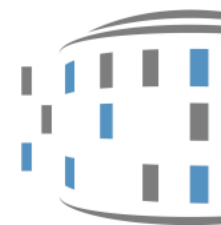


Highlights from attending the Simons Workshop on Transformers as a Computational Model

Breandan Considine



SIMONS
INSTITUTE
for the Theory of Computing

Three overall themes

- Formal language theory
 - Logical expressivity
 - Circuit complexity
 - Algebraic decomposition
- Mechanistic interpretability
 - RASP programs
 - Training dynamics
- Scaling up inference time compute
 - Recurrence and SSMs
 - Search and planning
 - Generator verifier gap



Circuit complexity (Merrill)

- Fixed precision circuits
- Can be studied as logical gates
- Transformer layers = circuit depth
- Computational parallelism vs. sequentiality



Logical expressivity (Chiang)

- Starts with linear logic
- Permits fine-grained control over resources
- Can be used to study transformer model expressivity
- Many fruitful connections with PL theory

Upper bounds

$O(\text{poly}(n))$ -precision transformers $\xrightarrow{[7]}$ $\text{FO}\#[+,x]$
 $\#x[\phi(x)]$
 $\#x[\phi(x)] = y$
 $\not\equiv \text{BFVP}^*$
 $= \text{DLOGTIME-uniform TC}^0$

$O(\log n)$ -precision transformers $\xrightarrow{[6]}$ $\text{FO}\#[+]$
 $\#x[\phi(x)]$
 $\not\equiv \text{PTIME}$

$O(1)$ -precision transformers $\xrightarrow{[5]}$ $\text{FO}\#[+]$
 $\#x[\phi(x)]$
 $\not\equiv \text{PTIME}$

[5] Chiang et al. Tighter bounds on the expressivity of transformer encoders. ICLR 2024.
[6] Merrill and Sabharwal. A logic for expressing log-precision transformers. ICLR 2024.
[7] Chiang. Transformers in DLOGTIME-uniform TC^0 . arXiv:2409.13629, 2024.

Please live-two and
 $\text{strict} \rightarrow (ab)^*$
 $\text{non-strict} \rightarrow (a+b)^*$

Language generation in the limit (Kleinberg)

- Identifying the grammar for a language is not possible (Gold)
- But generating strings from the language is possible!
- Very elegant proof technique

The slide is titled "A Stronger Result for Finite Collections of Languages". It contains the following text:

Suppose K is known to come from a finite collection $\mathcal{C} = \{L_1, L_2, \dots, L_n\}$.

- It follows from Angluin's characterization that there is an algorithm for identification in the limit for \mathcal{C} .
- But we cannot give an a priori bound $t = t(\mathcal{C})$ on how many steps the algorithm will need in order to start being correct:

Below this text are two ovals representing languages:

- Language L_1 : all strings
- Language L_2 : all even-length strings

Below the ovals, the slide states: "But for generation in the limit, we can bound the number of steps required."

Theorem 2 [KM24]: There is an algorithm with the property that for any finite collection of languages \mathcal{C} , there is a number $t(\mathcal{C})$ such that the algorithm will always generate from the true language starting within $\leq t(\mathcal{C})$ strings:

- For any language K in \mathcal{C} , and any sequence S of at least $t(\mathcal{C})$ distinct elements from K , the algorithm given S can produce an infinite sequence of distinct strings from K .

The man standing next to the slide is wearing a black shirt and a name tag.

Learning to reason with LLMs (Brown)

- Starts with his work designing poker bots
- (Re?)discovered the importance of planning
- Generator-verifier gap
- Cue O1, OpenAI's newest LLM with SoTA reasoning capabilities

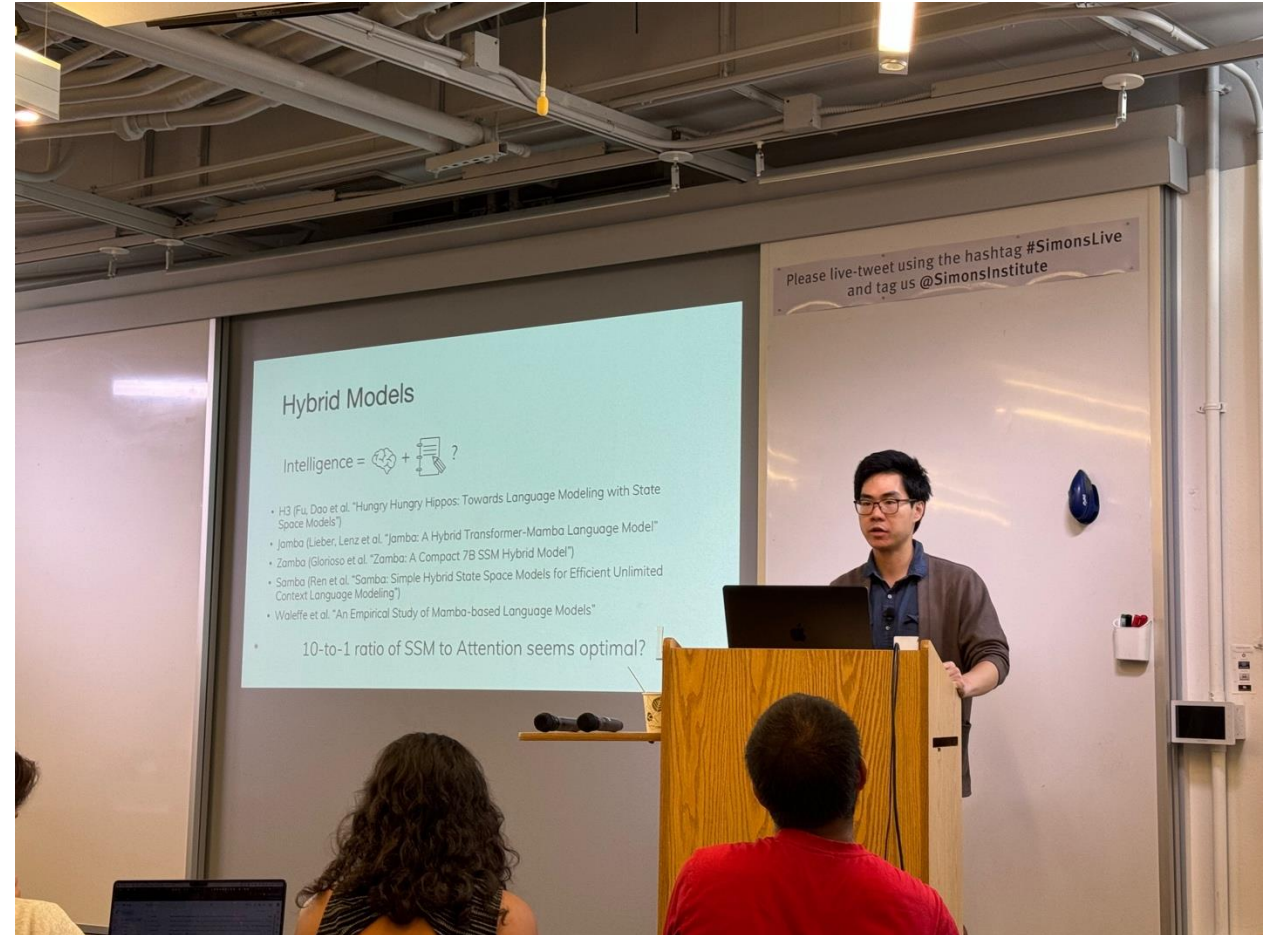


State space models (Gu)

- Combines attention with SSM layers
- S4, Mamba, H3 models
- Gives model variable time to “think”

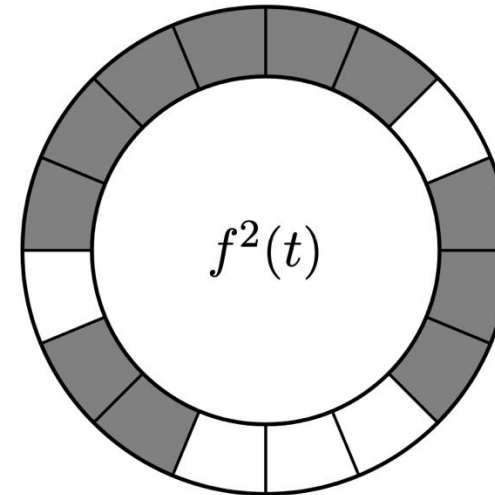
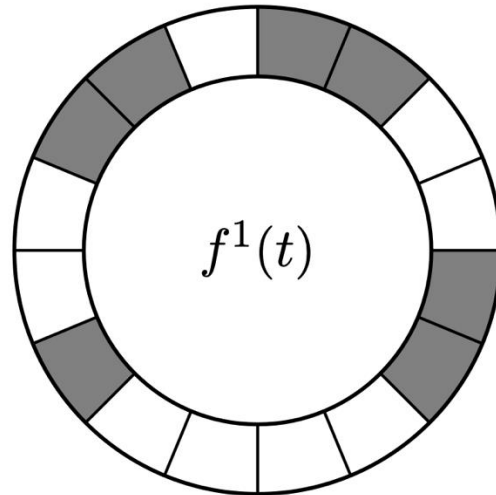
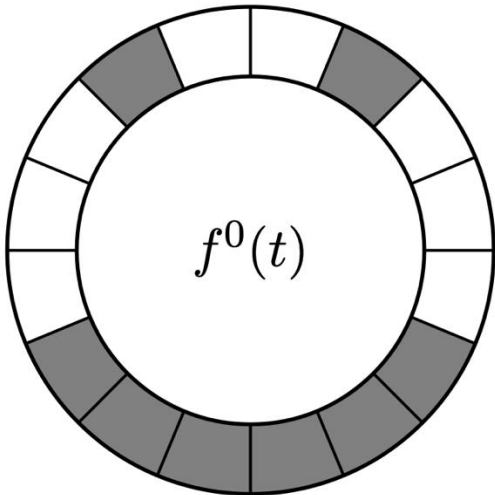
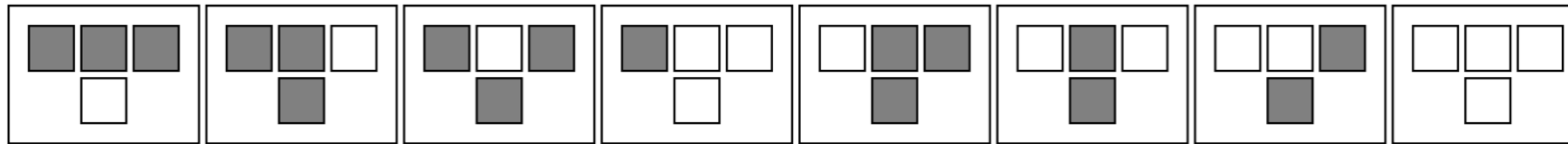
$$x'(t) = \mathbf{A}x(t) + \mathbf{B}u(t)$$

$$y(t) = \mathbf{C}x(t) + \mathbf{D}u(t)$$



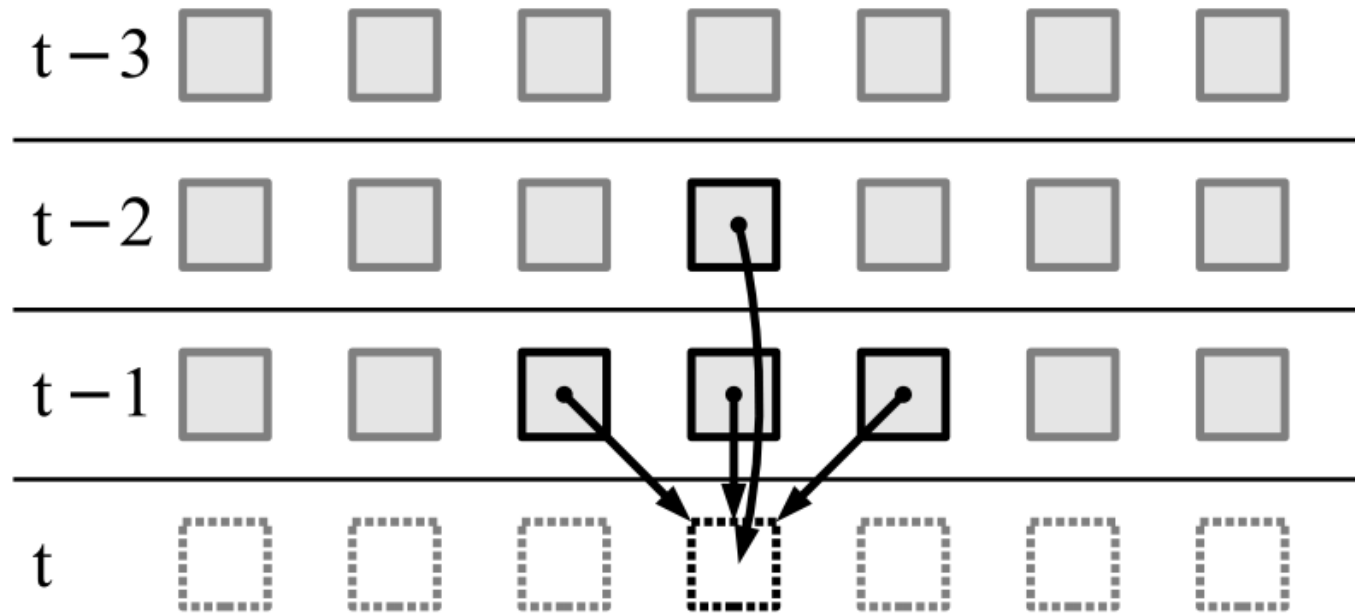
State space models (Breandan's primer)

- Consider the first-order discrete dynamical system



State space models (Breandan's primer)

- Now consider a second-order dynamical system



State space models (Breandan's primer)

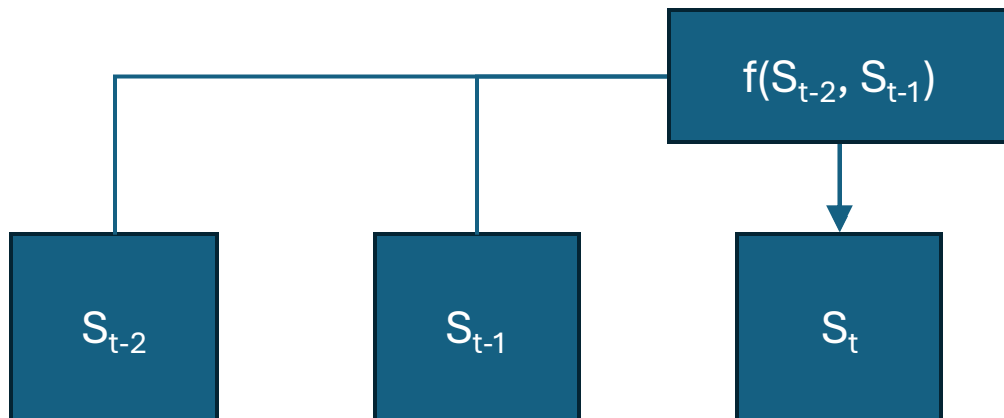
- A very natural question to ask is the following:
How do we represent a non-Markovian dynamical system in a Markovian way?

State space models (Breandan's primer)

- A very natural question to ask is the following:
How do we represent a non-Markovian dynamical system in a Markovian way?
- Using the state-space representation!
- Transforms higher-order dependence into a set of 1st-order (i.e., Markovian) differential equations

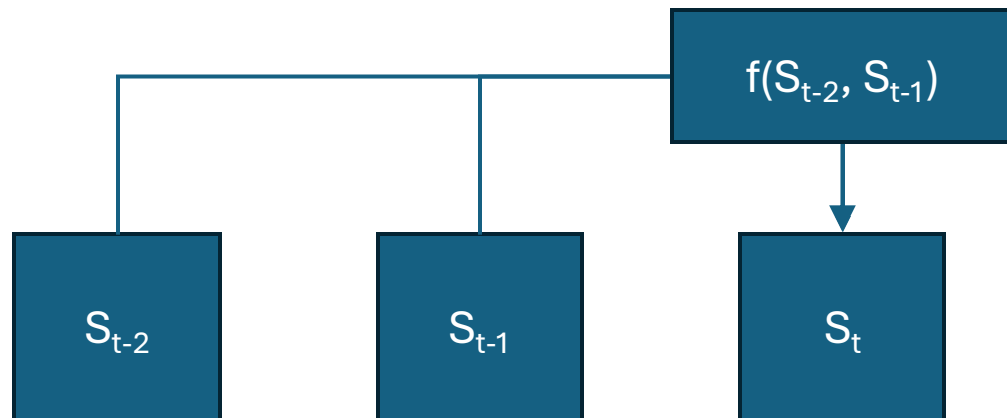
State space models (Breandan's primer)

We can take a second-order differential equation...

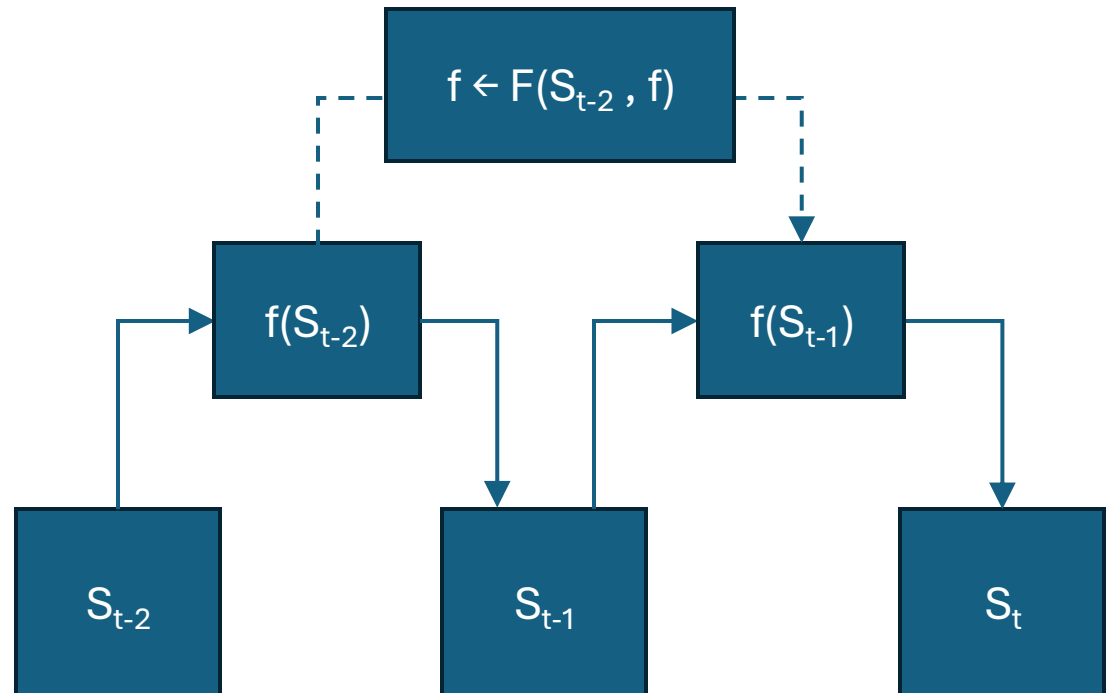


State space models (Breandan's primer)

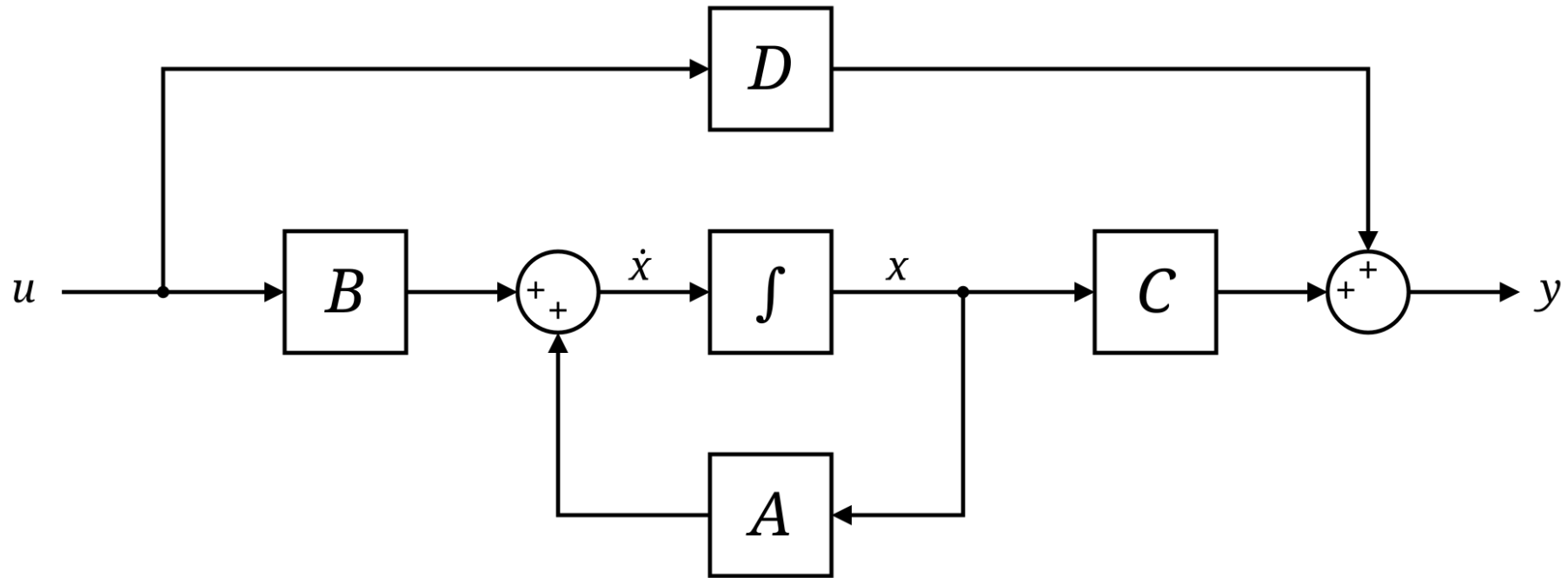
We can take a second-order differential equation...



...and make it stateful with an evolving transition function.



State space models (Breandan's primer)



$$\dot{x}(t) = \mathbf{A}x(t) + \mathbf{B}u(t)$$

$$y(t) = \mathbf{C}x(t) + \mathbf{D}u(t)$$

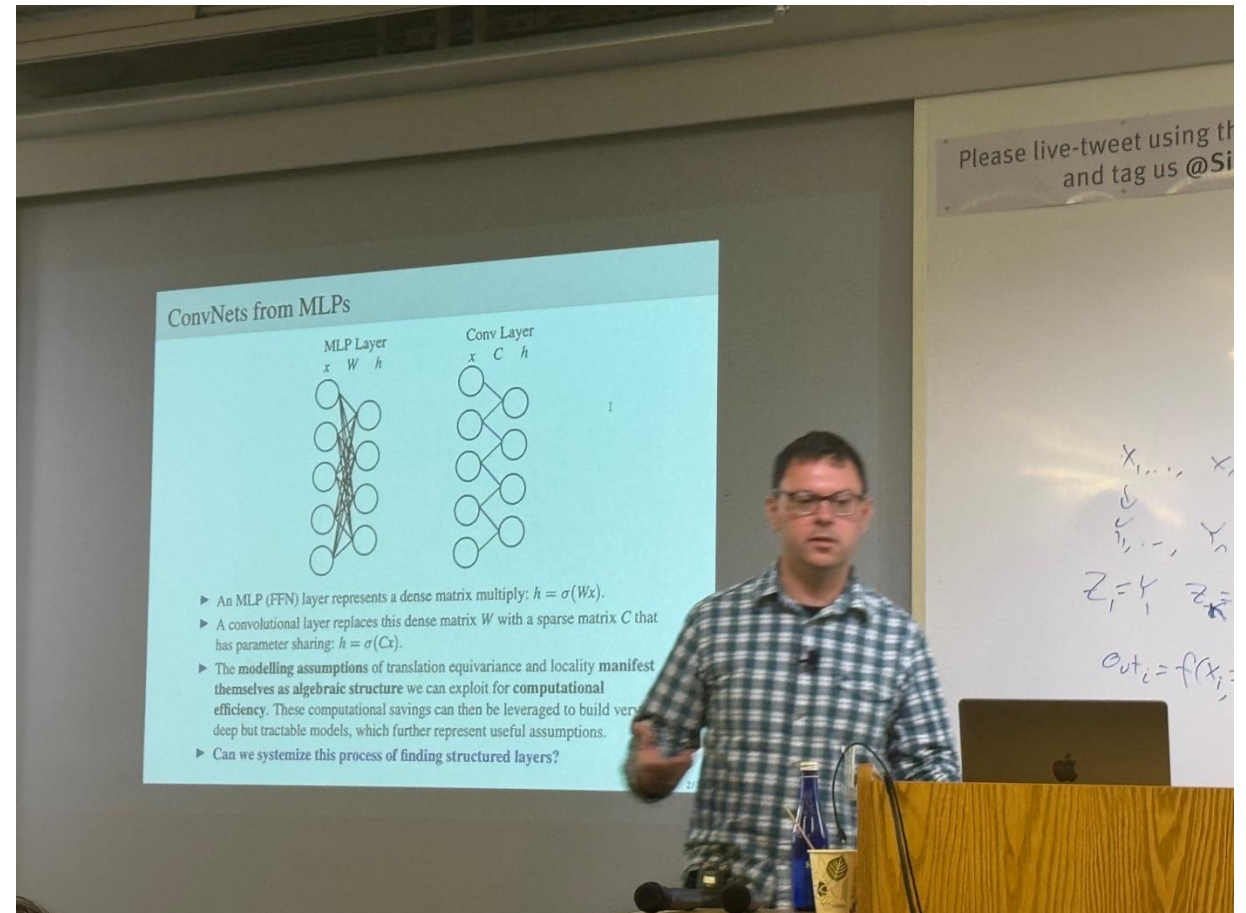
Weak to strong generalization (Goldstein)

- Train on easy reasoning tasks, then evaluate on harder instances of the same task
- Uses recurrent units
- Shows plenty of convincing results on mazes and sudoku



Machine learning is linear algebra (Wilson)

- Inductive biases for pretraining transformers
- Can leverage sparse structure from domain
- Continuous design space of structured matrices and tensors



My poster: Incremental structured decoding

Let's wrap this up! Incremental structured decoding with resource constraints
 Brendan Considine

Main Idea

- Language models have trouble with single-shot constraint satisfaction
- Typically solved via rejection sampling or backtracking style decoders
- We implement an incremental structured decoder for autoregressive LLMs
- Guarantees monotonic progress and preservation of resource constraints
- Ensures all valid words are generable and all generable words are valid

Motivation

Suppose we want to force an autoregressive LLM to generate syntactically valid next tokens $P(x_n | x_1, \dots, x_{n-1})$, under certain resource constraints. Here is a concrete example: "Generate an arithmetic expression with two or more variables in ten or fewer tokens." If we sample the partial trajectory, $(x + (y * ($ then we will spend quite a long time rejecting invalid completions, because this trajectory has passed the point of no return. Even though $($ is a locally valid continuation, we need to avoid this scenario, because we would like a linear sampling delay and to guarantee this, we must avoid backtracking.

Semiring Parsing

Given a CFG, $G: G \rightarrow (V, \Sigma, P, S)$, in Chomsky Normal Form (CNF), we may construct a recognizer $R_G: \Sigma^* \rightarrow \mathbb{B}$ for strings $x: \Sigma^*$ as follows. Let \mathcal{D}^* be our domain, where $0 \leq \mathcal{D}, \mathcal{B}$ in \mathbb{B} , and \mathcal{B} is defined as:

$$R_1 \otimes R_2 = \{C \mid (A, B) \in R_1 \times R_2, C \rightarrow AB \in P\}$$

If we define $\delta_i = \{v \mid (v \rightarrow a_i) \in P\}$, then construct a matrix with unit nonterminals on the superdiagonal, $M_{i,j+1} = \delta_i(G, a_j) = \delta_i$, the flopsum $M_{i,j} = M_i + M_j^T$ is fully determined by the first diagonal:

$$M_0 = \begin{pmatrix} \delta_1 & \delta & \delta \\ \delta & \delta & \delta \\ \delta & \delta & \delta \end{pmatrix} \Rightarrow \dots \Rightarrow M_n = \begin{pmatrix} \delta & \delta & \delta \\ \delta & \delta & \delta \\ \delta & \delta & \delta \end{pmatrix}$$

CFL membership is recognized by $R(G, a) = [S \in A_n] \Leftrightarrow [a \in L(G)]$.

Porous Completion

Let us consider an example with two holes, $a = 1, \dots$ with the grammar being $G = (S \rightarrow NON, O \rightarrow + | \times, N \rightarrow 0 | 1)$. This can be rewritten into CNF as $G' = (S \rightarrow NL, N \rightarrow 0 | 1, O \rightarrow + | \times, L \rightarrow ON)$.

This procedure decides if $\exists a' \in L(G) \mid a' \sqsubset a$ but forgets provenance.



Main Idea

- Language models have trouble with single-shot constraint satisfaction
- Typically solved via rejection sampling or backtracking style decoders
- We implement an incremental structured decoder for autoregressive LLMs
- Guarantees monotonic progress and preservation of resource constraints
- Ensures all valid words are generable and all generable words are valid

Motivation

Suppose we want to force an autoregressive LLM to generate syntactically valid next tokens $P(x_n | x_1, \dots, x_{n-1})$, under certain resource constraints. Here is a concrete example: "Generate an arithmetic expression with two or more variables in ten or fewer tokens." If we sample the partial trajectory,

$(x + (y * ($

then we will spend quite a long time rejecting invalid completions, because this trajectory has passed the point of no return. Even though $($ is a locally valid continuation, we need to avoid this scenario, because we would like a linear sampling delay and to guarantee this, we must avoid backtracking.

Regular Expression Propagation

Regular expressions that permit union, intersection and concatenation are called generalized regular expressions (GREs). These can be constructed as follows:

$$\begin{aligned} \mathcal{L}(\emptyset) &= \emptyset & \mathcal{L}(R^*) &= \{x \mid x \in \mathcal{L}(R) \vee x \in \mathcal{L}(R^*)\} \\ \mathcal{L}(e) &= \{e\} & \mathcal{L}(R \vee S) &= \mathcal{L}(R) \cup \mathcal{L}(S) \\ \mathcal{L}(R \wedge S) &= \mathcal{L}(R) \cap \mathcal{L}(S) & \mathcal{L}(R \wedge S) &= \mathcal{L}(R) \cap \mathcal{L}(S) \\ \mathcal{L}(R \cdot S) &= \mathcal{L}(R) \times \mathcal{L}(S) & \mathcal{L}(\neg R) &= \Sigma^* \setminus \mathcal{L}(R) \end{aligned}$$

Finite slices of a CFL are finite and therefore regular a fortiori. Just like sets, bitvectors and other datatypes, we can also propagate GREs through a parse chart. Here, the algebra will carry GREs, where \emptyset is \emptyset , and \emptyset_0, \emptyset_1 are defined:

$$a_1 \otimes a_2 = \left[\bigvee_{i_1, i_2 \in \{0,1\}} a_{1,i_1} \wedge a_{2,i_2} \right]_{\text{set}} \quad a_1 \otimes a_2 = [a_{1,0} \wedge a_{2,0}] \vee [a_{1,1} \wedge a_{2,1}]_{\text{set}}$$

Initially, we have $M_0[x] = 1 - \mathcal{O}(G, x) = \Sigma$. Now after computing the flopsum, when we unpack $A_n[S]$, this will be a GRE recognizing the finite CFL, also.

Brzozowski Differentiation

Janusz Brzozowski (1964) introduced a derivative operator $\partial_a: R(\Sigma) \rightarrow R(\Sigma)$, which slices a given prefix off a language: $\partial_a L = \{b \in \Sigma^* \mid ab \in L\}$. The Brzozowski derivative over a GRE is effectively a normalizing rewrite system:

$$\begin{aligned} \partial_a \emptyset &= \emptyset & \partial_a (e) &= \emptyset \\ \partial_a e &= \emptyset & \partial_a (e) &= \emptyset \\ \partial_a a &= \varepsilon & \partial_a (e) &= \emptyset \\ \partial_a b &= \emptyset \text{ for each } a \neq b & \partial_a (R^*) &= R^* \\ \partial_a R^* &= R^* & \partial_a (R^*) &= R^* \\ \partial_a \neg R &= \neg \partial_a R & \partial_a (R^*) &= \emptyset \text{ if } R(R) = \emptyset \\ \partial_a R \cdot S &= (\partial_a R) \cdot S \vee R(\partial_a S) & \partial_a (R \cdot S) &= \partial_a R \wedge \partial_a S \\ \partial_a R \vee S &= \partial_a R \vee \partial_a S & \partial_a (R \vee S) &= \partial_a R \vee \partial_a S \\ \partial_a R \wedge S &= \partial_a R \wedge \partial_a S & \partial_a (R \wedge S) &= \partial_a R \wedge \partial_a S \end{aligned}$$

The key property we care about is, this formulation allows us to sample lazily from language intersections, without first materializing the product automaton.

Example: Determinantal Point Processes

Consider a time series, A , whose points which are not too close nor far apart, and $n \leq \sum_{i=1}^n |A_i| = |A|$. We want to sample the typical set using an LLM.

- The words are bitvectors of some length, T , i.e., $A = \{\square, \blacksquare\}^T$
- Consecutive \blacksquare separated by \square^{b-1} , i.e., $B = \{\square^{b-1} \blacksquare^{a(n)} \blacksquare, \square\}^{\square}$

The DPP language is regular. Let C be an FSA such that $\mathcal{L}(C) = \mathcal{L}(A) \cap \mathcal{L}(B)$. For example, here is the minimal automaton for $T = 15, a = 3, b = 5, n = 2$.



This automaton for $\mathcal{L}(C)$ can grow very large, and we may only need to sample a small sublanguage with distributed support. Question: Can we incrementally subsample $\mathcal{L}(C)$ while ensuring partial trajectories always lead to acceptance?

My poster: Incremental structured decoding

Let's wrap this up! Incremental structured decoding with resource constraints
 Brendan Considine

Main Idea

- Language models have trouble with single-shot constraint satisfaction
- Typically solved via rejection sampling or backtracking style decoders
- We implement an incremental structured decoder for autoregressive LLMs
- Guarantees monotonic progress and preservation of resource constraints
- Ensures all valid words are generable and all generable words are valid

Motivation

Suppose we want to force an autoregressive LLM to generate syntactically valid nest tokens $P(x_1, \dots, x_n)$ under certain resource constraints. Here is a concrete example: "Generate an arithmetic expression with two or more variables in ten or fewer tokens". If we sample the partial trajectory $x = (x_1, \dots, x_n)$, then we will spend quite a long time rejecting invalid completions, because this trajectory has passed the point of no return. Even though x is a locally valid continuation, we need to avoid this scenario, because we would like a linear sampling delay and to guarantee this, we must avoid backtracking.

Semiring Parsing

Given a CFG, $G: \mathcal{G} = (V, \Sigma, P, S)$, in Chomsky Normal Form (CNF), we may construct a recognizer $R_G: \Sigma^* \rightarrow \mathbb{B}$ for strings $\sigma: \Sigma^*$ as follows. Let \mathcal{D}^G be our domain, where $0 \in \mathcal{D}$, $1 \in \mathcal{D}$, and \oplus is defined as:

$$s_1 \otimes s_2 = [C] \{ (A, B) \in s_1 \times s_2 \mid C \rightarrow AB \} \in P$$

If we define $\delta_i = \{ \sigma \mid (v \rightarrow \sigma_i) \in P \}$, then construct a matrix with unit nonterminals on the superdiagonal, $M_0[r+1] = \delta(G, \sigma) = \delta_i$, the fixpoint $M_{\infty} = M_0 + M_1^2$ is fully determined by the first diagonal:

$$M_0 = \begin{pmatrix} \delta_1 & \delta_2 & \delta_3 \\ \delta_2 & \delta_3 & \delta_4 \\ \delta_3 & \delta_4 & \delta_5 \end{pmatrix} \Rightarrow \dots \Rightarrow M_{\infty} = \begin{pmatrix} \delta_1 & \delta_2 & \delta_3 \\ \delta_2 & \delta_3 & \delta_4 \\ \delta_3 & \delta_4 & \delta_5 \end{pmatrix}$$

CFL membership is recognized by $R(G, \sigma) = [\sigma \in \Lambda_G^*] \Leftrightarrow [\sigma \in \mathcal{L}(G)]$.

Porous Completion

Let us consider an example with two holes, $\sigma = 1 \dots$ with the grammar being $G = \{ S \rightarrow NOY, O \rightarrow \{ \times, N \rightarrow 0 \} \}$. This can be rewritten into CNF as $G' = \{ S \rightarrow NL, N \rightarrow 0 \mid 1, O \rightarrow \{ \times, L \rightarrow ON \}$.

My poster: Incremental structured decoding

Let's wrap this up! Incremental structured decoding with resource constraints
 Brendan Considine

Main Idea

- Language models have trouble with single-shot constraint satisfaction
- Typically solved via rejection sampling or backtracking style decoders
- We implement an incremental structured decoder for autoregressive LLMs
- Guarantees monotonic progress and preservation of resource constraints
- Ensures all valid words are generable and all generable words are valid

Motivation

Suppose we want to force an autoregressive LLM to generate syntactically valid nest tokens $P(x_1 | x_2, \dots, x_{n-1})$ under certain resource constraints. Here is a concrete example: "Generate an arithmetic expression with two or more variables in ten or fewer tokens". If we sample the partial trajectory $x = (x \cdot y \cdot z)$, then we will spend quite a long time rejecting invalid completions, because this trajectory has passed the point of no return. Even though $(x$ is a locally valid continuation, we need to avoid this scenario, because we would like a linear sampling delay and to guarantee this, we must avoid backtracking.

Semiring Parsing

Given a CFG, $G: G \rightarrow (V, \Sigma, P, S)$ in Chomsky Normal Form (CNF), we may construct a recognizer $R_G: \Sigma^* \rightarrow \mathbb{B}$ for strings $x: \Sigma^*$ as follows. Let \mathcal{D}^x be our domain, where $0 \leq \mathcal{D}, \mathcal{D} \in \mathbb{N}$, and \mathcal{D} is defined as:

$$x_1 \otimes x_2 = [C] \{A, B\} \in x_1 \times x_2 \{C \rightarrow AB\} \in P$$

If we define $\delta_x = \{v \mid (v \rightarrow x) \in P\}$, then construct a matrix with unit nonterminals on the superdiagonal, $M_G(x+1) = \delta(G, x) = \delta_x$ the fixpoint $M_{G,1} = M_1 + M_1^2$ is fully determined by the first diagonal:

$$M_G = \begin{pmatrix} \delta & \emptyset & \emptyset \\ \emptyset & \delta & \emptyset \\ \emptyset & \emptyset & \delta \end{pmatrix} \Rightarrow \dots \Rightarrow M_{G,n} = \begin{pmatrix} \delta & \delta & \delta \\ \delta & \delta & \delta \\ \delta & \delta & \delta \end{pmatrix}$$

CFL membership is recognized by $R(G, x) = [S] \in \Lambda_G^* \Leftrightarrow [S] \in \mathcal{L}(G)$.

Porous Completion

Let us consider an example with two holes, $\sigma = 1, \dots$ with the grammar being $G = \langle S \rightarrow NON, O \rightarrow \{x, N \rightarrow 0\} \rangle$. This can be rewritten into CNF as $G' = \langle S \rightarrow NL, N \rightarrow 0 \mid 1, O \rightarrow \{x, L \rightarrow ON\} \rangle$.

This procedure decides if $[3\sigma] \in \mathcal{L}(G)$ if $\sigma' \in \mathcal{L}(G)$ but forgets provenance.



Regular Expression Propagation

Regular expressions that permit union, intersection and concatenation are called generalized regular expressions (GREs). These can be constructed as follows:

$$\begin{aligned} \mathcal{L}(\emptyset) &= \emptyset & \mathcal{L}(R^*) &= (\varepsilon) \cup \mathcal{L}(R \cdot R^*) \\ \mathcal{L}(\varepsilon) &= \{\varepsilon\} & \mathcal{L}(R \vee S) &= \mathcal{L}(R) \cup \mathcal{L}(S) \\ \mathcal{L}(a) &= \{a\} & \mathcal{L}(R \wedge S) &= \mathcal{L}(R) \cap \mathcal{L}(S) \\ \mathcal{L}(R \cdot S) &= \mathcal{L}(R) \times \mathcal{L}(S) & \mathcal{L}(\neg R) &= \Sigma^* \setminus \mathcal{L}(R) \end{aligned}$$

Finite slices of a CFL are finite and therefore regular a fortiori. Just like sets, bitvectors and other datatypes, we can also propagate GREs through a parse chart. Here, the algebra will carry GREs, where \emptyset is \emptyset , and \emptyset, \emptyset are defined:

$$x_1 \otimes x_2 = \left[\bigvee_{i_1 \rightarrow i_2 \rightarrow j} x_{i_1}[A] \cdot x_{i_2}[B] \right]_{i_1 \rightarrow j} \quad x_1 \otimes x_2 = [x_1[i] \vee x_2[i]]_{i \rightarrow j}$$

Initially, we have $M_G(x+1) = \delta(G, x) = \Sigma$. Now after computing the fixpoint, when we unpack $\Lambda_G^*[S]$, this will be a GRE recognizing the finite CFL slice.

Brzozowski Differentiation

Janusz Brzozowski (1964) introduced a derivative operator $\partial_x: \text{REG} \rightarrow \text{REG}$, which slices a given prefix off a language: $\partial_x L = \{b \in \Sigma^* \mid axb \in L\}$. The Brzozowski derivative over a GRE is effectively a normalizing rewrite system:

Example: Determinantal Point Processes

Consider a time series, A , whose points which are not too close nor far apart, and $n \leq \sum_{i=1}^n |A_i| = |A|$. We want to sample the typical set using an LLM.

- The words are bitvectors of some length, T , i.e., $A = \{[A_i] \in \mathbb{B}^T\}$
- Consecutive A_i separated by $\square^{(b)}$, i.e., $B = \{C \in \mathbb{B}^{(b) \times (b+n)} \mid [A_i] \in C\}$

The DPP language is regular. Let C be an FSA such that $\mathcal{L}(C) = \mathcal{L}(A) \cap \mathcal{L}(B)$. For example, here is the minimal automaton for $T=15, a=3, b=5, n=2$.

This automaton for $\mathcal{L}(C)$ can grow very large, and we may only need to sample a small sublanguage with distributed support. Question: Can we incrementally subsample $\mathcal{L}(C)$ while ensuring partial trajectories always lead to acceptance?

Brzozowski Differentiation

Janusz Brzozowski (1964) introduced a derivative operator $\partial_a: \text{REG} \rightarrow \text{REG}$, which slices a given prefix off a language: $\partial_a L = \{b \in \Sigma^* \mid ab \in L\}$. The Brzozowski derivative over a GRE is effectively a normalizing rewrite system:

$$\begin{aligned} \partial_a \emptyset &= \emptyset & \delta(\emptyset) &= \emptyset \\ \partial_a \varepsilon &= \emptyset & \delta(\varepsilon) &= \varepsilon \\ \partial_a a &= \varepsilon & \delta(a) &= \emptyset \\ \partial_a b &= \emptyset \text{ for each } a \neq b & \delta(R^*) &= \varepsilon \\ \partial_a R^* &= (\partial_x R) \cdot R^* & \delta(\neg R) &= \varepsilon \text{ if } \delta(R) = \emptyset \\ \partial_a \neg R &= \neg \partial_a R & \delta(\neg R) &= \emptyset \text{ if } \delta(R) = \varepsilon \\ \partial_a R \cdot S &= (\partial_a R) \cdot S \vee \delta(R) \cdot \partial_a S & \delta(R \cdot S) &= \delta(R) \wedge \delta(S) \\ \partial_a R \vee S &= \partial_a R \vee \partial_a S & \delta(R \vee S) &= \delta(R) \vee \delta(S) \\ \partial_a R \wedge S &= \partial_a R \wedge \partial_a S & \delta(R \wedge S) &= \delta(R) \wedge \delta(S) \end{aligned}$$

The key property we care about is, this formulation allows us to sample lazily from language intersections, without first materializing the product automaton.

Visit to UC Berkeley

