rebuilding programs from tests: ProgramBench and zero percent 7

Structure vs competence: can you learn a program from behavior? 9

Continual learning, abstraction, and Claude as an ecosystem 11

Autoformalization and the AlphaProof trick 15

No single true representation: specs, Petri nets and TLA+ 17

Thermodynamic computing: when noise is the computation 20

Bayesian uncertainty in the age of token streams 21

Hybrid compute: vibe-coding loops, binaries and Stockfish 23

Co-design, central-AI apps and API pricing 25

Chain of thoughtlessness and the Chomsky hierarchy 27

AI psychosis, slop and the broken social contract 29

Typing it yourself, teamwork and performance vs competence 31

References 34

Overview

Thomas Ahle wants Normal Computing to be the Lovable for chip design: type your intent, and a swarm of agents carries it from design through optimisation, formalisation and verification to tape-out. To get there, his team at Normal Computing wrote their own open-source Verilog simulator, 580,000 lines in 43 days, because commercial EDA verifiers run about $10,000 per core and there are no decent open-source compilers to build on.

That sets up the question Tim keeps pressing: if an agent can produce a chip design, a proof, or a working program, how do you actually know it is correct? Passing 70% of tests is not the same as being right, and a single fabricated bug can cost a company a fortune. They dig into ProgramBench (rebuild a program from its tests, roughly 0% success), the difference between structure and competence, and the "understanding debt" you take on when nobody reads the code.

From there: auto-formalisation in Lean and the AlphaProof trick of training on prove-or-disprove; why there is no single true representation of a spec (Petri nets, TLA+, Erik Curiel's "math does not represent"); and thermodynamic computing, where Normal Computing's CN101 chip is built so that its physical noise *is* the computation, settling a stochastic differential equation in hardware to invert a matrix. Plus Bayesian uncertainty, specialisation, the Chomsky hierarchy, AI slop, and whether performance is all that matters.

Recorded in Zurich.

Disclosure: Normal Computing paid our production and travel costs for this show. We retained full editorial control. They did not see the video before publication, and we did not show it to them or discuss it with them beforehand.

Meet Thomas Ahle: the Lovable for chip design

Thomas Ahle 00:00:00

Yeah. So why not try and build a chip that's just inherently random?

Tim Scarfe 00:00:04

Meet Thomas Ahle. I caught up with him in Zurich, and he's 1 of these rare galaxy brain people who's comfortable in probabilistic machine learning, formal verification, and chip design. However, there's a small problem. When the token god hands you something that looks like it works, how do you know it's actually right?[1][2][3][4]

Thomas Ahle 00:00:27

So my background is in theoretical computer science. I used to do algorithms for high dimensional data, locality sensitive hashing. Then I moved to Normal Computing to develop thermal computing, also to speed up Bayesian intelligence. Sometimes I think about it as the Lovable for chip design. So we take it all the way from your intent through the design, through optimizing your design, to formalizing and verifying your design, all the way to tape out.[5]

Tim Scarfe 00:00:52

Now I didn't fully appreciate this before. These days, a chip doesn't necessarily start in a factory. It can start as code. So engineers design the whole circuit in a language called Verilog, almost written like software and only much later does any of it become physical silicon. But 1st, that code has to be simulated and formally verified. It has to be proven correct because once a chip is fabricated, if there are any bugs, you're in big trouble. So a few months ago, Thomas blogged about building a Verilog simulator using a swarm of AI agents collaborating with each other and it generated over half 1000000 lines of code in 43 days. Now, the reason he needed to do this is that commercial software costs a ridiculous amount of money and isn't very friendly to using agents. $10,000 per seat or

[1] Thomas Ahle — person Guest; theoretical computer scientist who moved from high-dimensional algorithms to thermodynamic computing at Normal.

[2] Normal Computing — organization Thomas's company, building AI for chip design from intent to tape-out and developing thermodynamic hardware.

[3] Subsets and Supermajorities: Optimal Hashing-based Set Similarity Search — paper Representative of Thomas's earlier work on locality-sensitive hashing for high-dimensional data.

[4] Clustering the Sketch: Dynamic Compression for Embedding Tables — paper Another strand of Thomas's pre-Normal research on high-dimensional data and compression.

[5] Lovable — tool Thomas frames Normal as the Lovable for chip design, taking you from intent to a finished design.

something.[6]

Thomas Ahle 00:01:48

Yeah. Per for 1 CPU kernel.

Tim Scarfe 00:01:50

If an AI can generate a chip design or a proof or a working program, how do you know it's actually correct?

Thomas Ahle 00:01:58

The percentage of tests that it got right or like even like the Fable, it was oh yeah, it got like 80% tests correct. It's like, yeah, but I talked with the people who made the benchmark and like, but did did it get any of them actually right, you know, like

Tim Scarfe 00:02:11

Now in chip design, noise is the enemy. Manufacturers spend a fortune getting rid of it. But in thermodynamic computing, the opposite is kind of true. The noise is the computation. It comes straight out of probabilistic machine learning which already runs on randomness and uncertainty and thermodynamic computing tries to make the chip itself a stochastic differential equation. Let the chips own noise settle into place and then it can land on answers that would normally cost a fortune to compute.

Thomas Ahle 00:02:47

And then you infuse all of this noise, and it'll start to behave according to to to these stochastic differential equations. It actually behaves sort of according to the inverse of that matrix.[7][8]

[6] Building an Open-Source Verilog Simulator with AI: 580K Lines in 43 Days — other Thomas's blog on generating a Verilog simulator with a swarm of agents, the launching point of the conversation.

[7] Normal Computing Announces Tape-Out of the World's First Thermodynamic Computing Chip (CN101) — other Tim's narration introduces CN101, Normal's first thermodynamic chip taken to silicon.

[8] World's first thermodynamic computing chip reaches tape out — other Press coverage of the CN101 tape-out referenced in the chip's introduction.

Tim Scarfe 00:03:01

They've already released their 1st version of this technology. It's a chip called CN101 all the way to silicon. It's still early days and it's aimed at a fairly narrow band of probabilistic workloads. But the real test is what can the benchmark say once they scale this up? And something of huge interest to us at MLST is, if we're gonna let AI build everything, what kind of understanding do we actually keep in the overall process?

Thomas Ahle 00:03:28

It's not just that it's getting smarter, it's also that humans are getting dumber.

Tim Scarfe 00:03:33

Now I should disclose that Normal Computing very kindly offered to cover our production and travel costs for this show, but we kept full editorial control.

Why hardware needs formal verification

Thomas Ahle 00:03:41

These are tools for device manufacturing. So we're it's things like I mean, it's compilers for RTL. It's like the where RTL is like the programming languages you use. So if you're building a new chip, you typically don't draw all the different gates and so on. You write it in programming language like Verilog, which has sort of constructs that are quite, it's like a super parallel programming language which have a lot of constructs that map well to hardware. But then there are actually no good open source compilers for this system. The whole hardware industry, doesn't have the same open source feeling as like software where all the good stuff is free and there's like this big vibrant market of tools here. It's much more locked down to like big providers. But people then, yeah, they compile this too and they get like a netlist, they can start building their schematics, they can also look at it, you get like these giant things that you can eventually send to the fab. But yeah, so this, but they also need to do like do simulations for example, for testing, you can do like formal verification.

Tim Scarfe 00:04:57

Let me just play that back so I understand it. So so Verilog, it's a bit like a programming language for describing a circuit. And what we might probably before we go and get that fabricated is we wanna do simulations because it's very expensive to get it fabricated. So yes, we can design the circuit and do simulations.

Thomas Ahle 00:05:16

Yes, exactly. It's super expensive too. It could cost you what they said like there was this famous Intel division bug in the 90s that probably cost them between $500,000,000 and $2,000,000,000 and I mean that's a lot of money in the 90s. I think there are other examples later on that nearly bankrupted some of these companies like TinyBucks that managed to make it through to the to the fab and it's yeah, it's it's a really different world again from software where we just, know, people fix stuff in prod, you just move fast and and break things. It's definitely like yeah, people are much more paranoid in the hardware industry, which is also why it's a cool place for formal verification, formal methods to thrive.

Tim Scarfe 00:06:06

Okay. So you're saying at the moment there are these commercial, you know, verifiers and simulators and they cost a ridiculous amount of money. So is is it something like $10,000 per seat or something?

Thomas Ahle 00:06:16

Yeah. Per for 1 CPU kernel. Right. Dollars. Yeah. Like, say you wanna scale this up in a data center with 1000000 agents running like you're gonna like I mean, computer is already expensive but not that expensive. What is that $10,000,000,000 or something, right? Just for those licenses. And I think actually that's 1 of the reasons also

Ten thousand dollars per core and a six-month agent run

Thomas Ahle 00:06:36

why AI still is not as popular in the hardware space because they haven't been able to train like the models are not as trained to this kind of workloads because they just, it's not feasible. Like you don't have all of the open source code out there to start the training, but you also don't have the tools that they need to learn to use. You don't have the, yeah, all of the, like the tasks, like you can't just, all of the reinforcement learning on top of it. I mean I mean they they clearly are doing some of that. Like we can also see from model generation to model generation that they are getting better, but it's just it's kind of day and night between like Python or JavaScript. We've been sort of developing these EDA tools in house using AI, having, you know, think I probably have the world records of the longest running agents having like some 20 GPT agents running for around 6 months

now, still making progress. So as I understand correctly,[9]

Rebuilding programs from tests: ProgramBench and zero percent

Tim Scarfe 00:07:40

so similarly to Anthropic, Carlini had this blog post out and they had a functional specification of a C compiler and they had 40,000 agents, and they reproduced the C compiler. You did a similar thing for an EDA tool.

Thomas Ahle 00:07:52

By the way, right?

Tim Scarfe 00:07:53

Well, that's an interesting thing as well. Because this is what I want to get to, right? It's tantalizing. That that there there are there are some domains that are so well evolved that we have, you know, that might be very complex, but but we have a functional specification, and and it's reasonably coherent. So so the idea is we we get a shitload of agents, and we reproduce the function of this software based on these tests, we do it recursively, agentically, and and and so on. Now, my my contention with this is I think that it's not about where you end up. It's not about the functions and the tests passing. It's about how you got there and how structured it is. Yeah. So there is this tendency with agentic coding to build a spaghetti monster, which seems to work. But because if you think about it in in this project you're talking about, I think you said there's, 500 or more, 500,000 lines of code. And in 5 years' time, there's all of this code that probably, you know, you folks haven't read most of it. What you know, that that that sounds like a problem.

Thomas Ahle 00:08:53

Yeah. I think for a lot of this stuff, we're relying on the hope of some kind of escape velocity from code complexity that the models are gonna keep improving faster than our code gets messed up. I had those 3 days of you know Fable access and it was really good like it definitely cleaned up a few things that GPT-5.5 had messed up.

[9] Building a C compiler with a team of parallel Claudes — other Tim's parallel: Anthropic reproducing a C compiler from a functional spec with thousands of agents, the same recipe Thomas used for an EDA tool.

Tim Scarfe 00:09:16

But but but did it though? Do you think it could be deceptive? Because you know when when Fable comes out, all of a sudden 4.8 looks terrible, and when 4.8 came out, it looked amazing compared to, you know, 4 4.6. So so it it there's something deceptive about it, bit, you know, it's like a parlor trick.

Thomas Ahle 00:09:34

Yeah. No. It's true. Like, in principle, you could all be be smoke and mirrors. I mean but I think that's why it's good to have also hard test, right? So I could also see that it was actually suddenly I was making more progress on the more objective tests than I hadn't seen for a while with the other models. So I guess that gives me some trust. But it is, yeah, it is weird to sort of not have the same in-depth level of understanding everything in your code and, you know, definitely miss it.

Tim Scarfe 00:10:02

And I'm interested in what the consequences of that are because you you said you put this blog post out earlier in the year. And then when we had a chat on the phone, you you you said that you you noticed there were some things that that were that were wrong. Yeah. And it isn't there's a this tendency, though, to, accumulate understanding debt. And when that happens, you I think from an evolutionary point of view, you're stuck. Because I think deep grounded understanding of how things are are the basis for the next design decisions, the next evolution. So do you get into this kind of fishbowl thing where, like, now you're in no man's land and you don't really know what to do?

Thomas Ahle 00:10:38

Yeah. I do think it's important to understand as much as you can and have time for. I mean, think for compilers, there is some very important architectural design decisions. And then there's a lot of sort of just implement like these like 100 different functions. Like they all just, especially like modern compilers, it's like layer by layer by layer. You have to lower these things from like the front end level to the middle interpretation to the back end layer. And a lot of that, I could definitely don't want to have to write like it's it's it also paralyzes well. Think I mean I'm sure this could be 1 of the reasons why Anthropic also picked it. We talked about this program bench thing I think Facebook where it's a benchmark where the task, I think they took 150, 170 programs. Some of them are really complicated like FFmpeg is 1 of them. The AI just has to re implement them without Internet access. Yeah, basically when it came out they had like all of the LLMs got 0% because none of them like were able to pass all of the tests. But I've seen like people, like I very rarely see that when people post benchmarks for this thing. They always post this like, oh the percentage of tests that it got right or like even like the Fable, it was like, yeah, it got like 80% tests correct. And it's like, yeah, but I talked with the people who made the benchmark and like, yeah, but did it get any of them actually right, you know? Because it's, if the program only passes 70% of the tests, it's probably not right. In a way,[10]

Structure vs competence: can you learn a program from behavior?

Tim Scarfe 00:12:15

this is the thread that we were talking about before. You know, like for the last 60 years, going back to behaviorism, there's been this kind of thread between structure and competence and prediction. Right? So, essentially, this program bench is making the argument that you can learn the physiognomy of a program from its external behavior. So you can learn the deep structure, the constraints, and so on. And I suspect that that is not possible. Unless, of course, the LLM already knows about the source code because it's in its training data or whatever. But what do you think about that?

[10] ProgramBench: Can Language Models Rebuild Programs From Scratch? — paper The benchmark where models re-implement programs like FFmpeg with no internet and initially scored near zero, central to the benchmark-skepticism point.

Thomas Ahle 00:12:49

Yeah. I think I think humans do it all the time, actually. I think there's the whole field of kind of reverse engineering where people, yeah actually I listened to something with the FFmpeg people where they're talking about all these codecs that they put in there. And they like a lot of time they have no idea what they do like it's just like this obscure blob of code like they probably don't even have access to the code. They just have a couple of like example, video examples that were encoded with this. Maybe some people had shared some screenshots like with clips from the movie. And then they had to go and try and read this like encoded blob and being like, what is it doing? Like how could we like write a program that decodes this thing? And somehow people are able to do it. You know we must have some sort of prior, right? Like what what are the typical things people put in a video codecs? Like there's probably some you know know, Fourier transforms of some stuff where they're probably like you chunk things up or I don't know, I haven't done that much video encoding. Yeah, just like completely black box, just try and keep improving your program and visualize the output until the end of movie comes out and you have an idea of how this probably looks how the movie is supposed to look.

Tim Scarfe 00:14:12

Yeah, That's an interesting thing as well. So we can hill climb in a partial knowledge regime, right? So we might have a verifier. And as you say, like a lot of science is a form of intelligence where you're using MATLAB and you do like a Laplacian or you do an image plot, you take a distribution and it takes shape 1 step at a time. So what you're doing is you're taking steps into the unknown and every single it's like there's a generation discrimination asymmetry. Right? So we can discriminate well, but we can't generate yet. But we take these steps, and then when we get there, we now kind of collapse that into a model, and then that's the artifact of the intelligence.

Thomas Ahle 00:14:48

Right. It's like star search or something where you kind of, you, like, prune your search tree as you go, and, like, what what seems most successful so far, like?

Tim Scarfe 00:14:57

Exactly. But to your point, every single step of the way, we're using the toolbox. So, know, there are some folks that are really good at riddles or intelligence tests, and a lot of it is skill, right? Because they they have a toolbox of abstract tricks that they use, and but but then there's this question of, maybe that's all it is. Maybe there are these natural patterns, these abstractions that we can compose together and handle any kind of novelty in the intelligence setting. Yeah. So therefore, maybe language models could do this autonomously. I think they possibly could,

Continual learning, abstraction, and Claude as an ecosystem

Tim Scarfe 00:15:27

yeah.

Thomas Ahle 00:15:27

This is good point, because I tried to get an answer, I think, from Noam Brown at some point on Twitter about this because you see how better pre trained model you start with, and then they start reinforcement learning on it, and it seems like it gets up to some Like the reinforcement learning works better and better depending on how good the model was to begin with, in a sense.[11]

Tim Scarfe 00:15:51

Yeah, it's super interesting. I mean, I agree with you. I I think it's wrong to say that LLMs are not intelligent. Right? Because there is some kind of a combinational closure, and that means from the primitives in an LLM, can do hill climbing, and we can build some computational structure to solve problems. But that's not quite it, and it can happen at different levels of abstraction. So if the LLMs have higher abstractions, then they can, you know, traverse the combinational closure of the higher abstractions, they can solve the problem. What seems to happen though is, I mean, there's no continual learning, so those don't get, you know, added to a library and reused later. But there's also no abstraction. So what humans would do is they would they would look at this computational graph, and they would say, ah, I see that that's just, an analog of this thing over here, and I and I'm now going to compress it into a new variable that screens off all of

[11] Noam Brown — person Thomas references trying to get an answer from Noam Brown on how RL gains depend on the strength of the pre-trained model.

that complexity. Why do language models not do that?

Thomas Ahle 00:16:40

I think I think you're getting into like continual learning type thing which is is obviously a big thing we lack. I think, I mean, I do think language model do that during pre training. I do think like that's why for example, Methus is so good at at doing stuff in our codebases is because it has somehow seen a lot of code and it just seems to have this sort of intuition for bugs and problems and so on. And that must be kind of like that. It's seen the patterns before, it has some kind of abstract. But it's true like when you actually are doing sort of running them more live, they don't do it very well. Like they don't, and I think a lot of people these days are thinking about this, how can we do better continual learning so it can just build all of these abstractions and these things really well on the fly. I mean I know some companies are actively against trying to do that also. Why? Like I think Anthropic, right? I think Dario said like he considers it like a big safety issue too because you're going to lose, you could easily lose all of the work that has gone into alignment if the agent is learning too much on the fly. Then it gets further and further away from the sort of safe checkpoint. But I think there are enough other people working on it that it will probably happen. But it's definitely like a big unknown, right? And I think it will also have to change a lot of things in how we serve models, for example, because suddenly you need to be able, like if we mean by continual learning that we actually update the weights live, it's going to, it causes a lot of problem for the current paradigm because then suddenly you can't use the same weights for all customers too. I mean, guess thinking machines has this thing where they have 1 shared model and then like some loras on top. So maybe something like that can work. But it's still like you have to somehow, yeah, keep doing this all the time. So we are also working in Normal Computing on like alternative forms of computation. Like these thermodynamic chips or just sort of unconventional computing. It actually becomes sometimes hard to keep the memories in these analog resistors unless you keep learning at the same time as you're doing inference. Like you have to use these fancy substrate like memory stickers or something if they have to be permanent. But if you're just using like sort of basic capacitors like you have in DRAM or something that require constant refreshing. So either you just have to spend a lot of energy on that or you just want to keep the learning going forever. That way they kind of automatically refresh. And that's probably more similar to also how brains work, right? Like they don't, they never, they never like just[12]

[12] Anthropic — organization Discussed for treating continual learning as a safety risk and for the co-working team's everyone-builds-their-own approach.

Tim Scarfe 00:19:31

don't the same.

Thomas Ahle 00:19:31

Or Chomsky or somebody would say about it. If it ever just freezes. I think like the the synapses and stuff, they're always adapting.

Tim Scarfe 00:19:38

Yes. Exactly. I think, out of all of the words we can use to analogize intelligence, adaptivity is the number 1. And and in a sense, when we look at Claude, you can look at, you know, is Claude the model? Is it the ecosystem? I think it's the ecosystem. Right? So it it it is adaptive because We'd

Thomas Ahle 00:19:54

say it's the personality or the persona or something.

Tim Scarfe 00:19:56

Well, you know, there there's a tendency to anthropomorphize it. But but no, like, millions of people are using Claude code. Because this is tangentially related to how we, you know, how we come up with causal attractions in the world. So, you know, the way we do it is we are in the world, and we are agents, and we can make decisions, and we can, like, you know, reconcile uncertainty, we can say, what would have happened if I did this? And I can share it with my friends, and and there's this wonderful percolation process where the abstraction has just become embedded. Now, that does happen with Claude code. Right? Because there are people using it on their machines. They're actually running tests. They're doing counterfactuals. And then all of this gets retrained in the next version of Claude. Then someone else will reuse that abstraction. We have the adaptivity as a system.

Thomas Ahle 00:20:40

Like this brain, shared brain, where all of the experiences go what is called the pork or something. Yeah. Go back off and then

Tim Scarfe 00:20:48

yeah. Yeah. So, but but it's interesting to make you know, to to discuss, is that effectively the same? Like, if we actually had some hypothetical real time adaptive divergent Claude, you know, would it be much better than the Claude that we already have? And, it's also related to the to the work that you guys do, because, the process of intelligence, in my view, is the creation of these coarse grainings, skills. And that's kind of like what you guys are doing with ASICs. So you're building this customized hardware for making certain types of computation go really, really quickly. And it's almost like that's the result of intelligence. So you say, I'm gonna take a very, very complicated thing, and I'm going to whittle it down and represent it in the best way I can, and then I'm gonna bake it into a non adaptable hardware substrate. Is that fair?

Thomas Ahle 00:21:35

I think that is 1 that is what we're working on right now is this this idea of of trying to to build hardware that best fits like the models we have right now and just like really make a super efficient inference. You could say like the in the past at least the issue has been you stop yourself from innovating on the software side, but if you lock down your hardware too much, right, like the NVIDIA chips has been pretty good for innovation, like they're pretty flexible. Of course, they also have guided the way we do AI in a lot of ways like towards matrix multiplications and so on. Still people have been able to innovate. But on the other hand, I think also hardware is getting easier and easier to make with these kind of AI for EDA tools. So it's sort of shortening the like maybe now people are thinking about like making their cooler kernels, right? It's like it's not so unreasonable to think that soon we'll just be doing our AI for making like instead of CUDA kernels, we'll just like make some custom circuits for every single thing we want. So if you come up with a if you want your new algorithm to run really fast, you just design like a specialized circuit for that. And the AI helps you like optimize it and check that it's correct. Of course you still have the fab, but people are we're getting better at like batching things for the fab. I mean I also think having the really flexible hardware is gonna be very interesting, like the stuff where it's learning on chip and ideally like maybe the more adaptability that it has and the less we need to bake in. It's I mean, don't know. It's just more cool, I think.

Autoformalization and the AlphaProof trick

Tim Scarfe 00:23:17

Yeah. But it isn't that a wonderful example of this recursive self improvement? Because as you say, like, we're we're we're building the AI and then the AI is helping us build better kernels, better software, better hardware, which then in turn makes the AI better and then, you know, you get this this kind of loop. It is a recursive self improvement in a way. Right? Exactly. Exactly. But but but we should bring this to life. And so there's this concept called auto form formalization. And and and you guys on your website, you say basically that you have done something similar to AlphaProof in respect of building chips. And roughly speaking, AlphaProof is when, you know, they it was they won silver at the IMO in 2024. And what they did back then was they used a language model to generate, a bunch of lean code. And then obviously was a little bit messy, so some of the formalization was by hand and they did multiple renditions and so on. And then they would do verification with lean. And you're kind of doing something like that for chips.[13][14]

Thomas Ahle 00:24:11

Yeah. Yeah. So it's interesting because there is the auto formalization which we define as taking human specifications and writing up and turning them into formal specifications in Lean for example. Then, but then of course you also need the proof step where you provide the yes, prove that your code, whatever you have actually satisfies that specification. And I think some companies like Axiom, for example, they're very focused on this part, right? Like it's and also in some sense AlphaProof, that's also the main thing it did was it started with a formalization and then the hard part was training the model to provide a proof or disprove. I think it's actually really nice trick in AlphaProof was that when they did the formalization of the proof, it didn't really matter if they got it right or wrong because if they just asked the model to provide a proof or a disprove. So if they got it wrong and it was no longer true, then it would just prove that it was not true. Like it would that was like as you could still use it. Of course, like when they actually did the IMO challenges, they wanted the form out of formalization to be correct. So then they did it by hand. But they didn't need it for the training, which I think helped scale it up. I think, yes, so can do a similar thing with hardware by the way. It's pretty easy to just take some chip design and then you

[13] AlphaProof: AI achieves silver-medal standard at the IMO — other The template Thomas compares to chip autoformalization, including the prove-or-disprove training trick.

[14] Autoformalization with Large Language Models — paper Underpins the autoformalization idea of turning human specifications into formal statements in Lean.

know, you can basically just come up with some properties that may and may not be true. And then you can train the model to try and prove or disprove that this thing holds. So then, but that's all about creating the proof. Then the ortho-formization, in some sense it's harder because, it's all like because it's harder to create the training data for it, right? Like I think that's kind of also a story about AI in the last 2 years since reinforcement learning. Like anything you can create good RL environment for, you can probably learn. But anything else is like out of reach right now. So some of these chips have thousands of pages of specifications. And if you want to turn that, yeah, you're turning that into a formal model. It's not, it doesn't really like if you got just a couple of words wrong somewhere or a couple of numbers, then it doesn't work. Like then what you prove is not important or is not relevant. And I think, I mean it's always been an issue for, in the chip industry like that. And they've kind of tried to solve it by having orthogonal teams. So they have 1 team that's designing the chip and 1 team that's designing the tests and another team that's designing kind of tests of the tests, like where they kind of like coverage, they call it functional coverage, where they measure what the tests test and then check everything off. So hopefully like if all of the 3 teams have read something the same way and understood it the same way, they have the right idea of it. I mean you can try and do something similar with AI. You can argue whether it's really orthogonal if it's the same model that are doing each of the 3 jobs. I do think it is quite orthogonal. Like I do think there is enough entropy in these models when you do long runs And you can definitely find a lot of bugs by just doing things many times and see if there is agreement between them. But it's something, but it's definitely like a trust exercise too and human exercise to figure out how do we get the hardware engineers to trust and like how do we make it easy for them to verify that our formal model fits with what they thought. And you can come up with, yeah, there's all kind of different tricks you can come up with both to make it more visual like what your understanding is and like whether it matches and like ask questions and see if you agree with the questions or show examples or you can also try and do some kind of back and forth. I don't know. Lots of tricks you can try but it's an interesting problem and I feel like no 1 like the Alpha Proof and all of these other people, they never really solved this because their math statement for just 1 paragraph, they were not like thousands of pages long.[15]

[15] Gemini Deep Think achieves gold at the IMO — other Tim notes the gold-medal model did no formal verification, raising whether verification is a spectrum.

Tim Scarfe 00:28:31

Oh, yes. Yeah. Interesting. Yeah. Because with AlphaProof, that I think they wanted they wanted to use Lean 4, and there was hardly any stuff for Lean 4. So they they created a converter from Lean 3 to Lean 4, they needed lots of human, you know, people to fix it. And but but I guess my point is that they needed to fine tune a language model on a ridiculous amount of Lean code, and then they were using Lean as as an intermediate. Interestingly, like with their new model that they won the gold on IMO, they weren't doing any verification at all, not not not in a formal sense. But but then, is there a spectrum? Because I don't the way you were just describing it, it's not like it's not like it's a binary. Right? So you were you were talking about test coverage and, you know, different perspectives, you know, the blind man and the elephant, you know, so there there are these functional tests and these descriptions, and we can do visual inspection and so on. So will it will it end up being a case where we're always wrestling with something we don't completely understand, but we're using as many signals as possible to get there?

Thomas Ahle 00:29:27

Yeah. I think where we're

No single true representation: specs, Petri nets and TLA+

Thomas Ahle 00:29:31

going now is also trying to encompass more of the spec creation, right? Like obviously when we start out, people already have written down all of these specs and we want to help them out with those. But when you do that you also lack, you miss out on the whole intent that got into and the whole process of creating the spec. You don't know, you know, if there's just some numbers somewhere in there. You don't know why they chose those numbers and not some other numbers. And I think you sort of by internalizing more of the process, at some point you can at least be as safe as any human could have been about this chip, right? Like there might also just be some ambiguity that no 1 cares about.[16]

[16] Math Does Not Represent — other Tim invokes Erik Curiel on multiple orthogonal representations of general relativity to make the no-single-true-representation point.

Tim Scarfe 00:30:11

The ambiguity thing is interesting. There's a wonderful talk by Erik Curiel called Math Does Not Represent and you know, he was talking about generalized relativity and he was talking about 4 different, you know, completely orthogonal representations of that. And it was a similar thing actually with alpha proof, so, you know, like how many ways are there in lean to represent non negative numbers? Apparently, there's quite a few. And so the Google guys were just just to stimulate the model, they were representing the problems in in different ways. And this seems like this is almost an

Thomas Ahle 00:30:42

training data in a way, like, right? If they auto formalize the same problem in different ways, they actually get different problems.

Tim Scarfe 00:30:49

Both both in in terms of the intelligence, so how can we kind of bootstrap the model to do better intelligence, but also in terms of the legibility and abstraction of the final output. So I I guess when people think of auto formalization, we we have this almost, you know, kind of idealist view that there is a 1 true representation and it's just gonna be legible and and everything. But but it seems quite vague.

Thomas Ahle 00:31:13

Yeah. It's a 1 true representation. No. I I I guess definitely definitely there's not 1 true representation. With with smaller chips, like, say you have like just like floating point or a crypto chip or something. I mean the specification is actually pretty simple for those. It's more when you come up to like the big like system level stuff. I think software is in many ways ahead of hardware. I mean in terms of formalization they're behind, but in terms of thinking about these like architecture and architecture together with AI, right? I think a lot of people now they'll run their architecture documents by the, by Claude or somebody and being like, hey, what do you think of this? Should we move things around here? And I think this whole discussion like it sort of helps the models understand like what your intents are and what you care about and what you don't care about.[17][18]

[17] Autoformalizing Memory Device Specifications with Agents — paper Thomas's own paper on the DRAM/memory-spec work Tim references as the article he published earlier.

[18] DRAMBench: Autoformalizing DRAM Specifications with Timed Petri Nets — other Supports the surviving DRAM and timed-Petri-nets discussion of protocol-level memory properties.

Tim Scarfe 00:32:02

And just to give us an example to bring it to life, was that DRAM article you published earlier. The article was talking about these timed Petri nets. And I did look this up on the Internet. Apparently, it's a thing from the 19 sixties for describing distributed systems. Explain

Thomas Ahle 00:32:17

that. What we talked about before with the formalizations is very it's like at the RTL level at the very sort of cycle level where you care about, yeah, you want to prove that the exact thing happens at every single clock cycle. But of course, there is another kind of formula and thatis very important for things like adders or arithmetic circuits or crypto. But a lot of the hard problems people have are more of these kind of system problems, more like protocol level stuff, like the things actually maybe also more popular in software. It's been like proving things like, oh, the system can never deadlock or livelock or of like protocol level properties. And like all of the sort of timing requirements between different things makes sense and there's no internal inconsistencies. There so currently there exists different formal languages for these things like for the cycle level stuff. People use these system very log assertions, SVA type things. For the higher level protocol things, there are more classic things like the TLA+ I think by Lamport or something. And yeah, so we try and formalize things in both ways because it can have different it's useful for different things. It's possible at some point you can all merge if you supply. It's kind of like you're doing very high level math and then you can reduce it all the way to the actions if you want to. But I think it's also interesting. Yeah, I mean it's a very new field, so we're trying to explore different ways and these timed Petri nets is 1 way to represent like these super parallel systems that you have, for example, these memories where all these different banks, you can do, activate, like for example, if you activate 1 of these rows, have to wait for the data to run down to the bottom before you can read it, for example. Then after a while you have to refresh it because again it's like these capacitor DRAM cells have to be refreshed all the time. But you can also be working in a different memory bank. There might be like lots of them on this chip at the same time. So there's certain like but then there might be some circuitry that's shared between the banks and so certain things that are interbank dependencies and intra bank dependencies and also bank groups. And people build really crazy stuff actually in there.[19]

[19] TLA+ — tool Thomas names TLA+ as the classic language for higher-level protocol properties like deadlock and livelock.

Thermodynamic computing: when noise is the computation

Tim Scarfe 00:34:43

Very cool, very cool. I mean, what's really exciting to me is that it's possible to build chips that can do certain types of things orders of magnitude faster. And and that's why I wanna talk a little bit about thermodynamic computing. Right? So, you know, so apparently, instead of forcing transistors to settle at 0 or 1, you let noise do a random walk and bias it, so the chip is a stochastic differential equation. Right? I mean, that sounds crazy. Like, how does that work?

Thomas Ahle 00:35:12

Yeah. I mean, this is 1 of the things that really that got me to Normal in the 1st place. Before Normal I was at Facebook as we call them, in the research group that does probabilistic computing. Like we were doing like these Bayesian neural networks where you assume like probability distributions or all your weights and you try and infer like the posterior from like the prior data that you look at. But a lot of these techniques were kind of slow because you had to do, you had to like either try and do lots and lots of repetitions with different random seeds or you were trying to do it analytically. I mean that kind of maybe takes a little bit of the point out of the probability. But then you could also just, but then it's funny because you have these chips and like the chip manufacturers, they spend so much time like getting out every single little piece of noise out of their systems and like having these extremely sharp margins for everything like so much you know, precision is probably like the most precise business in the world. And then what do we do with them? We just like add randomness everywhere. And yeah, so why not try and build a chip that's just inherently random. Like I mean, yeah, I think the brain probably has bunch of randomness. But here the 1st chip we made was then this, you basically, you have this array of capacitors and you have certain resistances between them you can program and then you infuse all of this noise and it will start to behave according to these stochastic differential equations. And then you think, okay, what can we do with that? It's sort of, it's a new computational paradigm that we've found very interesting to explore where you can say 1 of the things, for example, you could do with it is it actually turns out that the matrix that you put onto the chip in the weights differential or the stochastic differential equation. It actually behaves sort of according to the inverse of that matrix. Yeah. And so then we could try and capture and average

it out.[2021222324]

Bayesian uncertainty in the age of token streams

Tim Scarfe 00:37:32

And when we spoke about this on the phone, said something very interesting. Because, you know, we often talk a good game about this. Was talking with Michael Jordan the other day and, you know, like, yeah, we need uncertainty quantification, we need adaptive computation and Yeah.[25]

Thomas Ahle 00:37:45

And I think this was kind of 1 of the issues we had, like, because I think Bayesian machine learning was really strong for a certain amount of time when, like at a certain point in time before the generative AI. Because you had like the 1 output and then it made sense to have this sort of distribution as the output. But now you have these sequences here, as you're saying, like you keep putting these tokens out. And I think no 1 really cares about what is the uncertainty about like 1 particular token and having a better distribution on that. You really want to know after the model have thought about 10 different options and you know it's backtracked and all this stuff and it comes out with a final answer, what is like how much can I trust this answer? You either have to go like really deep on mechanistic interpretability to try and really carry all of the uncertainty all the way through that, or you have to try and use some more maybe anthropomorphic methods, I think, that are based on like how how would humans estimate their uncertainty and you know, and try and apply that at really high level.

Tim Scarfe 00:38:48

But,

[20] Thermodynamic Computing: From Zero to One — other Background reading for the thermodynamic-computing paradigm Thomas describes.

[21] What is the difference between Extropic, Normal Computing, and D-Wave? — other Context for where Normal sits among probabilistic and thermodynamic hardware efforts.

[22] Thermodynamic AI and the Fluctuation Frontier — paper Foundational paper for using fluctuations and noise as the basis of thermodynamic AI.

[23] Thermodynamic Computing System for AI Applications — paper Describes the capacitor-array system that settles into a stochastic differential equation, as Thomas explains.

[24] Thermodynamic Linear Algebra — paper Backs the claim that the chip behaves according to the inverse of the programmed matrix.

[25] Michael Jordan — person Tim cites a recent conversation with Michael Jordan on the need for uncertainty quantification and adaptive computation.

Thomas Ahle 00:38:49

you lose some of that beautiful math in there.

Tim Scarfe 00:38:52

Because it's actually really cool that you've a Bayesian background because the way I introspect about this is, know, like, when you have a thought and you have some intuitive notion of how confident you are, And it seems to be because you have a deep structure. So you can introspect and you can rationalize and you can say, okay. Well, there there's this component and this component, and then there are these constraints. I'm not quite sure about that bit. And that seems to be the missing link.

Thomas Ahle 00:39:16

Yeah. Yeah. We actually did some experiments back also when transformers 1st became popular where we tried to do it's before we knew exactly what the company was gonna do, that it was gonna be AI for hardware. So we wanted to do like predictions. And we just sort of building Bayes' Law into the model. So for, for some particular question, it would try and find like lots of pieces of evidence and then it would say what is the probability that this, if the answer is yes and if the answer is like different things, like what is the probability that I would see this evidence? And we do that lots of times and then at the end, you know, you could use Bayes' Law to say, okay, then what is the probability that the original statement was true or false? It actually worked really well like we did this I did some like internal like prediction game and it beat everyone. We had 2 multiple versions. 1 of the versions we sort of it was more a neurosymbolic, you could say, where you would just like have the model come up with all of these probabilities and then you would manually calculate it. And then I also tried another version where it I just told the model now use Bayes' Law and do it or something. And somehow it actually did a little bit better. So I don't know how it did that or maybe it just was able to yes, maybe it's because it was able to look back and think actually a couple of these values, probably bullshitted and I should I'll just ignore that or but, yeah, it's interesting. I know there are some people training, like there are some benchmarks now for where people are trying to predict polymarket and this kind of stuff with LLMs and they have benchmarks for that. But I'm not quite sure if they use these techniques or it's all just end to end reinforcement learning and and they hope it just picks up a good mythology by itself.

Hybrid compute: vibe-coding loops, binaries and Stockfish

Tim Scarfe 00:41:12

But it's a wonderful example of this kind of adaptivity specialization loop. Yeah. So, you know, for the 1st and this is common with vibe coding. So for the 1st month, there was a false positive every single day. Yeah. And, you know, so I I just created a skill and then Claude would just retrain it and then eventually it just kind of converges. And and this is this is the the bull case of vibe coding. Right? But, you know, like, just every single day, you fix it, fix it, fix it, fix it, and eventually, you'll you'll come into the landing track, and and it works really wrong. But, in a way, this is similar to what you guys are doing. Right? Because you you you have this kind of outer loop, which takes something complex and then you kind of compress it and you optimize it and then you bake it into a kind of crystallized hardware.

Thomas Ahle 00:41:53

Yeah. I I was thinking about there is this older tweet by Elon Musk. I don't know if people were laughing at him. I mean, I don't know. He tweets a lot. But he was saying like, why don't LLMs just write the binaries directly or the assembly?

Tim Scarfe 00:42:09

Oh, yeah. Saw that. I

Thomas Ahle 00:42:11

don't see why we wouldn't be able to it. It's a question of if we would want to do it or like do we think because I think there's some fundamental computational problems always and everywhere like in the world we don't always like recent years we don't focus so much on them because we're so excited by AI and we wanna we're just interested in all the stuff that AI can do really well. But I mean obviously things like cryptography or like some of these like basic algorithms like you never want to want the LLM to just do it even if they can do very large number multiplications. It's just super inefficient. Or like you might as well optimize like a circuit a piece of code for that. Compilation has some of the same problems where you want to and also and chip synthesis where you're trying to explore all these different designs and so on. So you don't necessarily want to have the LLM do it because it's so slow versus some super optimized loop. Yeah, not to jump too much on things, but it's interesting to think about chess again. Of course you have like the AlphaGo thing where it was like very everything ran through a neural network, right? But today the state of the art is in like Stockfish, of course, that they took, they did more like hybrid. So they took the neural networks and they made these like a certain type of neural network that can update really fast when you change the state. And then they combine it with like just super fast search and it actually outperforms the best like open source like AlphaGo type chess engines.

Tim Scarfe 00:43:51

Oh, really? Using like adaptive fine tuning? And structured inference? I don't know if you call

Thomas Ahle 00:43:59

it that. It's more like they took the classic chess search engine and they replaced the evaluation function with a neural net, but like kind of a very shallow wide neural net that they can, that is really, really fast to evaluate. And yeah, think it could be that for something like synthesis and compilation, there's a similar thing where it's like you could try and do it all with LLMs, but at some point that's the speed is a bottleneck and like you can throw, get a benefit from having more knowledge and more intuition on all of this stuff. But at some point there's also just a hard commutation problem where you just want to brute force some stuff. And at that point, want to be able to switch to more classical algorithm.

Tim Scarfe 00:44:44

Absolutely.

Co-design, central-AI apps and API pricing

Tim Scarfe 00:44:45

Now before, we were talking about thermo computing. So you folks have done some work that is incredibly, like, it has huge potential for things like, know, Markov chain Monte Carlo, think, and, diffusion models. But you did say to me when we spoke last time that, in some cases it can be a false economy. Like, for example, you could have a diffusion model which might have a different type of neural network on the end of it. And and you might find that the benefit that you have doing the diffusion might be bottlenecked by another part of the model. So so in in in practice, where can we see a huge uplift?[26][27]

Thomas Ahle 00:45:18

I think it's it's always interesting when you build hardware because you're there's always this co design problem with the algorithms, right? The algorithms are so based around the GPUs and the hardware we have now. And so you can try and target some of those bottlenecks like you can make hardware like we talked about with the thermal DRAM that's super efficient for the memory. Like that might speed up inference but it can only speed it up like say we have, there's some like 10% GPU utilization now for inference, then you can hope to have like 10x there. But you don't know if you suddenly have access to that much like that different architecture, what new algorithms could you build if you really went all in that. And there's some similar thing with the diffusion stuff, right? It's like now the architecture is not that efficient in terms of randomness like the GPUs. It's kind of people don't really do want to sample like tons of Gaussian random variables everywhere. And so they maybe also build architectures where they sort of try to remove that bottleneck by not focusing too much on these things and then instead, so we can build new hardware that makes them that more efficient and that has some performance gain. To really make the most of it you then also have to come up with the new models that sort of go all in on that. It's interesting with this Notion and apps. Have you thought about this? How all of the apps now they want to be like your central AI? Like Notion has this. They want to integrate with the other apps and your linear have an AI and they want to integrate with the other apps. Like, everyone is trying to, like, capture, like, being your central AI assistant, and then all the other apps will just be tool calls.

[26] An efficient probabilistic hardware architecture for diffusion-like models — paper Relevant to the discussion of thermo hardware for diffusion models and where the real uplift versus a false economy lies.

[27] Stockfish — tool Thomas's example of a hybrid system: a fast shallow neural eval plus brute-force search beating pure-neural engines.

Tim Scarfe 00:47:11

I know. I have to be careful because Notion are now sponsoring some ads. But no. They I actually use it a lot because it's got an amazing agentic interface. It's got like a CLI interface. So I I use it from, you know, from Claude. But what they want you to do is to pay them to use the agents that are built in at API prices.

Thomas Ahle 00:47:28

Yeah.

Tim Scarfe 00:47:29

And I don't really see the reason in doing that when I've just got a Notion MCP, I've got a Notion CLI.

Thomas Ahle 00:47:34

API prices stuff is really killing innovation, I think, in that space, right? I mean, think Codex, I think they are actually opening up more that you can use your accounts. I saw just yesterday, I think maybe also Anthropic opened up some that you could use some more, just like the SDK you can use with your accounts. But it's just like the price difference is so big that if you have to use API pricing, like nothing is competitive.

Tim Scarfe 00:48:00

Well, I mean, you know, we don't have to go too deep into this, but I think creativity is all about respecting constraints. And, I mean, you studied linguistics at Oxford, and Chomsky always says that there's a difference between linguistic competence and linguistic performance. So, you know, he's using this bulldozer analogy for ages, you know, he said that I love bulldozers too, they're great for clearing the snow, they're not a contribution to science. And he even said about Deep Blue that a computer winning at Deep Blue in the way it did is a little bit like a bulldozer winning the weightlifting, you know, weight lifting competition at the Olympics.

Thomas Ahle 00:48:33

I think it was in a way, like with Deep Blue. I mean, now maybe modern chess AI is a bit more is a bit different. I actually worked on chess engines for 15 years. But I I remember him saying that too and I was like a bit surprised because you'd think that if like, sure, if your science is biology or you want to understand how humans do language, sure, like then maybe the LLMs is kind of not so relevant how they learn language. But if your science is more the abstract of what is language, I I thought you would be really interested in seeing like different systems developing language and you can comparing and see what's common, what's the same, what's different and so on. That would give you like a a wider understanding of what is the concept of this thing. I mean of course it has to you have to have some kind of respect for AI language to even like include it if you just think it's like completely just some, I don't know, stochastic parrot thing. Maybe you're like I don't really care about it. Don't want to include it in my model of language. But if you do think it's actually doing language, then I think it doesn't really matter if it does the same way as humans or not. Maybe it's more interesting if it's in a different way.

Chain of thoughtlessness and the Chomsky hierarchy

Tim Scarfe 00:49:45

I know. I know. You you mentioned chain of thought, by the way. How much can we read into chain of thought? Right? Because you because, you know, like, some people just call it chain of thoughtlessness, like, you know, and, but actually, it is probably the the modus operandi now for doing interpretability. Right? For actually understanding what they're what they're thinking. Yeah. But, you know, you you could argue that chain of thought is like the press secretary, not the orchestrator. So it's it's almost a post hoc confabulation. And but that does that's not quite right, is it? Because that sounds a little bit like there's no cause or link between the chain of thought and what the language model outputs. That's not true. So how much can we read into it?[28]

Thomas Ahle 00:50:22

Yeah. It's like you want to give the model somewhere to think, right? Like are you when you do reinforcement learning, I mean the chain of thought before the reinforcement learning and after reinforcement learning I think is very different because before it was kind of you try and prompt it and some tricks they kind

[28] Chain of Thoughtlessness? An Analysis of CoT in Planning — paper Anchors the chain-of-thought-skepticism thread when Tim raises how much we can read into chain of thought.

of work and kind of doesn't work. But once you do the reinforcement learning you need like, it's like a Turing machine, right? Like it needs to have infinite memory and being able to have something to operate on. Whereas when you just had the transformer and just like single shot output, it's like, I mean probably in the Chomsky hierarchy it would be a completely different type of system like, I don't know, an automaton or something, right? There's like only finite amount of computation it can do, but now it can do as much computation as it wants. And it just has to learn to figure out how to do it. Like you could imagine building a really simple LLM type system and with access to chain of thought it would be a universal Turing machine. So now at that point it's just now because it can have this tape and it can keep reading and putting. I think I think that would be pretty easy to make a structure like that. But then the question is whether it can learn it. But at least now it has the like representation capacity so it can do it and clearly something is improving and working.

Tim Scarfe 00:51:35

This is actually the bigger problem, like, with the ecosystem, which is that software was always designed to be a thing that that reduces complexity and introduces canalization. So spreadsheets, they're they're a great example of this. Now, you know, accountants use spreadsheets and financial and, you know, like, everyone's using spreadsheets, and it creates an interface that everyone uses, and it reduces complexity in the system. This this agentic AI, it just creates spaghetti everywhere. And this is part of the reason why people aren't shipping because, you know, it creates ephemeral, you know, complex it it's a little bit like bash scripting on steroids. So I've now created this web that only I understand and it's becoming more and more specialized over time, I can't share it with other people. And and the entire ecosystem is becoming very messy.

Thomas Ahle 00:52:18

Yeah. No. I I definitely think a lot of things are up breaking now because of that. I mean, a lot of the open source stuff is is like everyone just also like writes new code from scratch instead of trying to come together and hone these like shared libraries. But I mean it is just for a lot of things more efficient than also going through and you can get everything just the way you want it. But yeah, and I actually think it's very interesting what you're talking about with the spreadsheets and like this channeling, right? Because there's a lot of kind of domain knowledge inside of these tools. And that's actually also what I think people might be worried about who build these tools, how you see it like it is to kind of extract that knowledge and like people building clones and so on of their tools. Like if you have like everyone these days are like locking down all of the data, right? Because they're realizing data is becoming so valuable. They want to keep it themselves or build something themselves rather than have other people use it. I think it could be a similar thing with a lot of like specialized tools because the tools basically are data like they're developed over so much time like finding all the right patterns, everything is built in there. So how do you avoid other people from going in and sort of, yeah, extracting, distilling it, kind of.

AI psychosis, slop and the broken social contract

Tim Scarfe 00:53:40

I'm sure this has happened to you. Random people on the internet will say, I've just generated this paper, I've just generated this code, have a look at it. And, you know, there's massive amounts of AI psychosis out there, which is that, like, you do things that are slightly outside your domain of expertise and Claude will convince you that your stuff is not mediocre, it's great, and then you share it with other people and other people immediately see through it. And this is a bit of a serious problem as well, in my opinion, because I think there are experts out there who have really, really clear ideas about things. They've been doing software engineering and, you know, they've been in science. And when they use Claude, it's brilliant if if they're diligent. Right? Because they can actually use good abstractions and representations. But there's now a tsunami of pollution out there.

Thomas Ahle 00:54:21

Yeah. I don't Yeah. And it breaks the social contract, right? Like in the past if if I wrote something and ask you to read it, you could at least have assumed that I would have spent 10 times more times writing it than you reading it. But now

you're really skeptical about anything because like why would I want to spend time reading some stuff that you didn't even read yourself maybe. Yeah, like I mean, but a lot of these are like social problems, right? Like how do we protect also the open source ecosystem from AI generated PRs where like no 1 has ever read them and then these like volunteer maintainers have to like read through all the slop. Yeah. And yeah, it's it's not really fun. Yeah. Do you have you talked with anyone about that? Like people talking about like we need like a GitHub, like a social credit system, like a karma system so so people can, like, down rate you. And, like, if if you don't have high enough karma, maybe they'll just not read your PRs or

Tim Scarfe 00:55:23

We do need that. Yeah. Yeah. And arXiv recently put gates on people uploading things there as well.

Thomas Ahle 00:55:30

Right. Yeah. And they had this ban. It's like a 1 year ban if if you have a hallucinated citation. Yeah. But it's yeah. It's sad in a way because it makes it harder especially for for new people and like young people to to break in to that stuff like because then they they don't have any any karma or anything to show.

Tim Scarfe 00:55:48

I I know. But I think the the the the broad reason we have this problem is that this technology is the most deceptive thing ever created in human history. Right? And it's a serious problem, because it's all about epistemic subjectivity, which is that you generate things that you don't understand, and it convinces you that it's correct, and you can't see the glitches. And obviously, an expert can look at it and see the glitches straight away, but and it creates dependency. You know, when you start posting stuff that you that you don't, you know, you don't understand, you want to be consistent. Right? You you you've now made a statement that I know about this thing, I'm gonna keep doing it. And then weirdly, people are very defensive about it as well, because if you criticize work that they've done with Claude, they take it personally. So it just creates this perpetuating cycle.

Thomas Ahle 00:56:34

It creates a feeling of understanding that you might not actually have, right? And it's like, I don't know, doing an exam or copying somebody else's work, like you might feel like you wrote it but you don't, your brain didn't go through the motions. And yeah, I think it's dangerous also for like if you're worried about sort of AI taking over, you know, because it's not like it's not just that it's getting smarter, it's also that humans are getting dumber, like we no longer like know and like we get get lazy in terms of understanding stuff. We don't read the papers, you just put them in AI and be like, oh explain this paper to me or something. But yeah, it's it's strange right because on the other hand, I mean it can be like this big level, like speed, like it can speed you up so much so it's it's to not use it. But like finding out when you should stop and when you should start using it again.

Typing it yourself, teamwork and performance vs competence

Tim Scarfe 00:57:34

I know, I mean it's reminiscent of the Elon Musk tweet, you know, he said there are no researchers at Tesla, there's only engineers. And this for me is big problem actually, it's a paradox because you can use language models to increase your knowledge. That's a fact, right? If the purpose is to increase your knowledge, then, you know, if you're a curious person, you can just dig and dig and dig and you can learn a hell of a lot. So why is it the case that on average they erode our knowledge? And I think it's Elon's fault, not his fault, but, you know, he says that engineering is like, you know, this is a means to an end. So we are building this thing and it needs to pass these tests, and in a sense, I don't care if your knowledge erodes during the process because that's not what I'm measuring you on. And, you know, we shouldn't be binary about it because clearly engineers, they trip up on things and they learn a lot along the way, but it seems to be quite convergent, know, like, when when you're pursuing knowledge for its own purpose, you seem to you seem to build deeper foundations and and discover new things.

Thomas Ahle 00:58:34

Yeah. And, I guess that's like a tension with like, in capitalism or something, know, the companies they're not trying to necessarily develop their employees. I mean maybe if they can see a profit motive in it but at the end of the day they just want to get the job done. Yes. And like, of course, if you spend a lot of energy developing your employees and then they leave and stuff, it's always a bit tough

there. I don't know if there's a way to force it. But I think it's also a little bit self inflicted because I mean learning stuff is just hard and like it can be like frustrating and it doesn't always get you in the same kind of flow as just prompting or something. Right? And it's like I think yeah. I I've been trying to use this. I think it was Karpathy who who who suggested this, like when he was learning some new stuff from LLMs. He didn't like copy paste it. He would have like side by side windows. But he would always like write all of the code by hand. I made an internal app to try and because we hire a lot of AI engineers. Like it's hard to find people who are both like really good hardware engineers, really good AI engineers and also software engineers, of course, you have to be that. So we have to scale up people. We have to scale up hardware people on AI. We have to scale up AI people on hardware. So I made like internal like training tools and so on where I try to have the same like blocks in there if you trying block copy and paste in the least like give people a warning and saying like you should maybe try and type it yourself. Somehow it like, you know, you wouldn't think so. Like it doesn't feel like anything. Like if I'm just looking at it and typing myself, like why would I learn it better? But it's just empirical that somehow like our neurons and stuff like it's important like to go through the movements. You have to figure out what projects you need to understand in-depth and which ones you're just making like, wait, wait, it doesn't matter as much. I think we we get tempted with the AI to just start way more projects in parallel, right? Because like oh I can just start another 1 and while it's working that I'll start another 1 and another 1. And then suddenly you're like well there's obviously no way I could actually understand all of these in-depth because now I started so many projects. So then you have to tempt stop yourself and being like, maybe instead of starting the 5th project, I'll go back and try and understand what's actually going on in in the in the 1st couple of projects. I think maybe there'll be some evolution. Like, we're still learning all the social stuff around using these tools and and what is effective and what's not effective hopefully will. And maybe we can even build some stuff into the tool system. Is that what you're saying also with the scoping like to to try and make them better? And it's hard. I mean another really hard social problem I think right now is how do you do teamwork? Like that's something I also think about as a manager like if everyone on my team is like staring at their 10 agents or something working on some stuff. When are they gonna go and talk to their colleagues about this stuff? Or like if they don't understand the code themselves, how are they going to explain to the other people in the company what the code does and sort of and build together? And I don't think anyone has really solved it well and like at scale. I heard some podcasts also with the Anthropic co working team. And least the way I remember it was that the podcast asked, Hey, how did you make this so good? And they were like, Yeah, just everyone on the team coded their own version and then we picked the best 1. And I was like, Damn, If that's if that's the best we can do in teamwork, you know. It's kind of I mean, I

in a way it's it's cool because it's like it's like the ensemble method just applied to people and and, you know, it's it's cool you can explore so much more. But it it's also like it does remove all of the teamwork. At that point, it's just everyone working in isolation. You're just duplicating. Yeah. So I'm I'm curious how we're gonna solve that.

Tim Scarfe 01:02:42

You know, it's performance versus competence. And I guess we're kind of making a bet collectively that performance is all that matters. Thomas, it's been amazing having you on the show.

Thomas Ahle 01:02:53

Yeah. Been so much fun. Thank you, Tim.

References

[1] **Thomas Ahle**
https://thomasahle.com
person Guest; theoretical computer scientist who moved from high-dimensional algorithms to thermodynamic computing at Normal.

[2] **Normal Computing**
https://normalcomputing.com/
organization Thomas's company, building AI for chip design from intent to tape-out and developing thermodynamic hardware.

[3] **Subsets and Supermajorities: Optimal Hashing-based Set Similarity Search**
https://arxiv.org/abs/1904.04045
paper Representative of Thomas's earlier work on locality-sensitive hashing for high-dimensional data.

[4] **Clustering the Sketch: Dynamic Compression for Embedding Tables**
https://arxiv.org/abs/2210.05974
paper Another strand of Thomas's pre-Normal research on high-dimensional data and compression.

[5] **Lovable**
https://lovable.dev/
tool Thomas frames Normal as the Lovable for chip design, taking you from intent to a finished design.

[6] **Building an Open-Source Verilog Simulator with AI: 580K Lines in 43 Days**
https://normalcomputing.com/blog/building-an-open-source-verilog-simulator-with-a
other Thomas's blog on generating a Verilog simulator with a swarm of agents, the launching point of the conversation.

[7] **Normal Computing Announces Tape-Out of the World's First Thermodynamic Computing Chip (CN101)**
https://www.normalcomputing.com/blog/normal-computing-announces-tape-out-of-world
other Tim's narration introduces CN101, Normal's first thermodynamic chip taken to silicon.

[8] **World's first thermodynamic computing chip reaches tape out**
https://www.tomshardware.com/tech-industry/semiconductors/worlds-first-thermodyna
other Press coverage of the CN101 tape-out referenced in the chip's introduction.

[9] **Building a C compiler with a team of parallel Claudes**
https://www.anthropic.com/engineering/building-c-compiler
other Tim's parallel: Anthropic reproducing a C compiler from a functional spec with thousands of agents, the same recipe Thomas used for an EDA tool.

[10] **ProgramBench: Can Language Models Rebuild Programs From Scratch?**
https://arxiv.org/abs/2605.03546
paper The benchmark where models re-implement programs like FFmpeg with no internet and initially scored near zero, central to the benchmark-skepticism point.

[11] **Noam Brown**
https://x.com/polynoamial
person Thomas references trying to get an answer from Noam Brown on how RL gains depend on the strength of the pre-trained model.

[12] **Anthropic**
https://www.anthropic.com/
organization Discussed for treating continual learning as a safety risk and for the co-working team's everyone-builds-their-own approach.

[13] **AlphaProof: AI achieves silver-medal standard at the IMO**
https://deepmind.google/blog/ai-solves-imo-problems-at-silver-medal-level/
other The template Thomas compares to chip autoformalization, including the prove-or-disprove training trick.

[14] **Autoformalization with Large Language Models**
https://arxiv.org/abs/2205.12615
paper Underpins the autoformalization idea of turning human specifications into formal statements in Lean.

[15] **Gemini Deep Think achieves gold at the IMO**
https://deepmind.google/blog/advanced-version-of-gemini-with-deep-think-officiall
other Tim notes the gold-medal model did no formal verification, raising whether verification is a spectrum.

[16] **Math Does Not Represent**
https://www.youtube.com/watch?v=aA_T20HAzyY
other Tim invokes Erik Curiel on multiple orthogonal representations of general relativity to make the no-single-true-representation point.

[17] **Autoformalizing Memory Device Specifications with Agents**
https://arxiv.org/abs/2605.00058
paper Thomas's own paper on the DRAM/memory-spec work Tim references as the article he published earlier.

[18] **DRAMBench: Autoformalizing DRAM Specifications with Timed Petri Nets**
https://www.iese.fraunhofer.de/blog/drambench-autoformalizing-dram-specifications
other Supports the surviving DRAM and timed-Petri-nets discussion of protocol-level memory properties.

[19] **TLA+**

https://lamport.azurewebsites.net/tla/tla.html
tool Thomas names TLA+ as the classic language for higher-level protocol properties like deadlock and livelock.

[20] **Thermodynamic Computing: From Zero to One**
https://extropic.ai/writing/thermodynamic-computing-from-zero-to-one
other Background reading for the thermodynamic-computing paradigm Thomas describes.

[21] **What is the difference between Extropic, Normal Computing, and D-Wave?**
https://www.zach.be/p/whats-the-difference-between-extropic
other Context for where Normal sits among probabilistic and thermodynamic hardware efforts.

[22] **Thermodynamic AI and the Fluctuation Frontier**
https://arxiv.org/abs/2302.06584
paper Foundational paper for using fluctuations and noise as the basis of thermodynamic AI.

[23] **Thermodynamic Computing System for AI Applications**
https://arxiv.org/abs/2312.04836
paper Describes the capacitor-array system that settles into a stochastic differential equation, as Thomas explains.

[24] **Thermodynamic Linear Algebra**
https://arxiv.org/abs/2308.05660
paper Backs the claim that the chip behaves according to the inverse of the programmed matrix.

[25] **Michael Jordan**
https://people.eecs.berkeley.edu/~jordan/
person Tim cites a recent conversation with Michael Jordan on the need for uncertainty quantification and adaptive computation.

[26] **An efficient probabilistic hardware architecture for diffusion-like models**
https://arxiv.org/abs/2510.23972
paper Relevant to the discussion of thermo hardware for diffusion models and where the real uplift versus a false economy lies.

[27] **Stockfish**
https://en.wikipedia.org/wiki/Stockfish_(chess)
tool Thomas's example of a hybrid system: a fast shallow neural eval plus brute-force search beating pure-neural engines.

[28] **Chain of Thoughtlessness? An Analysis of CoT in Planning**
https://arxiv.org/abs/2405.04776
paper Anchors the chain-of-thought-skepticism thread when Tim raises how much

we can read into chain of thought.