

THE MACHINE LEARNING COLUMN

BY

PABLO BARCELÓ

Pontificia Universidad Católica de Chile
Avda. Vicuña Mackenna 4860
Santiago, Chile
pbarcelo@uc.cl

AND

DAVID SAULPIC

Institut de Recherche en Informatique Fondamentale (IRIF)
CNRS & Université Paris Cité
8 Place Aurélie Nemour
75013 Paris, France
david.saulpic@irif.fr

THE COMPUTATIONAL POWER OF TRANSFORMERS: INTERVIEW WITH ANTHONY WIDJAJA LIN

Foreword

For this edition of the Bulletin, Pablo and I interviewed Anthony Widjaja Lin back in June. The TCS world changed quite dramatically since then, and it would seem odd to publish a column speaking about TCS and ML without any mention of it.

First revolution, Pablo was appointed Director of the Chilean National Center for AI, and is therefore stepping down from the bulletin – I wish him luck in his new role, and a thousand thanks for the work we have done together.

Besides this new appointment, the theoretical research community is in crisis. The new AI models make theorem proving easier than ever, and the consequences in our community are endless: the number of papers written and submitted increases exponentially (even faster than it used to) and famous open problems are closed every day: the research time seems to contract. This in turns affects our practice. It is harder than ever to keep up-to-date with the progress made on our favorite areas, to publish papers that respects our former scientificness criterion, and to prove theorems the old way – a reason why we enjoyed our job. And those changes raise fears: of being scooped by AI-powered competitors or even by AI agents scrolling the web for open problems to be solved¹, which encourages to keep our research private; of the prospects of a mathematical practice we may not like; of our research becoming dependent on big tech companies; of the impact of this, on nature and society; of being simply not as good as AI, hence useless; or, of governments being convinced of the later, which would be a good reason for cutting theory jobs.

The mathematical community is facing similar questions, and is reacting by communicating a lot online. Members of this community write text ², participate in blogs, engage in discussion on social media such as mathstodon. A particular

¹see <https://yangpliu.github.io/repository.html>

²see <https://mathandai.org/>, which collected thousands of endorsements, or <https://webusers.imj-prg.fr/~isabelle.gallagher/AIg.pdf> (in French)

website was started, collecting and publishing opinions every day (Proofs and prompts); and a scientific organization was created, the Association for Human Mathematics.

On the other hand, the TCS community seems more silent. I am merely aware of Oded Goldreich's post³, that of Pravesh Kothari, Raghu Meka, and Prasad Raghavendra⁴, and of C. Seshadhri⁵. Nonetheless, we are affected, and there is a concerning number of people dropping theory studies, facing existential concerns, depression or anxiety – particularly people with a non-permanent job, but not only.

I am therefore calling for more discussions, more shared opinions, more debate on AI, our use of it, our vision of the job, and why humans are important (or not). We need, at least, venues for collective discussions, and to take our time thinking what's coming.

The EATCS bulletin is not the place for this, because of its periodicity. If nothing better appears in the meantime, I propose to list in the next edition all the contribution that I will be aware of – please, let me know if you read or write something! – but I would love to see nicer initiatives.

I am looking forward to reading your essays, ideas, impressions, or feelings, and I hope you will enjoy Anthony's interview.

David Saulpic

PS: AI is currently the main crisis in our small community, but the world is facing several others that will end up impacting us as well, such as wars or the climate crisis. We should also spend time and energy on anticipating, and in not accelerating them: for this too we need discussions.

³<https://www.wisdom.weizmann.ac.il/~oded/on-proofs-t2.html>

⁴<https://scottaaronson.blog/?p=10116>

⁵<https://users.soe.ucsc.edu/~sesh/essays/ai-and-math.html>

1 Introduction

Few technologies have reshaped computing as quickly as the transformer. Introduced in 2017 [6], it is the architecture beneath today’s large language models, and with them the wave of systems now writing code, drafting text, and answering questions across the sciences. Yet for all this success, a basic question remains stubbornly open: what can these models actually do, and what lies beyond their reach? This is precisely where theoretical computer science enters. A transformer, stripped to its essentials, is a machine that reads strings and makes decisions—an object of exactly the kind that logicians and automata theorists have studied for decades. By asking which formal languages a transformer can recognize, and matching them against classical yardsticks like circuit complexity and first-order logic, researchers have begun to draw a rigorous map of the model’s powers and its blind spots. The results are often surprising: a network that can write a sonnet may be provably unable to count whether a letter appears an even number of times. Beyond intellectual curiosity, the stakes are practical, touching trust, verification, and our hope of one day guaranteeing how these systems behave.

Few people are better placed to guide us through this territory than Anthony Widjaja Lin. Based at the University of Kaiserslautern-Landau and the Max-Planck Institute for Software Systems, Lin built his career in the classical heartland of theoretical computer science—logic, automata, automated reasoning, and the verification of software, where he is known for work on string constraint solvers and model checking. In recent years he has turned that arsenal toward machine learning, becoming one of the architects of the emerging theory of transformer expressivity: he has helped show which logics transformers can capture [1], uncovered a startling link between their counting ability and the undecidability of Diophantine equations [4], proved that they describe concepts with extraordinary compactness (at the cost of making them notoriously hard to verify) [2], and most recently established that chain-of-thought transformers with soft attention are Turing-complete [3]. Equally at home with a deep theorem and with what it means for the systems we actually deploy, he is an ideal voice for a conversation at the crossroads of theory and modern AI.

2 Technical Background

The interview turns on a single question: *what can transformers provably compute, and what lies beyond them?* The fruitful way to make this precise is through formal language theory. A transformer reads an input string and either accepts or rejects it; the set of accepted strings is its *language*. By comparing these languages against well-understood yardsticks—regular languages, circuit-complexity classes,

fragments of logic—one can pin down both the powers and the limits of the architecture. This note fixes the minimum vocabulary needed to follow that discussion.

Strings, languages, and counting Let Σ be a finite *alphabet* of tokens. A *string* is a finite sequence $w = a_1 \cdots a_n$ with $a_i \in \Sigma$, and a *language* is a set $L \subseteq \Sigma^*$ of strings. Two recurring touchstones are the *counting languages*

$$\text{PARITY} = \{w \in \{a, b\}^* : |w|_a \text{ is even}\}, \quad \text{MAJ} = \{w \in \{a, b\}^* : |w|_a \geq |w|_b\},$$

where $|w|_a$ denotes the number of a 's in w . Both are *permutation-invariant*: membership depends only on letter counts, not on order. They are deceptively simple, yet they sharply separate the models below.

Transformer encoders We describe a transformer encoder semantically. A *token embedding* $\text{em} : \Sigma \rightarrow \mathbb{R}^d$ maps each symbol to a vector, extended to strings coordinatewise. The transformer is then a composition of *length-preserving* maps $(\mathbb{R}^l)^* \rightarrow (\mathbb{R}^h)^*$ built from two ingredients: (i) *position-wise* feed-forward functions (compositions of affine maps and $\text{ReLU}(x) = \max\{0, x\}$, applied to each vector independently), and (ii) *attention layers*, which let positions exchange information. To recognize a language one fixes an output vector $\mathbf{t}$ and accepts w iff $\langle \mathbf{t}, \mathbf{v} \rangle > 0$, where $\mathbf{v}$ is the final vector at the last position.

Definition 1 (Attention layer). *Given input vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$ and affine maps A, B, C , the attention score of query i at key j is $s_{ij} = \langle A\mathbf{x}_i, B\mathbf{x}_j \rangle$. A normalizer turns the scores $(s_{ij})_j$ into weights $w_i(j)$, and the layer outputs*

$$\mathbf{y}_i = C(\mathbf{x}_i, \mathbf{v}_i), \quad \mathbf{v}_i = \sum_{j=1}^n w_i(j) \mathbf{x}_j.$$

The choice of normalizer fixes the model:

- **Softmax (SMAT).** $w_i(j) \propto e^{s_{ij}}$. This is the mechanism used in practice, but the exponentials make it hard to analyze.
- **Unique hard attention (UHAT).** All weight is placed on the single *leftmost* position j maximizing s_{ij} .
- **Average hard attention (AHAT).** The weight is spread *uniformly* over all positions j that maximize s_{ij} .

UHAT and AHAT are tractable idealizations of softmax; AHAT, in particular, can average (and hence count), which softmax approximates well.

Positional information Without positional cues, attention treats its input as a bag of tokens and can only recognize permutation-invariant languages. Order is recovered in two ways: a *positional encoding* (PE) $p : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}^d$ added to the embeddings, or *causal masking*, where position i attends only to $j \leq i$. The label *NoPE* marks a transformer with *no* positional encoding. As the interview stresses, this seemingly minor knob is decisive: it is exactly what lets some models recognize PARITY, and a single fixed *relative* PE is enough to push chain-of-thought transformers to Turing-completeness.

Circuit complexity. AC^0 is the class of languages decided by families of constant-depth, polynomial-size Boolean circuits with unbounded-fan-in AND/OR/NOT gates; TC^0 adds *majority* gates, which suffice to encode arithmetic and counting. A landmark fact is that $PARITY \notin AC^0$, whereas $MAJ \in TC^0 \setminus AC^0$. These classes upper-bound the models: UHAT lies (strictly) inside AC^0 , and AHAT inside TC^0 .

Logic on strings. A string is viewed as a structure with a linear order $<$ on positions and a unary predicate Q_a marking the positions labelled a . First-order logic (FO) over this structure defines exactly the *star-free* languages, and—by *Kamp’s theorem*—is equivalent to Linear Temporal Logic (LTL), whose formulas are read as sets of positions. Enriching FO with arbitrary *unary* (“monadic”) numerical predicates gives FO[Mon], a class strictly between FO and AC^0 . Adding to LTL terms that *count* how often a subformula holds to the left or right yields *Counting LTL*, which can express MAJ and other counting properties.

The landscape in one breath UHAT sits strictly inside AC^0 and captures all of FO[Mon] (hence every regular language in AC^0), but cannot recognize PARITY [1]. AHAT reaches into TC^0 and captures Counting LTL, so it *can* recognize PARITY and MAJ—though, strikingly, whether it does so can hinge on the positional encoding. Finally, with *chain-of-thought* (feeding the model’s own outputs back as inputs), transformers become Turing-complete [5]. These are the coordinates against which the rest of the conversation is plotted.

3 The Interview

David Saulpic: Transformers have transformed AI and our field, and they work astonishingly well, without anyone having proven a theorem about them. So, why should we care about theory? What should we care about proving what they can and cannot do?

Anthony Lin: Firstly, I should mention that this is a field I have been getting into only in the last three years, so what I say should not be taken as authoritative. For me, the most important thing is simply the joy of understanding. I already learned this many years ago with neural networks — it is very surprising to see intuitive connections between neural networks and things I had been working on, like logic, automata, and circuit complexity.

To answer your question more directly: how can TCS, logic, and automata contribute to machine learning? My general answer is that TCS provides a solid mathematical foundation where you can prove what models can or cannot do, and by what neural mechanism. This is the role TCS can play. In particular, it could be useful when you want to understand specific phenomena — for instance, why some things can or cannot be trained so easily. You may also want to develop more explainable models that are better understood by people, and once you have accurate, explainable models, you can develop ways to reason about them algorithmically at a post-learning stage.

I would add a caveat. Despite the massive progress over the last seven years or so, there is still a huge chasm between theory and practice, especially at the scale of large language models. We do not have a theory that can reliably predict why LM agents can successfully code some programs and what kinds of problems they would struggle with. Whether what we do can scale up to that level remains an open question. Terence Tao said something interesting: the mathematics behind LLMs is quite simple, but no one really understands why they work. I find this paradox quite interesting.

Pablo Barceló: I think it is not surprising, but it is very interesting to see that LLMs are models of computation. It is then very natural to try to understand what kind of computation they can pursue. Why is that important? Because then we can understand their limitations and capabilities much better. I think we currently understand the limitations somewhat better than the capabilities, and this has been instrumental in trying to design new architectures that are more capable of reasoning.

AL: And of course it is also important to go back to the history: this was already discussed in early work on neural networks as a model of computation, so it is very natural to try to understand what you can and cannot do with this model.

DS: Pablo, you said that this understanding of limitations has already led to new architectures?

PB: From a theoretical point of view, yes. They are not being implemented at the moment, or only in a proof-of-concept way. But yes, we understand what is needed for transformers to be able to reason better or to compose functions. Transformers are very poor at composing functions — they require many steps

for that. People are now trying to understand how they can be enriched so that they can compose functions more effectively, which matters because composing functions is fundamental to reasoning efficiently.

DS: So this is part of an answer to what TCS can bring to practitioners of machine learning. But when you use an LLM, it seems they are already reasoning. We understand the limitations, but it is not clear they manifest in practice. What is the gap between theory and practice? Typically, the PARITY problem is hard for a transformer in theory. Is it hard in practice as well?

AL: I think the first versions of GPT struggled with this, but it no longer happens. What I understand is that when you query a modern LLM on parity, it will write a Python program that does the calculation, using a kind of chain of thought to overcome the theoretical limitation. I would still think that if you somehow bypass this — for example by asking to count the number of balls in an image — it might actually struggle.

DS: For the record, the last time I tried, three months ago, LLMs answered correctly the number of R's in *strawberry*, but not in *raspberry*.

AL: This has to do with tokenization: the sentence is split into smaller components, so when you try to count the R's, some of them simply disappear.

PB: In a way, it is surprising that the vanilla version of transformers cannot really count, and that you need to add something. The vanilla version is surprisingly weak, but when you add some form of reasoning via chain of thought, you regain this ability — at high computational cost.

AL: There are quite a lot of things at play. Some of the challenge during inference also comes from computing the attention matrix, which is $O(n^2)$. For RNNs this is linear, which is part of why people are now talking about bringing RNNs back. But there is also the additional cost of the sequential computation steps in chain of thought.

DS: Going back to parity: it is perhaps surprising that it is a hard problem, given what transformers are doing. Can you explain why parity is so hard, and what is its particular role in the field?

AL: When people try to understand the ability of transformers, they can do so experimentally, but they can also develop theory that explains the phenomenon. On the theory side, a number of different formalizations of transformers have studied what kinds of tasks they can solve. These correspond to concepts that can be expressed in first-order logic, linear temporal logic, and related formalisms. It is a classic result that parity is not in AC^0 — and this is also connected to circuit complexity. I think one of the first papers to establish this connection for transformers was by Michael Hahn. This is basically why parity is hard: it

is connected to the kinds of concepts expressible in first-order logic and linear temporal logic, and we can tap into the classical results from theoretical computer science to infer what transformers can and cannot do.

One of the answers to dealing with the parity problem is chain of thought, which you can think of as the transformer talking to itself. When you supplement a transformer with that power, it can do more.

DS: In your recent work, positional encodings appear again and again as the decisive ingredient: without them there is no parity, and a single relative positional encoding — fixed and task-independent — is enough to reach Turing completeness. What is so special about positions? Why does this apparently minor detail change the model’s power so dramatically?

AL: Let me start with a general overview. You can think of transformers as a kind of parallel machine, but without any order — they process everything in parallel, so they do not know which token comes first. Tokens are treated as a bag, without sequence information. This is why you need some kind of positional information, so that the transformer can distinguish what comes first or before. That is the first answer.

PB: And this was actually noticed as a weakness of transformers in the early practical work. It was practitioners who included positional encodings — it is not a theoretical artifact introduced to study computational power, but something that turned out to be necessary in practice.

AL: Absolutely. There are papers that have looked at what kinds of positional encodings are needed in practice, and there is not just one class. Now, without positional encoding, what can you actually do? We know you cannot do parity. It turns out you can actually capture polynomial constraints even without positional encodings — we managed to prove this even without chain of thought.

Going back to your question about Turing completeness: for a one-letter alphabet — which is of course a very artificial setting — with chain of thought, transformers are already Turing complete without any positional encoding. But if you go beyond a one-letter alphabet, the main problem is that without positional encoding you cannot refer back to the inputs, and then transformers, even with chain of thought, are not Turing complete. The main technique is related to the fact that you need to refer back to inputs in order to perform any reduction. What is nice about our result is that with a single relative positional encoding — one that we can explicitly construct and that is quite simple, even if it is not known to be learnable — the model becomes Turing complete.

DS: One of your results connects the counting power of transformers — without positional encoding — to Diophantine equations, and therefore to undecidability. How do you get from an attention mechanism to something akin to Hilbert’s

tenth problem, and what does it mean that analyzing certain transformers is formally impossible?

AL: I can tell you the history. I have done quite a lot of work on logic and automata related to counting — it is one of the things I really enjoy. And if you have a hammer, everything becomes a nail; that is kind of how we got into this. One of the main questions in counting is whether you can do semi-linear counting, whether you can capture linear equations and linear constraints. We were looking at this question for transformers: what kind of counting can they do?

For the result you mentioned, we were very surprised to find that we can capture not just linear or nonlinear counting, but in fact all polynomials. This essentially allows you to encode Hilbert's tenth problem into the analysis of transformers, making their verification undecidable. The proof itself is not so complicated. The attention mechanism does averaging, so you can talk about frequencies. When you want to count, you cannot count directly, but you can simulate counting using frequencies. Using certain gadgets, we showed that we can simulate multiplication and addition this way, and then encode Hilbert's tenth problem.

PB: You also have to be careful about the type of attention you are using. With what is called unique attention — where you attend only to the position that maximizes the value of interest — the expressive power of transformers is very limited. But when you allow average attention, where you average over all positions that maximize the value, the counting power becomes much more powerful, even without positional encodings.

AL: Yes. But interestingly, even with average attention and no positional encoding, you still cannot do parity. There is a classical result showing that if you allow bidirectional attention, with both forward and backward masking, you can in fact express parity. It is expressible, though I do not know if it is trainable.

DS: That is a good distinction to draw out. Can you explain the difference between expressible and trainable, and how it shows up in your work?

AL: This is indeed a very important point. Expressibility means that there exists some transformer — perhaps one we do not have yet — that can solve a given task exactly. Trainability is about whether, given data and running stochastic gradient descent via backpropagation, we can actually converge to a model that solves the task. Of course, people have been formalizing notions of trainability for a long time — PAC learnability, exact learnability in Dana Angluin's sense, and many other notions. That is the fundamental difference.

DS: Can you explain briefly the transformer model you formalize and study?

PB: The model of transformers we use is an idealized version of what is done in practice — one that is easier to study theoretically — but we are trying to bring

it ever closer to practice. The core idea is that when you analyze a sentence, the meaning of each token depends on all the other words in the sentence and their relative positions. This is unlike an image, where a pixel is largely determined by adjacent pixels. Here the meaning of a word depends on all other words and their relative positions. The question is how you attend to other words in order to understand your own meaning. This is also what brings the quadratic complexity: for each word, you have to attend to all other words, measure their importance, and produce a summary of that attention.

AL: One thing worth remarking is that for a long time, the standard way to process text with neural networks was sequential. The problem was that long-range dependencies were always very difficult to capture using RNNs: processing from left to right, what happens at the end of the string can depend on something far back at the beginning — this is the vanishing gradient problem. What attention overcomes this by explicitly looking at all combinations: you compute the attention matrix, which is $O(n^2)$ rather than $O(n)$, but fully parallelizable.

PB: In a way, it is a brute force solution to the problem.

AL: But a parallelizable one.

DS: You proved that transformers describe certain concepts extraordinarily compactly — doubly exponentially more so than automata. The price of that compactness is that verification is EXPSPACE-complete. Are we facing some trade-off, perhaps unavoidable, between how powerful transformers are and our ability to audit or trust them?

AL: That is right, and I am still working out all the corollaries. One corollary is indeed that verification is difficult. But there are others. For example, some tasks that you want transformers to capture may require a very large amount of data to train. And there is another corollary: some concepts can be expressed more succinctly using transformers than by RNNs — exponentially more succinctly. We are running experiments to validate this theoretical claim, and we are preparing a journal paper on this. I am happy to report that after this result, quite a lot of people were excited and building upon it, including some I would not have expected.

One thing to note is that in practice you deal with finite precision — floating point numbers. This renders everything, to some extent, decidable, since transformers are no longer super-powerful under finite precision. But even then, some things can still be expressed very succinctly, which is one of the corollaries I am still working through.

DS: From a learnability perspective, is that something you have taken into account, or are the results purely about expressive power?

AL: So far only expressibility. We are running experiments to validate this, and I can report back in a few months.

DS: And a final question: one of your most recent results proves that softmax chain-of-thought transformers are Turing complete and trainable, by simulating counter machines rather than Turing machines directly. Looking ahead, where does the limit of softmax attention really lie? Which open problem would you most love to solve, and what can the whole arsenal of verification and string-solver techniques you come from bring to AI?

AL: This is difficult to answer briefly. On the softmax chain-of-thought result: our simulation uses counter machines, and counter machines are not very efficient — you would need exponential time to solve something that Turing machines can solve efficiently. So when you account for complexity, things are still quite unclear.

Beyond that, there are open questions about the cost of computing the attention matrix, which is motivating renewed interest in state-space models — essentially a return to RNNs with modifications, removing nonlinearities to simplify the architecture. These have been shown to scale and to allow training of language models, and people are also asking how they can be combined with transformers.

On the verification side, we have only recently started applying techniques from that world. Even when a problem is undecidable — as in the case where you can encode Hilbert’s tenth problem — in formal verification we do not stop at undecidability. Termination is undecidable, but people still built tools like Terminator. We have been applying SMT-based techniques to analyze transformers, and this has turned out to be quite successful.

The most honest summary is that we have only scratched the surface. The field gives you many questions you would never have thought about otherwise. The classical tools from TCS never match perfectly — you never get an exact characterization, it is never exactly TC^0 or exactly any known class, always something in between. But you use those techniques to guide your understanding, and this is precisely what is exciting about the field. I hope more people from the TCS community will start working on it.

PB: There is also this very interesting and surprising intersection with the linguistics community.

AL: Absolutely — people like John Rawski and David Chiang, who are linguists but also logicians by training. I am now connected with so many different people I would not have thought to reach out to before. The field moves extremely fast, but it is also very collaborative, and there are many different perspectives to draw on.

DS: To conclude: you mentioned the field is moving extremely fast, and yet

you are using tools from the 1970s and 80s — constant-depth circuits, automata, counter machines — and they turn out to be currently the best lens for the most modern AI models. Where are they coming back to life, and what is the community forming around this?

AL: I can only philosophize here. I think we analyze things based on what we know. We want to understand things epistemologically based on the theory we have. But you see that not everything falls neatly into place: it is never exactly TC^0 , never exactly $GAP-NC^1$. You use these techniques to guide and understand the problems, but the classical tools never provide a perfect fit. What they do provide is new notions and exciting open questions that can inspire future people in the TCS community. I feel that not so many people from our area have been working on this yet, but more and more are starting to notice it — and I hope that continues.

References

- [1] Pablo Barceló, Alexander Kozachinskiy, Anthony Widjaja Lin, and Vladimir V. Podolskii. Logical languages accepted by transformer encoders with hard attention. In *ICLR*, 2024.
- [2] Pascal Bergsträßer, Ryan Cotterell, and Anthony W. Lin. Transformers are inherently succinct. *CoRR*, abs/2510.19315, 2025.
- [3] Hongjian Jiang, Michael Hahn, Georg Zetsche, and Anthony Widjaja Lin. Softmax transformers are turing-complete. *CoRR*, abs/2511.20038, 2025.
- [4] Chris Köcher, Alexander Kozachinskiy, Anthony Widjaja Lin, Marco Sälzer, and Georg Zetsche. Nope: The counting power of transformers with no positional encodings. *CoRR*, abs/2505.11199, 2025.
- [5] Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is turing-complete. *J. Mach. Learn. Res.*, 22:75:1–75:35, 2021.
- [6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.