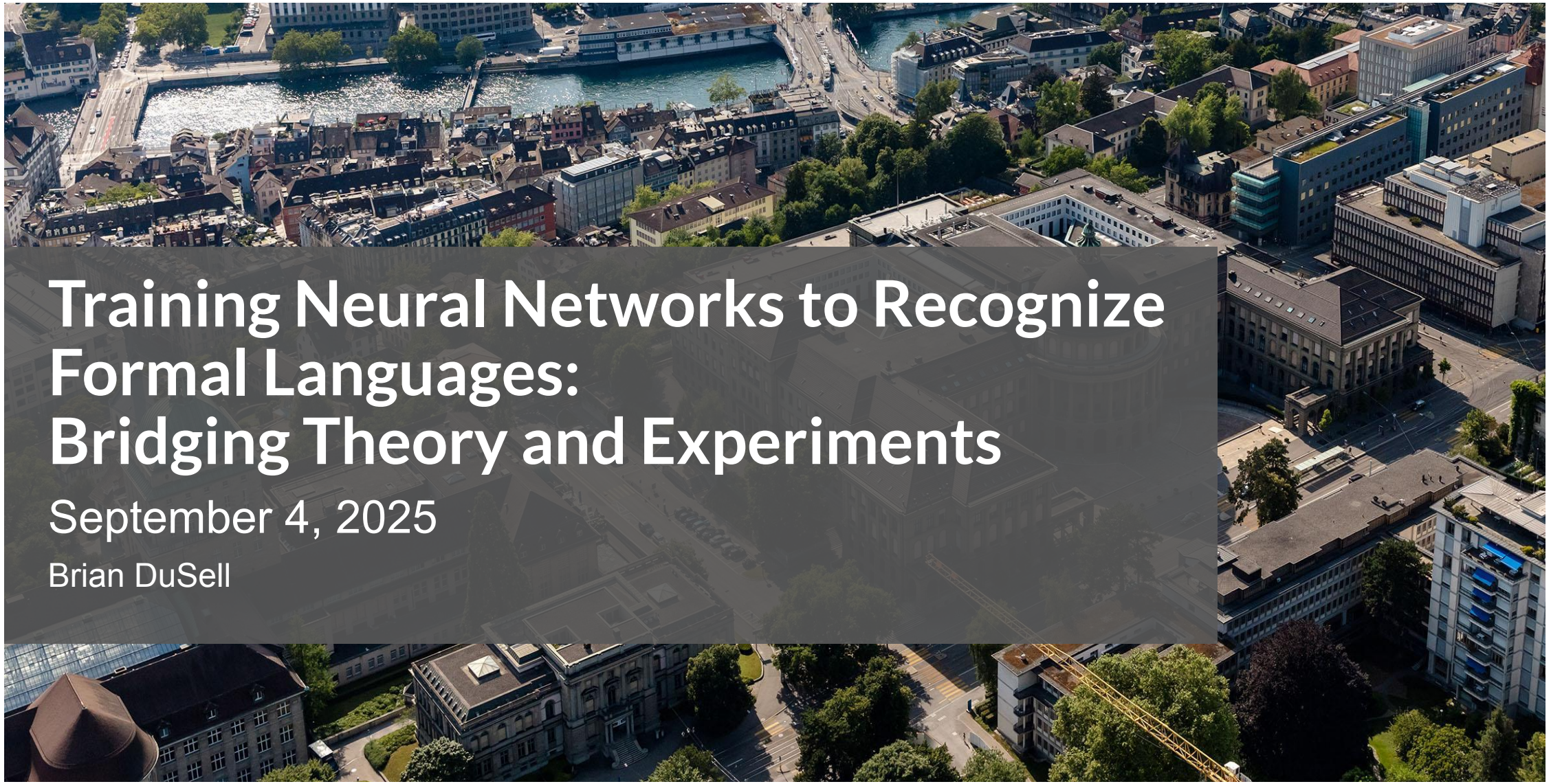


Training Neural Networks to Recognize Formal Languages: Bridging Theory and Experiments

September 4, 2025

Brian DuSell

ETH zürich

About Me

- Postdoc at ETH Zürich with Ryan Cotterell
- Formerly: PhD student at University of Notre Dame with David Chiang
- Interests
  - Natural language processing
  - Formal language theory
  - Neural networks

Basic Research Questions

What can LLMs do?

What can't LLMs do?

Published as a conference paper at ICLR 2025

TRAINING NEURAL NETWORKS AS RECOGNIZERS OF FORMAL LANGUAGES

Alexandra Butoi[1] **Ghazal Khalighinejad**[2] **Anej Svete**[1]
Josef Valvoda[3] **Ryan Cotterell**[1] **Brian DuSell**[1]
[1]ETH Zürich [2]Duke University [3]University of Copenhagen
{alexandra.butoi, anej.svete, ryan.cotterell, brian.dusell}@inf.ethz.ch
ghazal.khalighinejad@duke.edu jval@di.ku.dk

ABSTRACT

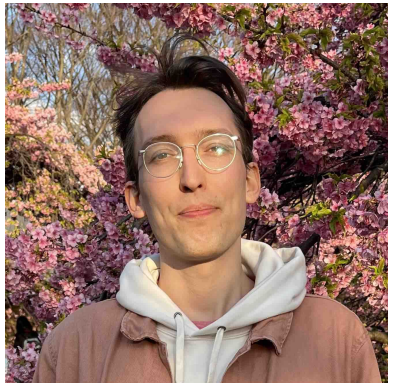
Characterizing the computational power of neural network architectures in terms of formal language theory remains a crucial line of research, as it describes lower and upper bounds on the reasoning capabilities of modern AI. However, when empirically testing these bounds, existing work often leaves a discrepancy between experiments and the formal claims they are meant to support. The problem is that

My Collaborators

Alexandra Butoi

Ghazal Khalighinejad

Anej Svete



Josef Valvoda

Ryan Cotterell

Definition: Alphabet

An **alphabet** is a non-empty finite set of elements called **symbols**.

Examples:

- { 0, 1 }
- { 0, 1, # }
- the set of all Unicode characters
- { above, accept, admire, ..., zebra, zebras, ., ? }

Definition: String

A **string** is finite sequence of symbols from an alphabet.

Examples:

- 0 1 0 1 1
- 0 1 0 1 1 # 0 1 0 1 1
- the salamanders don't amuse my newt .

Definition: Language

A **language** or **formal language** is a (possibly infinite) set of strings.

Examples:

- { #, 0 # 0, 1 # 1, 0 0 # 0 0, 0 1 # 0 1, 1 0 # 1 0, 1 1 # 1 1, ... }
  = { w#w | w ∈ {0, 1}* }
- { the salamanders don't amuse my newt . ,
  our zebra doesn't applaud the unicorn . ,
  some newt doesn't comfort the ravens . ,
  ... }

Membership in a Language

$\{ w\#w \mid w \in \{0, 1\}^* \}$

✓ 0 1 1 # 0 1 1

✗ 0 1 1 # 0 1 0

✗ # 0 # # 1 0

Natural Language as a Formal Language

✓ your orangutans giggle

✗ orangutans giggle your

✓ your vulture comforts our peacocks by our newts

✗ peacocks by comforts newts our your our vulture

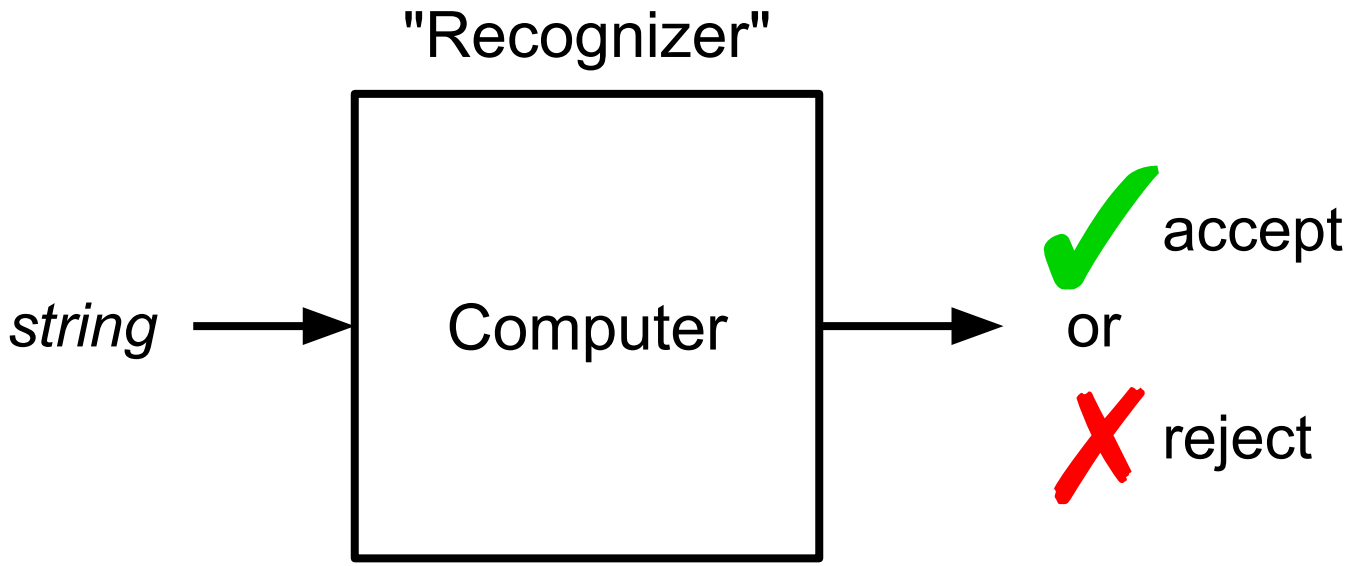
✓ some newts that our zebra amuses wait

✗ some newts that our zebra amuses waits

Definition: Recognizer

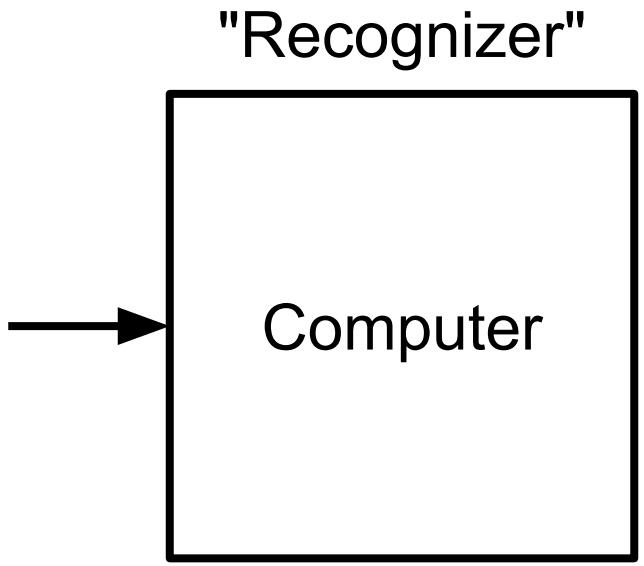
A **recognizer** is any computing device that reads a string as input and produces a decision to **accept** or **reject** it as output.

A recognizer **recognizes** the language of strings that it accepts.



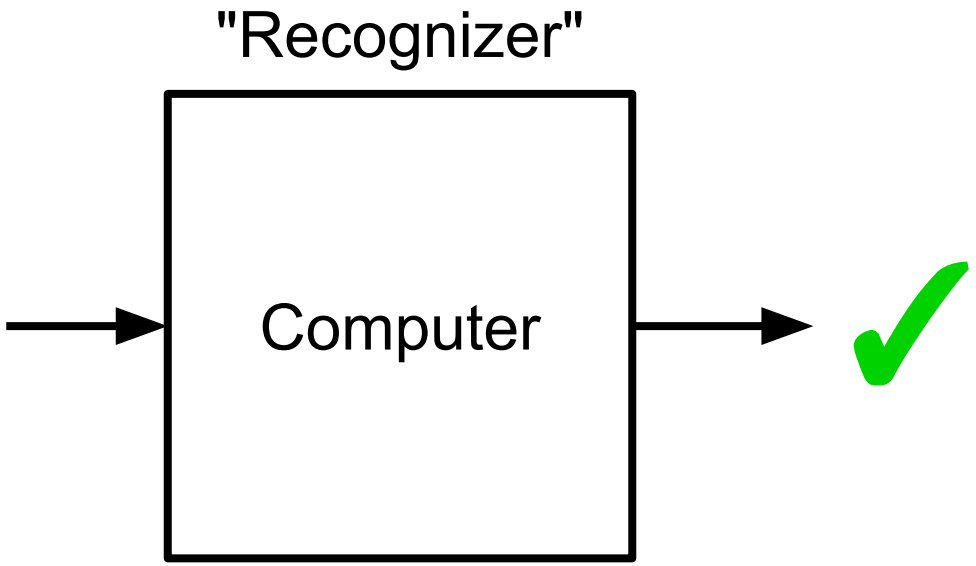
Recognizers

"some newts that our zebra amuses wait"



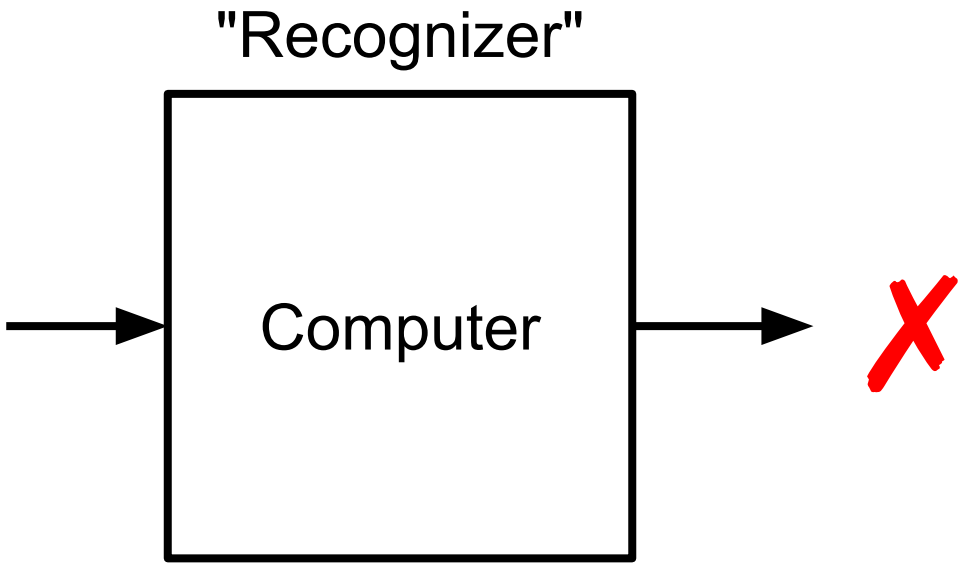
Recognizers

"some newts that our zebra amuses wait"

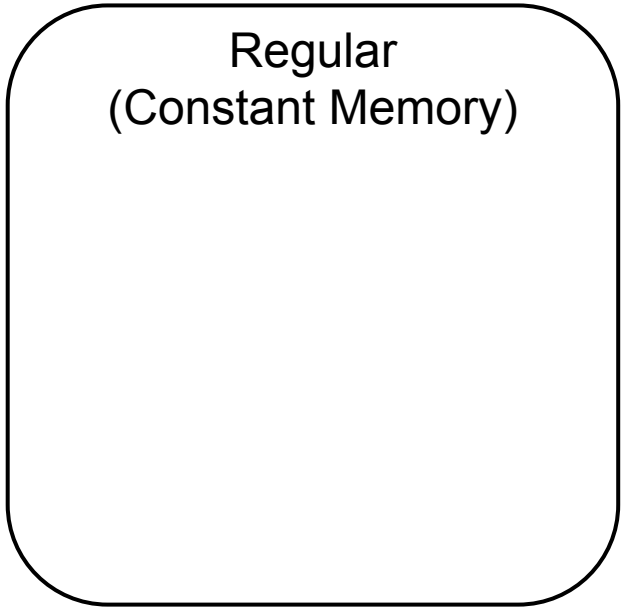


Recognizers

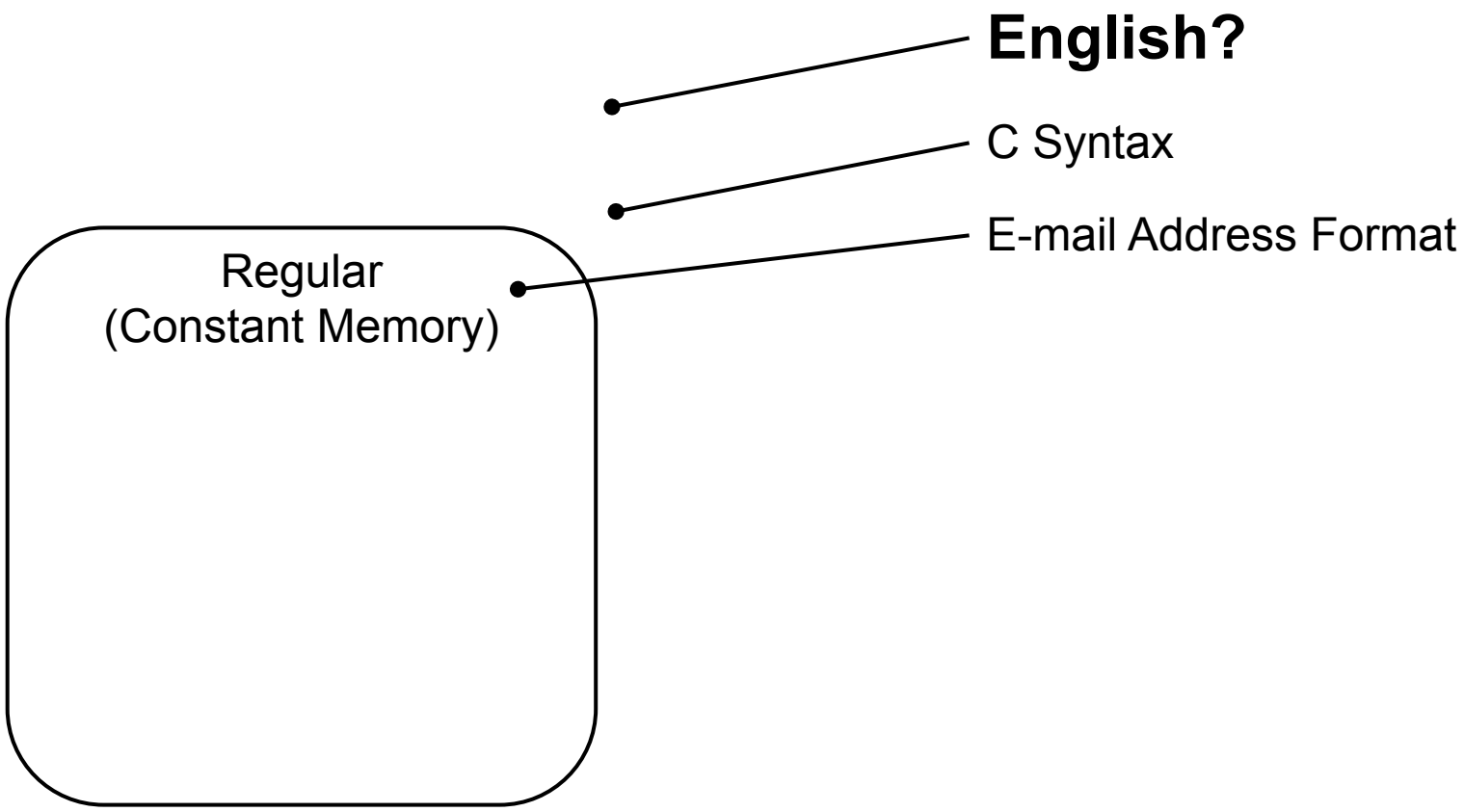
"some newts that our zebra amuses waits"



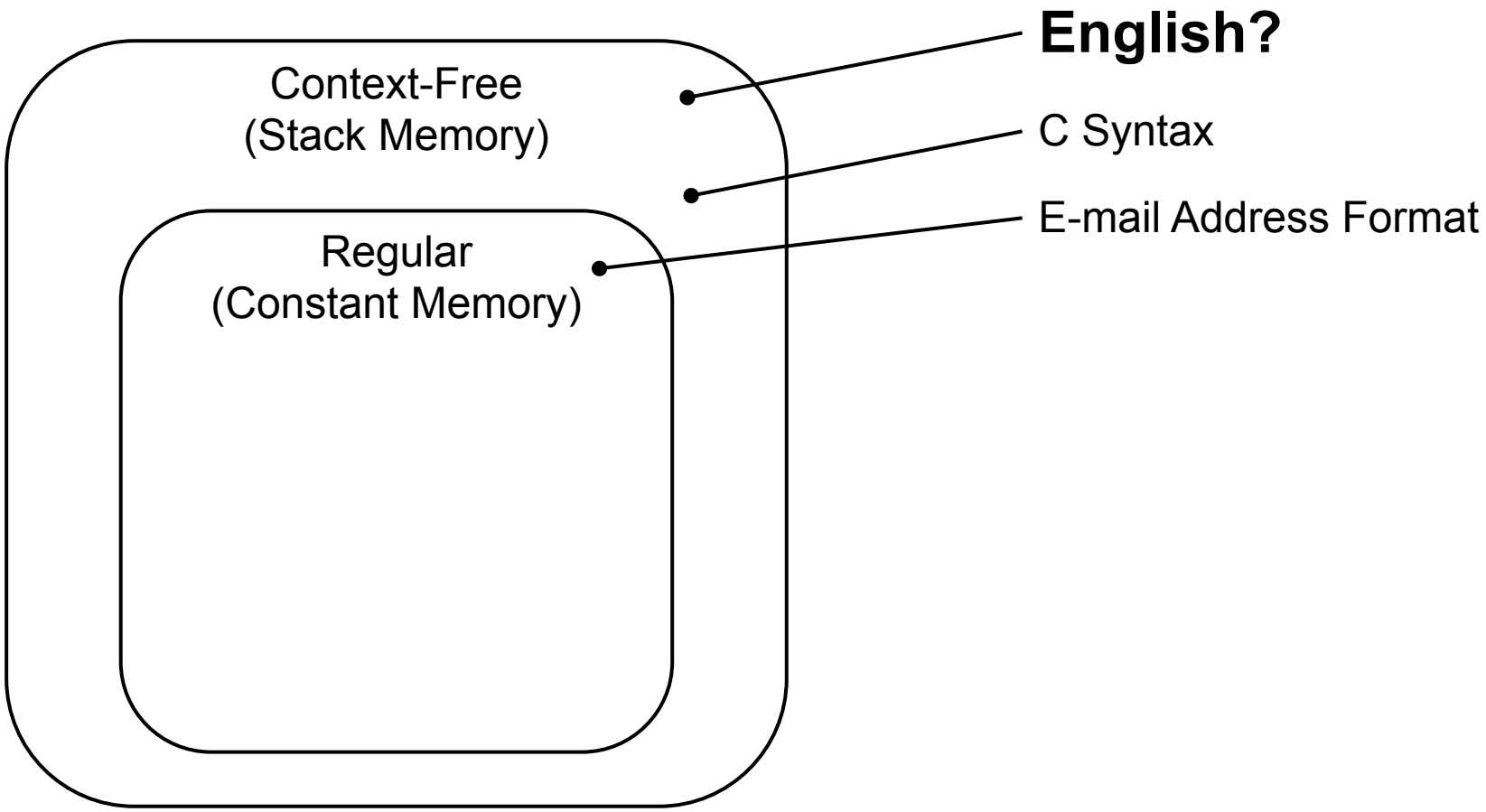
Chomsky Hierarchy



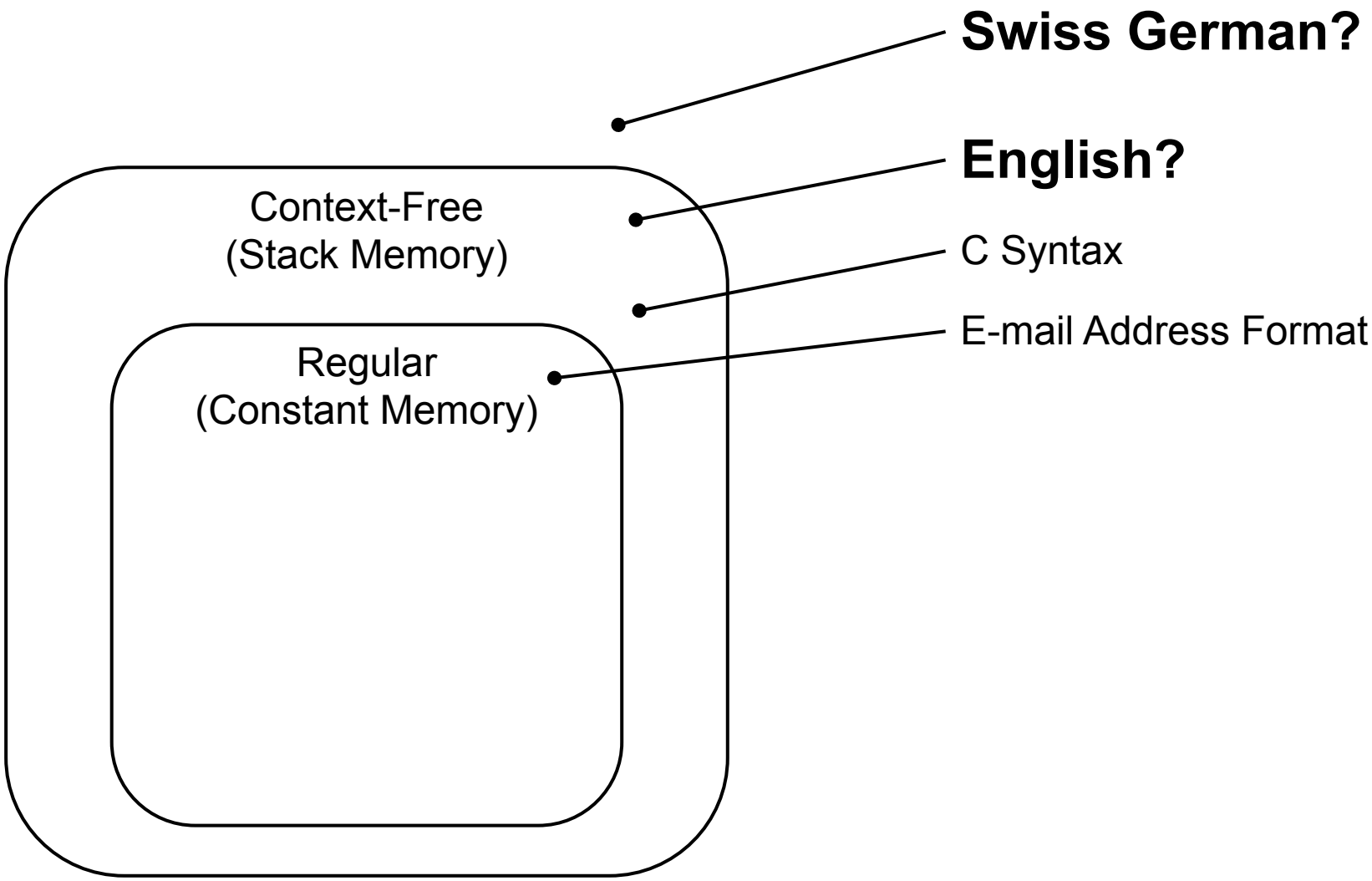
Chomsky Hierarchy



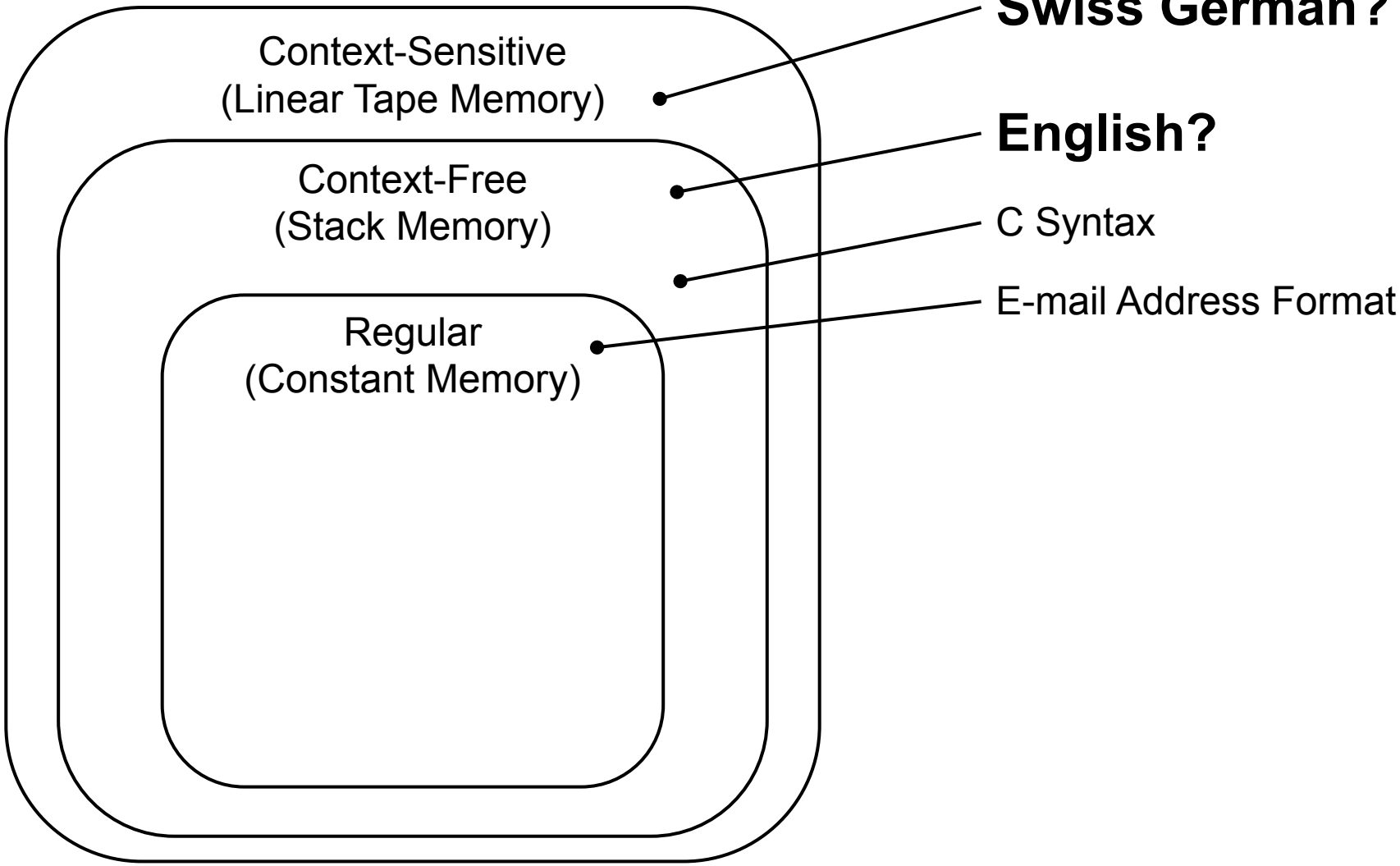
Chomsky Hierarchy



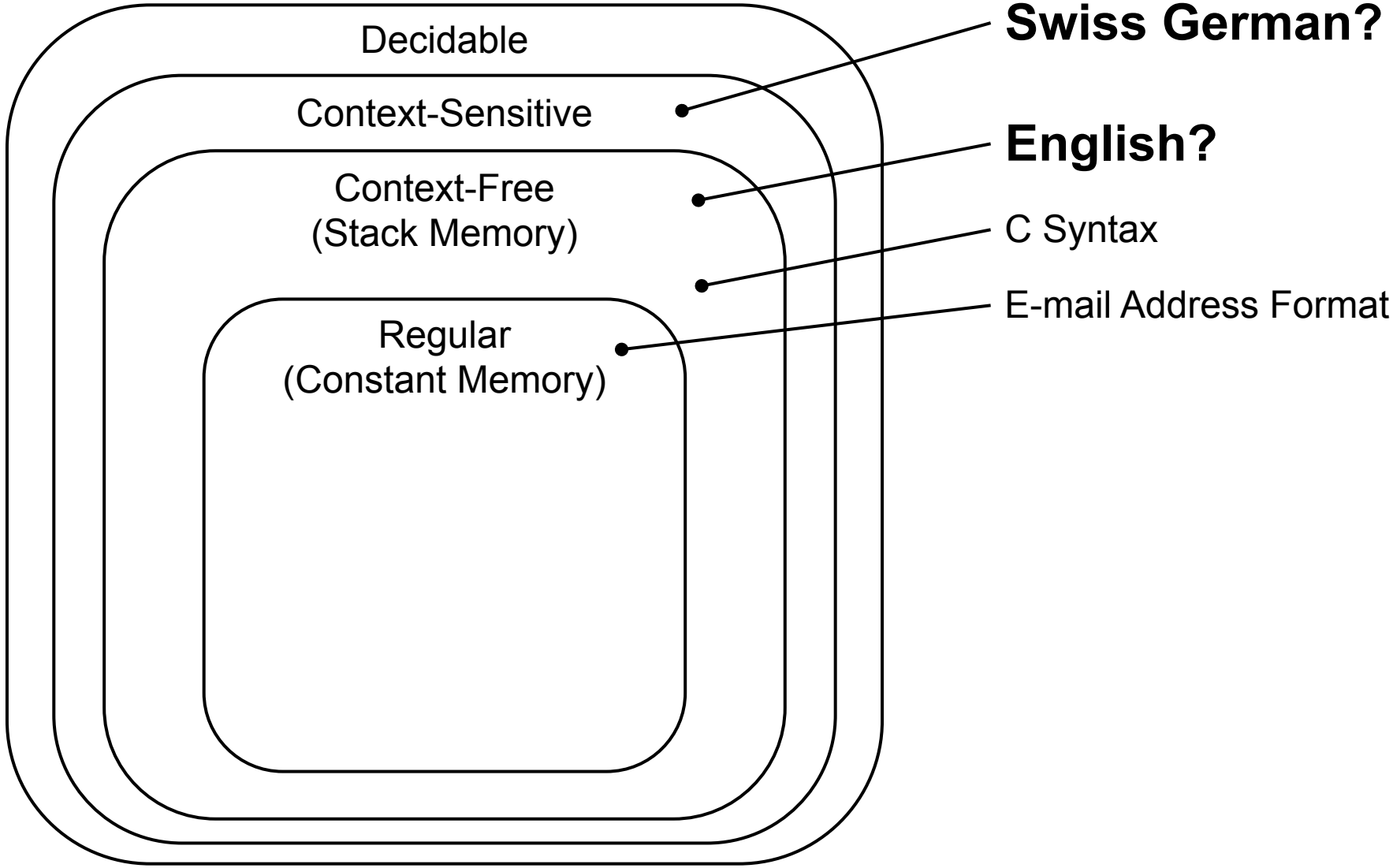
Chomsky Hierarchy



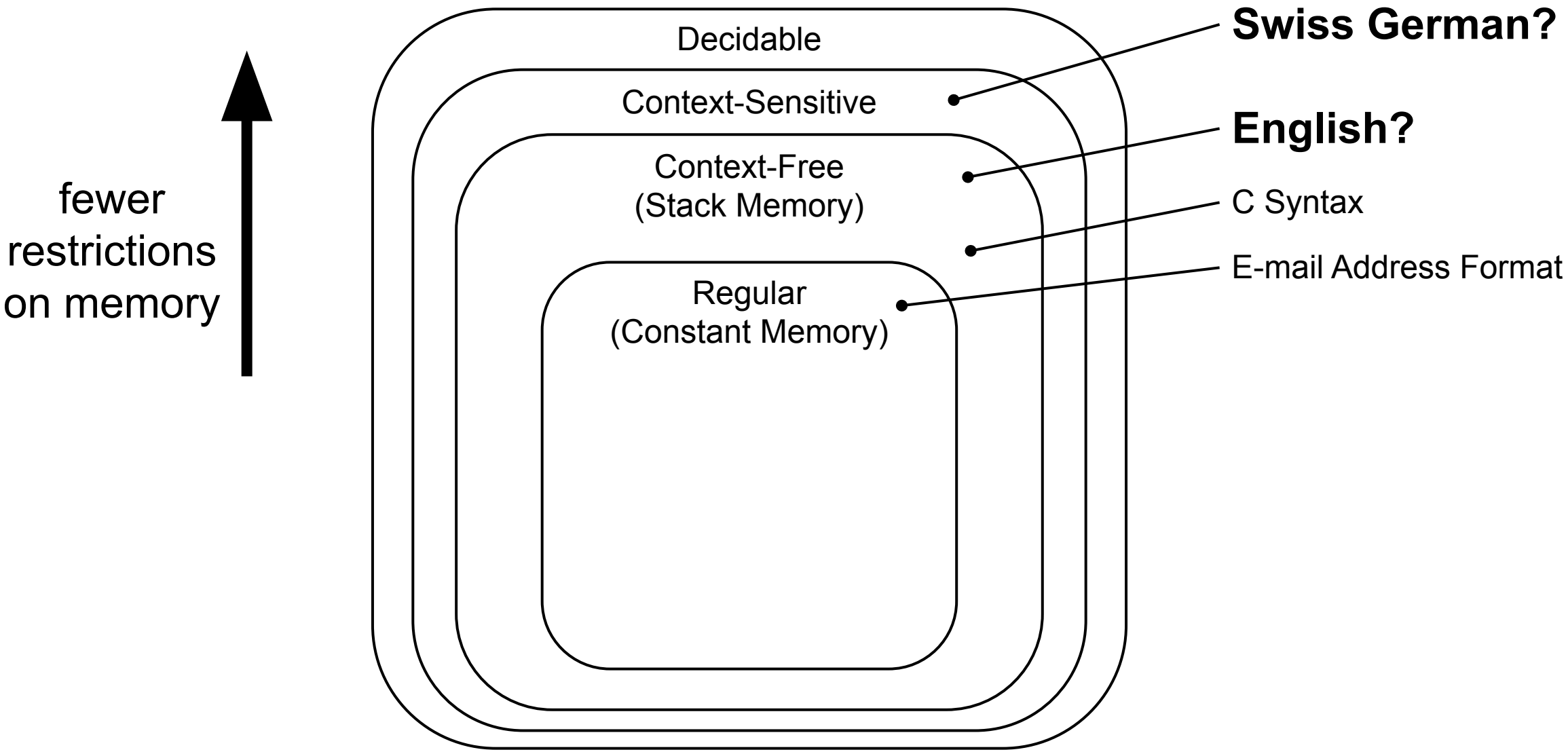
Chomsky Hierarchy



Chomsky Hierarchy



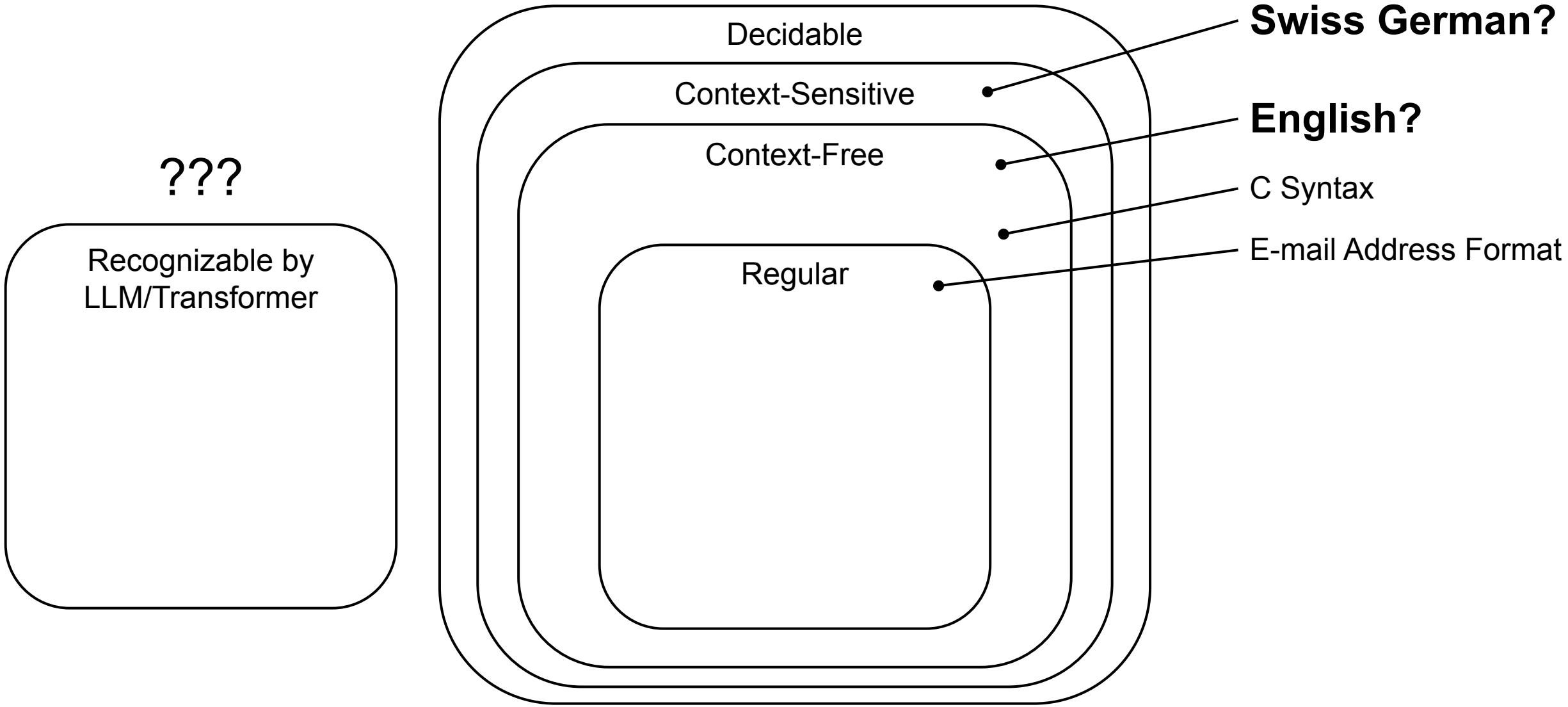
Chomsky Hierarchy



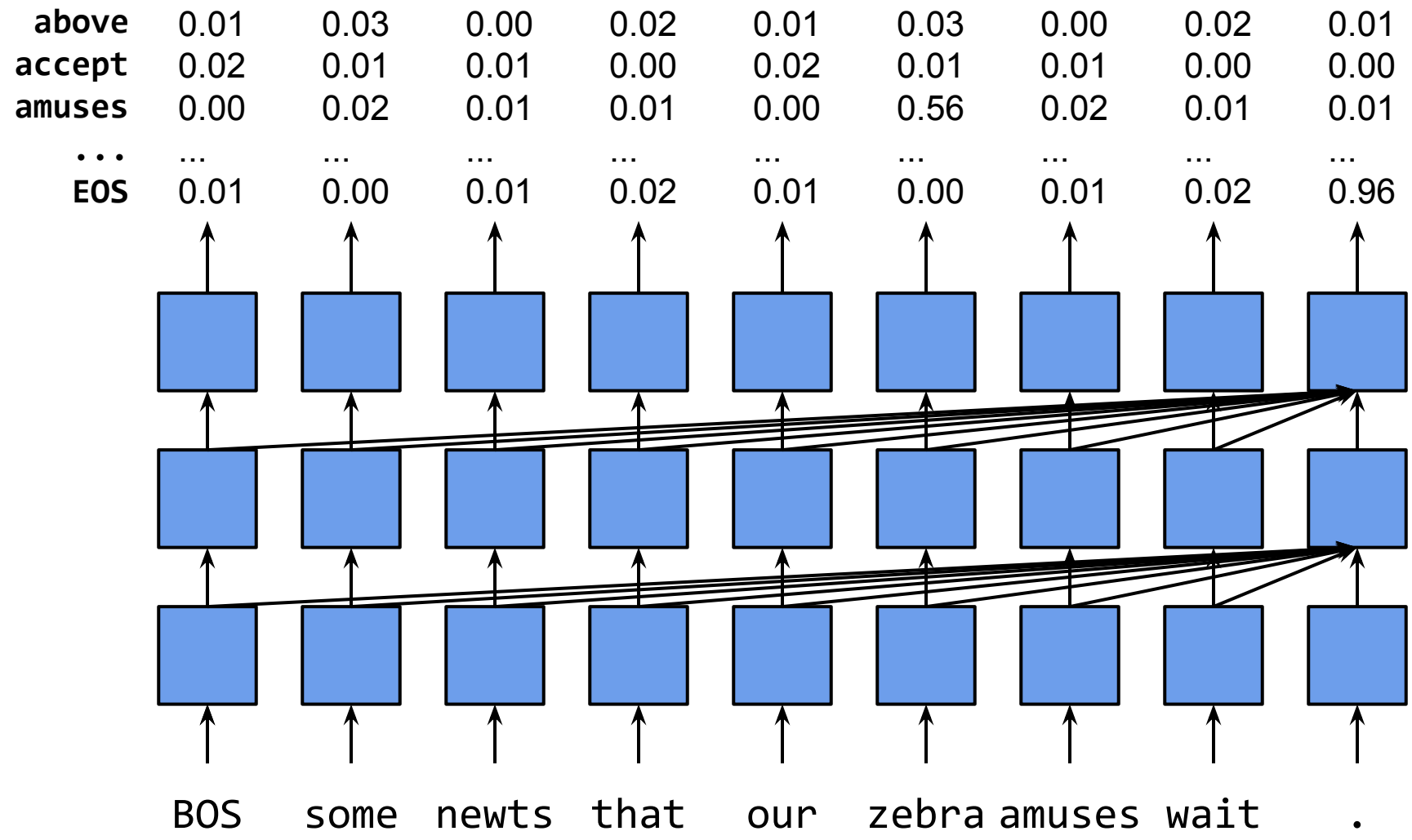
Definition: Class

A **class** or **language class** is a (possibly infinite) set of languages.

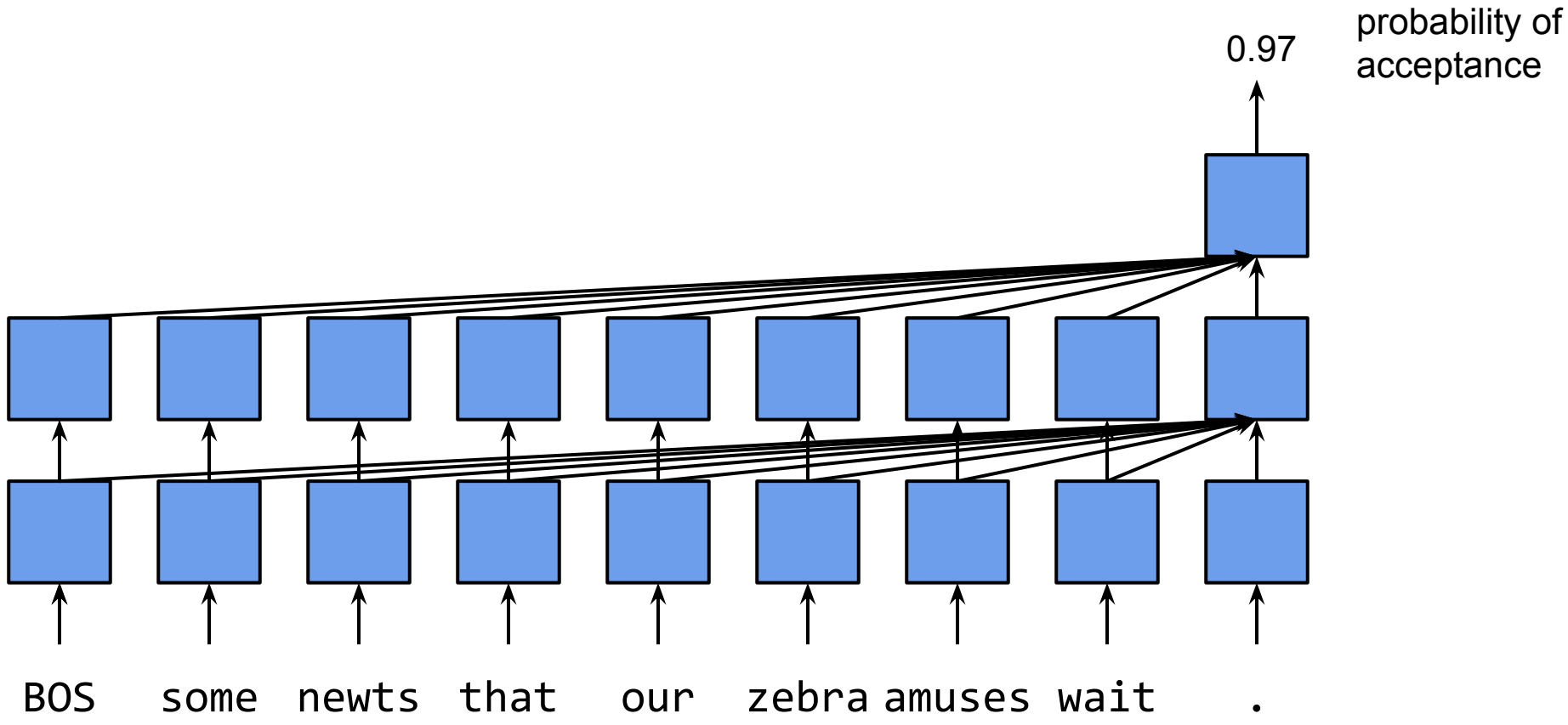
Chomsky Hierarchy



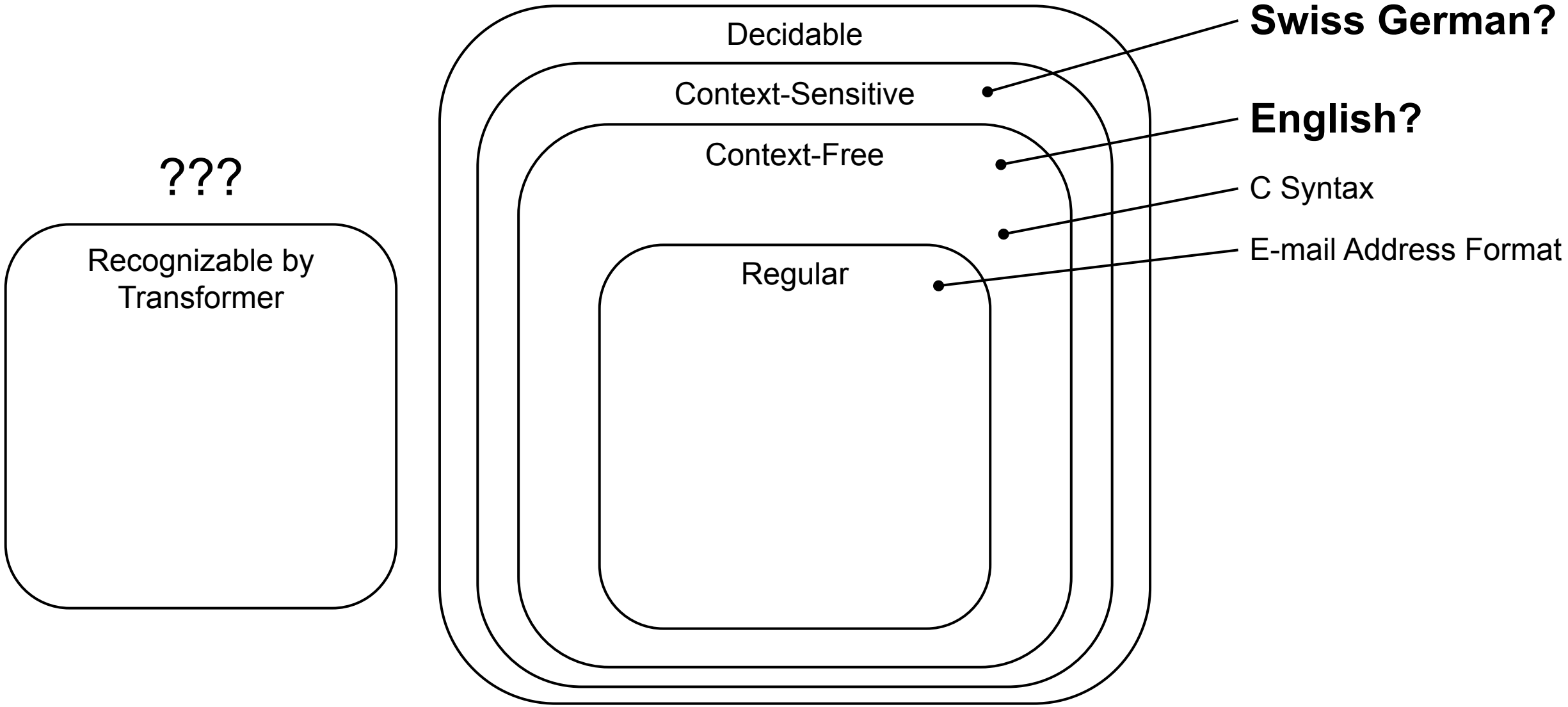
Transformer Language Model



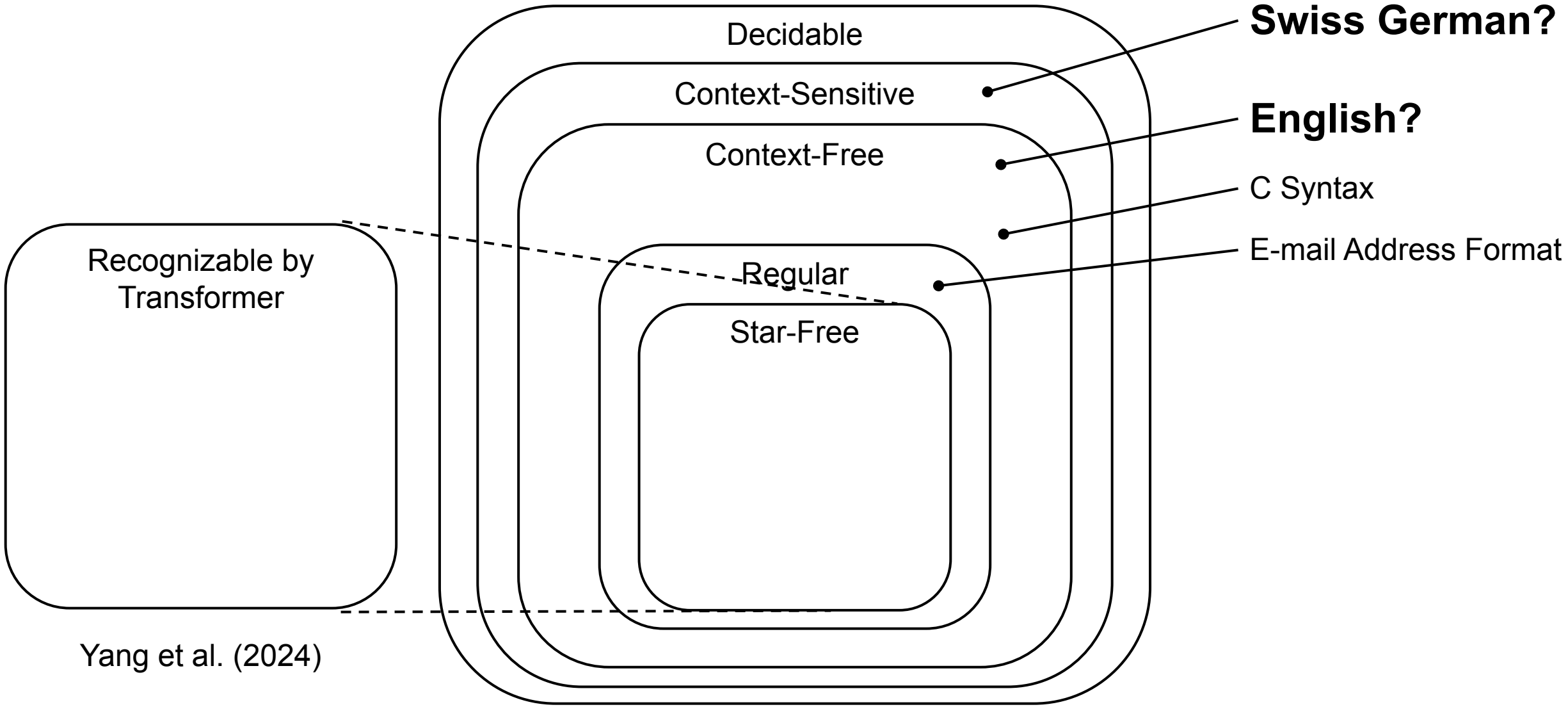
Transformer Recognizer



Chomsky Hierarchy

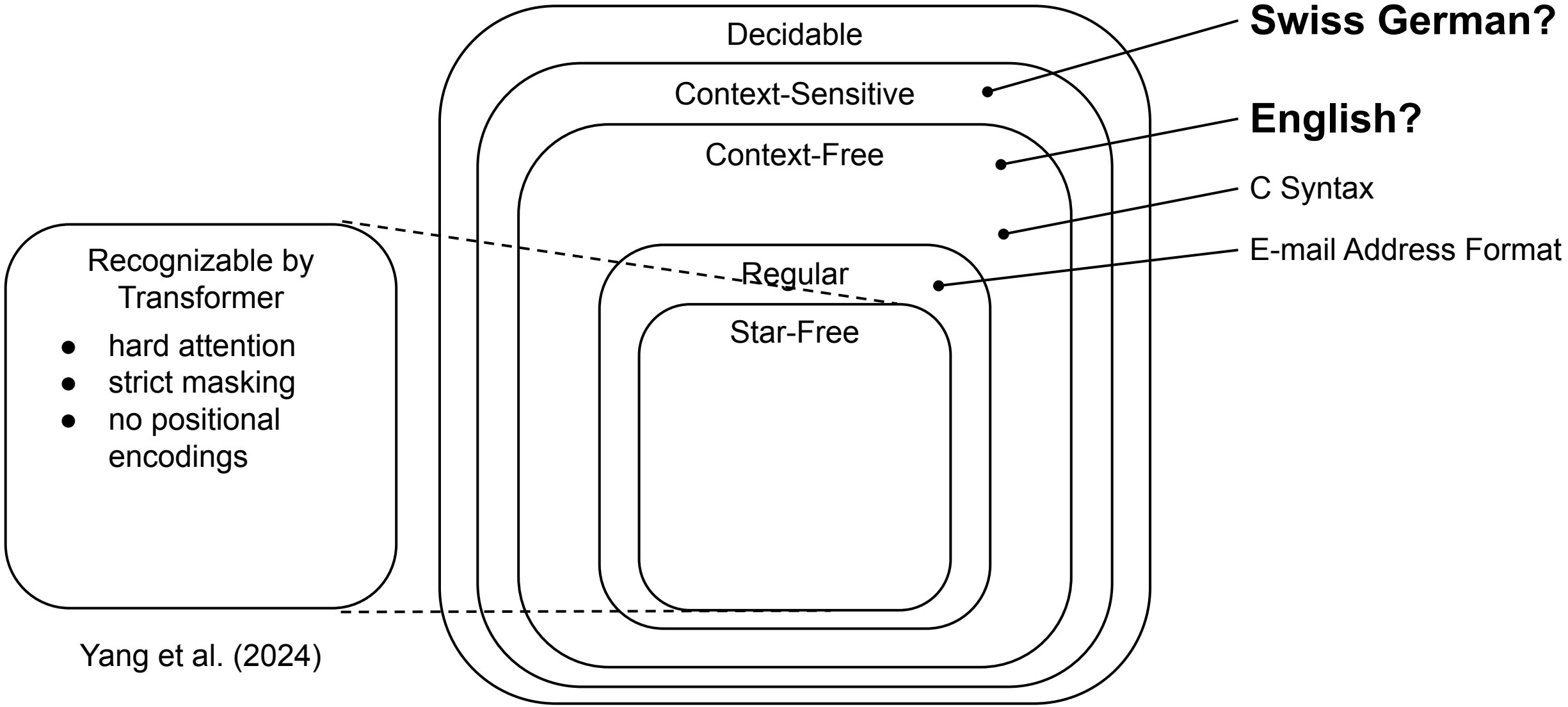


Chomsky Hierarchy



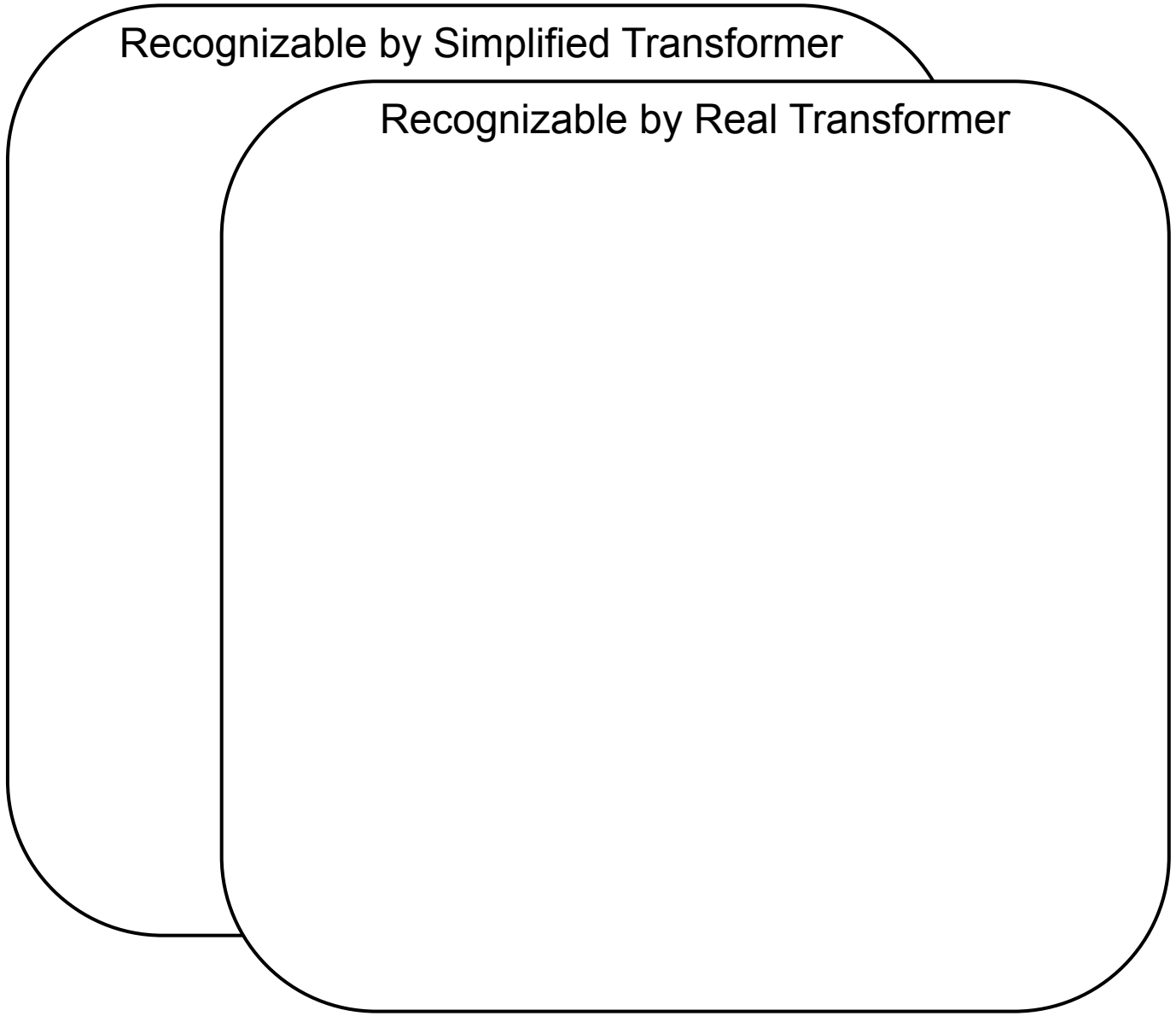
Yang et al. (2024)

Chomsky Hierarchy

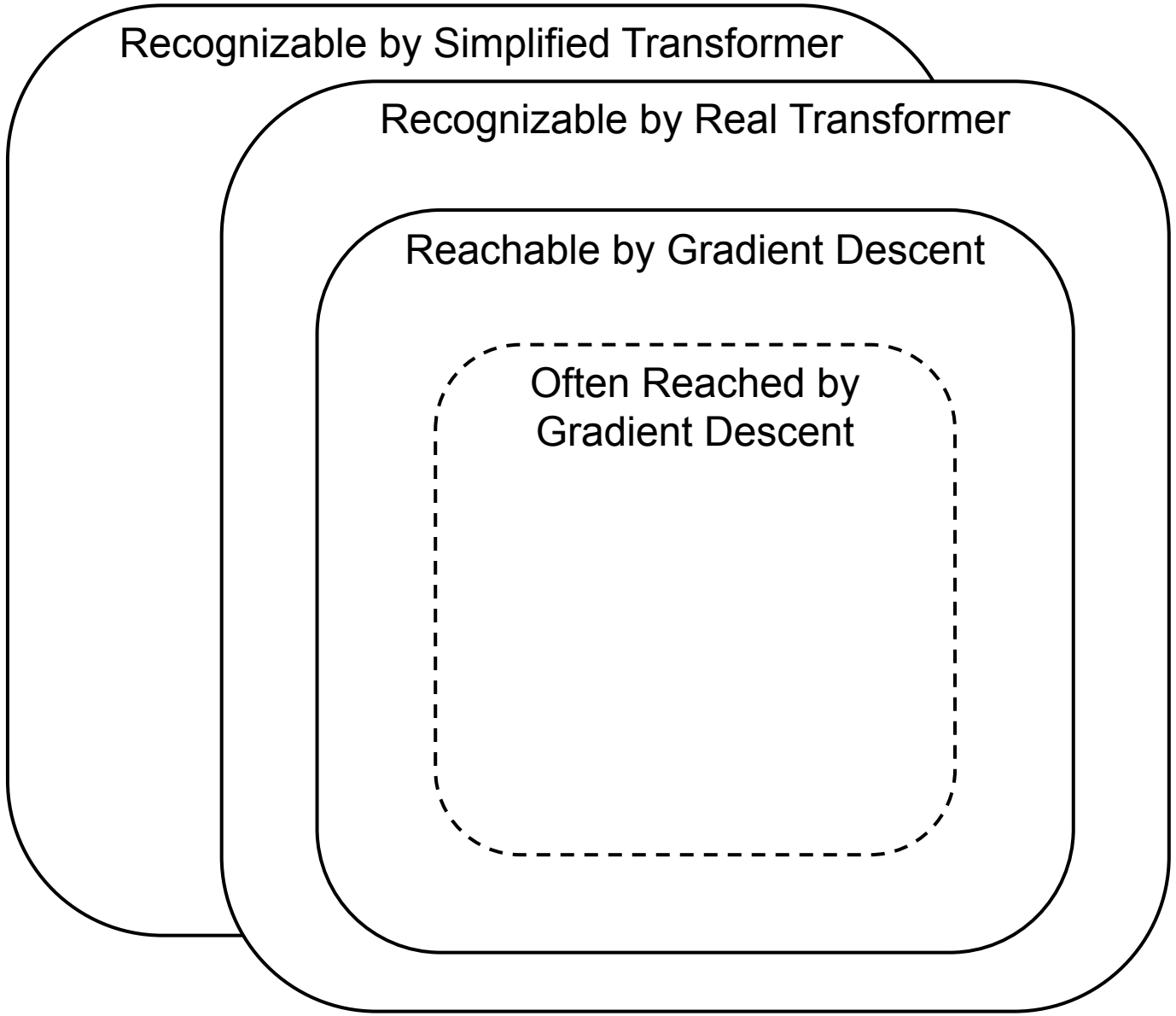


Yang et al. (2024)

Transformer Expressivity



Transformer Expressivity



Purpose of This Work

How Do We Test Hypotheses about Neural Networks as Recognizers?

Our Solution

Train Neural Networks as Recognizers (Binary Classifiers of Strings)

Basic Approach

- Get a dataset of strings labeled accept (1) or reject (0)
- The network outputs a probability of acceptance
- Use an objective function that maximizes the accept probability for accept and minimizes it for reject

Disconnect between Experiments and Theory

- Generating negative examples (strings labeled reject) is hard
- Positive-only tasks
  - Language modeling
  - Sequence-to-sequence transduction
- Can't convert a neural LM to a recognizer
  - How do you distinguish between probability of EOS that is **just close to 0** or **should be exactly 0**?

Published as a conference paper at ICLR 2023

NEURAL NETWORKS AND THE CHOMSKY HIERARCHY

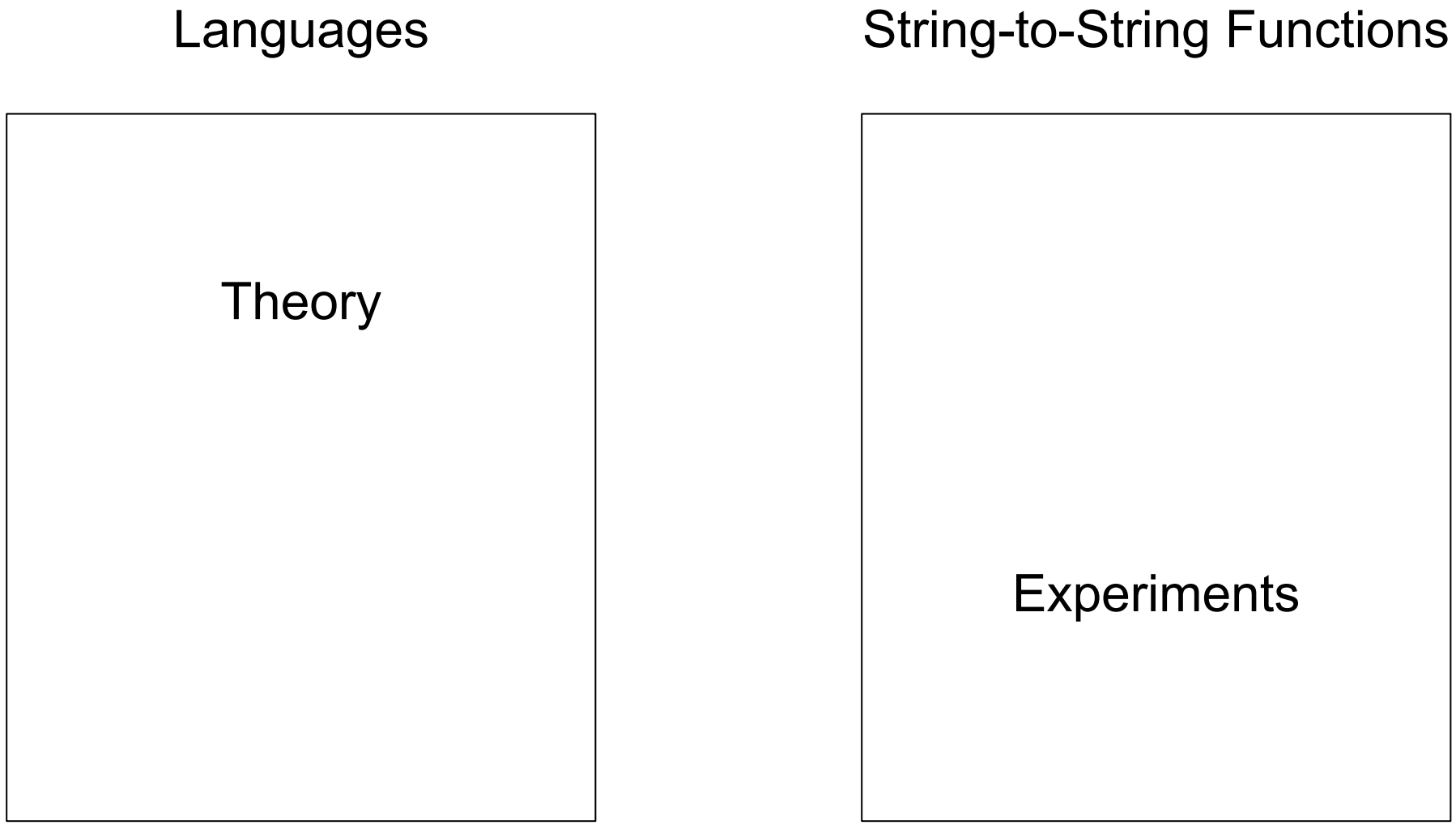
Grégoire Delétang[*1] **Anian Ruoss**[*1] **Jordi Grau-Moya**[1] **Tim Genewein**[1] **Li Kevin Wenliang**[1]

Elliot Catt[1] **Chris Cundy**[†2] **Marcus Hutter**[1] **Shane Legg**[1] **Joel Veness**[1] **Pedro A. Ortega**[†]

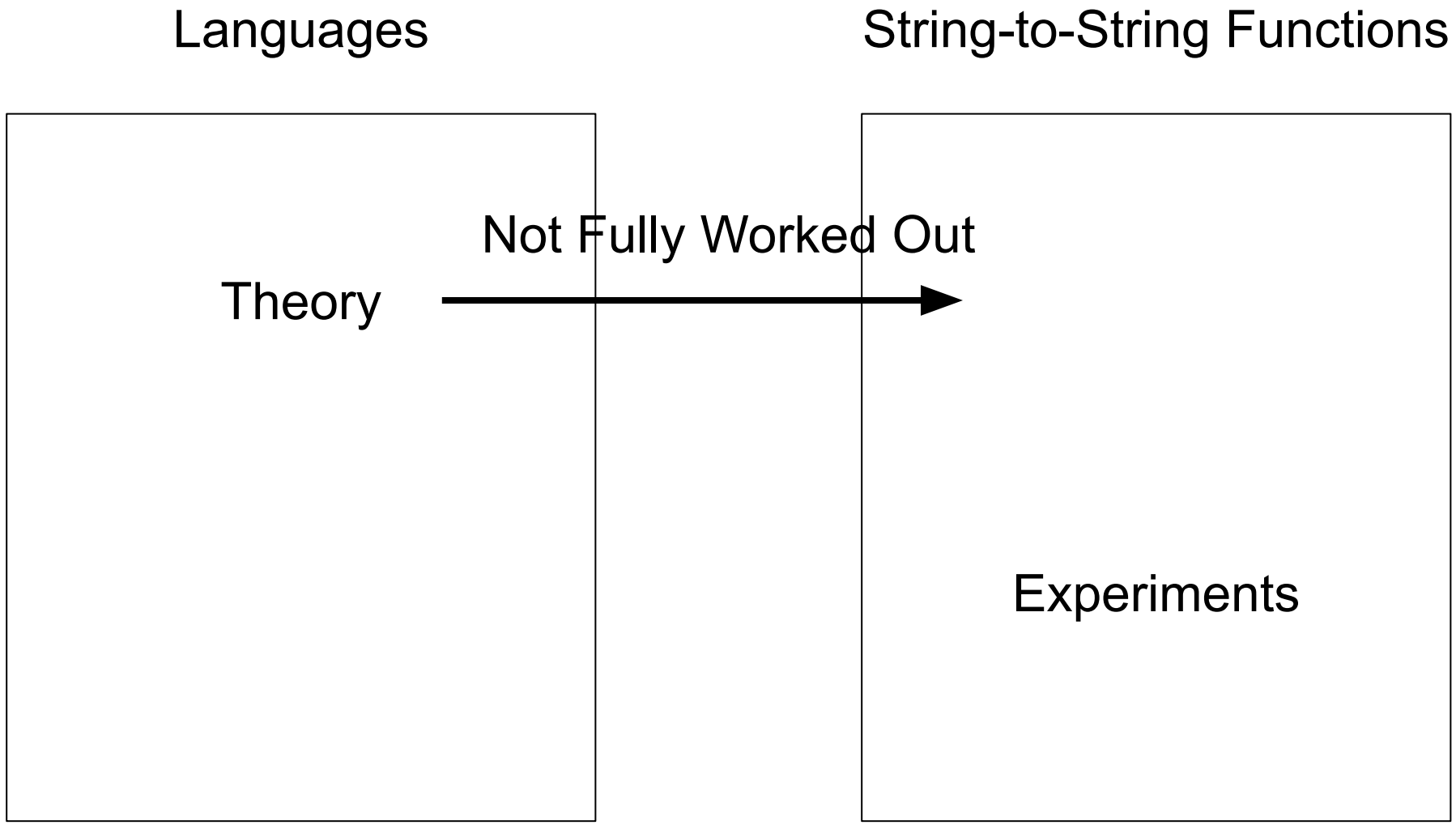
ABSTRACT

Reliable generalization lies at the heart of safe ML and AI. However, understanding when and how neural networks generalize remains one of the most important unsolved problems in the field. In this work, we conduct an extensive empirical study (20 910 models, 15 tasks) to investigate whether insights from the theory of computation can predict the limits of neural network generalization in practice. We demonstrate that grouping tasks according to the Chomsky hierarchy allows us to forecast whether certain architectures will be able to generalize to out-of-distribution inputs. This includes negative results where even extensive amounts of data and training time never lead to any non-trivial generalization, despite models having sufficient capacity to fit the training data perfectly. Our results

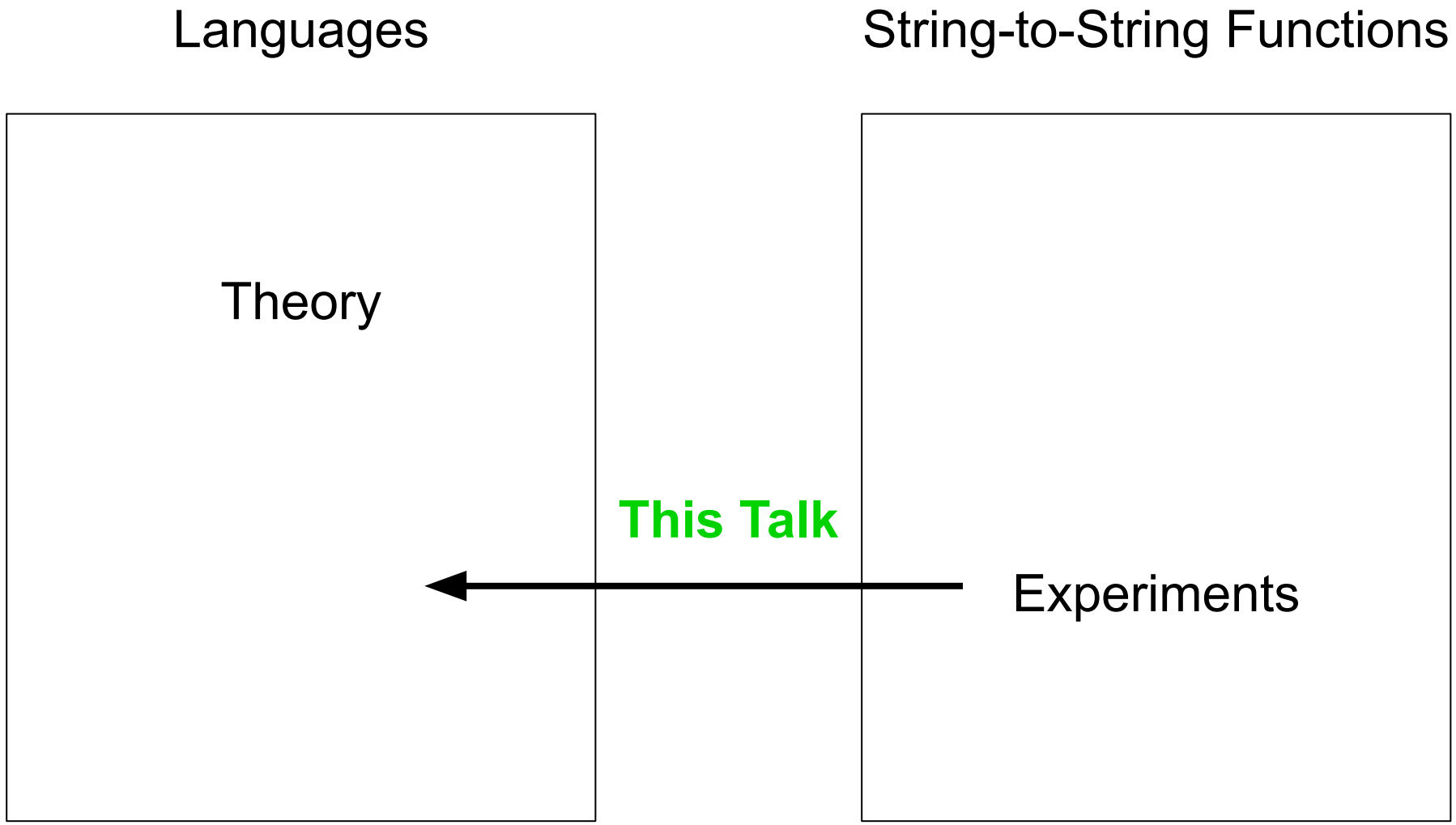
Bridging the Gap



Bridging the Gap



Bridging the Gap



Research Questions

- How powerful is each NN architecture?
- Do our results change if we do recognition instead of string-to-string?
- What training objectives work best?

Languages

Languages

Bigger Language Classes

Table 1: Formal languages tested in this paper and included in FLaRe. For each language, we show the language class that it belongs to: regular (**R**), deterministic context-free (**DCF**), context-free (**CF**), or context-sensitive (**CS**). Each language does not belong to the previous language classes. Let $c_u(w)$ be the number of times substring u occurs in w, let $w_{i \to a}$ be w with its i^{th} symbol replaced with a, and let $\langle x \rangle$ be the little-endian binary encoding of $x \in \mathbb{Z}_{\geq 0}$. See App. E for details.

Class	Language	Description	Example String
R	Even Pairs	$\{w \in \{0,1\}^* \mid c_{01}(w) + c_{10}(w) \text{ is even}\}$ $= \{aua \mid a \in \{0,1\}, u \in \{0,1\}^*\} \cup \{\varepsilon, 0, 1\}$	010110
	Repeat 01	$\{(01)^n \mid n \geq 0\}$	010101
	Parity	$\{w \in \{0,1\}^* \mid c_1(w) \text{ is odd}\}$	11011001
	Cycle Navigation	A sequence of left (<), right (>), stay (=) moves on a 5-position cycle, then the final position (0-indexed).	>>=<>2
	Modular Arithmetic	Expression involving $\{+, -, \times\}$ and $\{0, \ldots, 4\}$, then the result mod 5. No operator precedence.	1-3×2=1
	Dyck-$(2,3)$	Strings of balanced brackets with 2 bracket types and a maximum depth of 3.	[()([])]()
	First	$\{1w \mid w \in \{0,1\}^*\}$	100010
DCF	Majority	$\{w \in \{0,1\}^* \mid c_1(w) > c_0(w)\}$	101101
	Stack Manipulation	A stack from bottom to top, a sequence of push and pop operations, and the resulting stack from top to bottom.	011 POP =10
	Marked Reversal	$\{w\#w^R \mid w \in \{0,1\}^*\}$	001#100
CF	Unmarked Reversal	$\{ww^R \mid w \in \{0,1\}^*\}$	001100
CS	Marked Copy	$\{w\#w \mid w \in \{0,1\}^*\}$	001#001
	Missing Duplicate	$\{(ww)_{i \to \_} \mid w \in \{0,1\}^*, 1 \leq i \leq 2\lvert w\rvert, (ww)_i = 1\}$	1_011101
	Odds First	$\{a_1 b_1 \cdots a_n b_n a \# a_1 \cdots a_n a b_1 \cdots b_n \mid$ $n \geq 0; a_i, b_i \in \{0,1\}; a \in \{0,1,\varepsilon\}\}$	01010=00011
	Binary Addition	$\{\langle x \rangle 0^i + \langle y \rangle 0^j = \langle x+y \rangle 0^k \mid x, y, i, j, k \in \mathbb{Z}_{\geq 0}\}$	110+01=10100
	Binary Multiplication	$\{\langle x \rangle 0^i \times \langle y \rangle 0^j = \langle xy \rangle 0^k \mid x, y, i, j, k \in \mathbb{Z}_{\geq 0}\}$	110×0100=011
	Compute Sqrt	$\{\langle x \rangle 0^i = \langle \lfloor \sqrt{x} \rfloor \rangle 0^j \mid x, i, j \in \mathbb{Z}_{\geq 0}\}$	01010=1100
	Bucket Sort	Sequence of integers in $\{1, \ldots, 5\}$, then # and the sorted sequence.	45134#13445

Dataset Generation

Positive examples of w#w

✓ 0 1 0 1 1 # 0 1 0 1 1

✓ 1 1 0 # 1 1 0

✓ #

✓ 1 0 0 1 0 0 # 1 0 0 1 0 0

Negative Examples of w#w

✓ 0 1 0 1 1 # 0 1 0 1 1

✓ 1 1 0 # 1 1 0

✓ #

✓ 1 0 0 1 0 0 # 1 0 0 1 0 0

✗ # 0 # 1 0 # # 0 1 0 #

✗ 0 1 # 1 # 0 0

✗ # 1 1 # 0 # 0

~~0 1 # 0 1~~

Negative Examples of w#w

✓ 0 1 0 1 1 # 0 1 0 1 1

✓ 1 1 0 # 1 1 0

✓ #

✓ 1 0 0 1 0 0 # 1 0 0 1 0 0

✗ # 0 # 1 0 # # 0 1 0 #

✗ 0 1 # 1 # 0 0

✗ # 1 1 # 0 # 0

Negative Examples of w#w

✓ 0 1 0 1 1 # 0 1 0 1 1

✗ 0 1 0 1 1 # 0 0 0 1 1

✓ 1 1 0 # 1 1 0

✗ 1 1 0 0 # 1 1 0

✓ 1 0 0 1 0 0 # 1 0 0 1 0 0

✗ 1 0 0 1 0 # 1 0 0 1 0 0

Sampling Datasets

- Our method only needs two algorithms
  a. **Sample a positive string** with length in the range $[n_{min}, n_{max}]$
  b. **Membership testing** -- is a string in the language?
- Balanced label distribution: 50% positive, 50% negative
- Not specific to any language class

Negative Sampling

- Pick a length n uniformly from $[n_{min}, n_{max}]$
- Propose random strings, test whether they are in the language, reject positives
  - 50% chance: Generate a random string of symbols of length n
    - Tend to be too easy
  - 50% chance: Generate a positive example of length n, apply K random edits to it
    - Adversarial

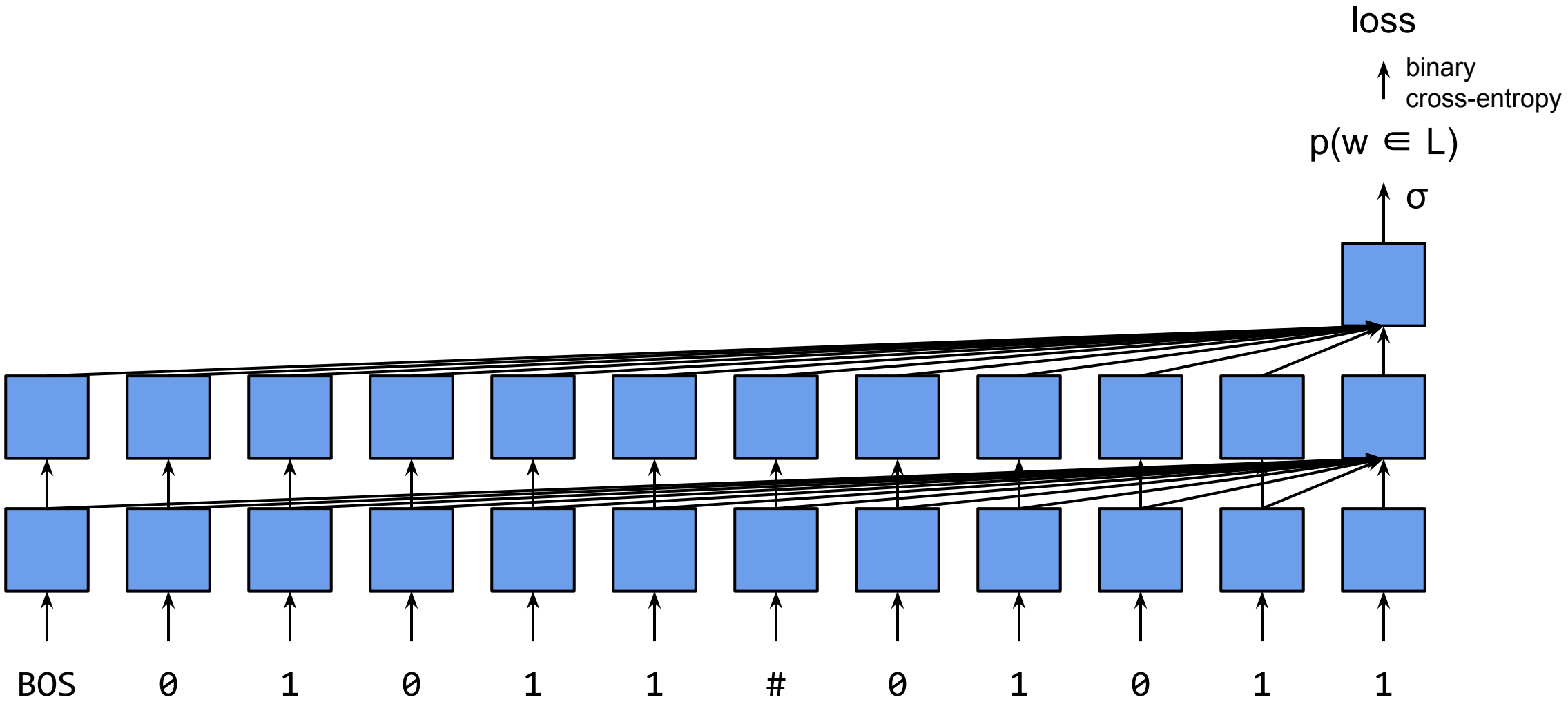
Architectures and Training Objectives

Three Architectures

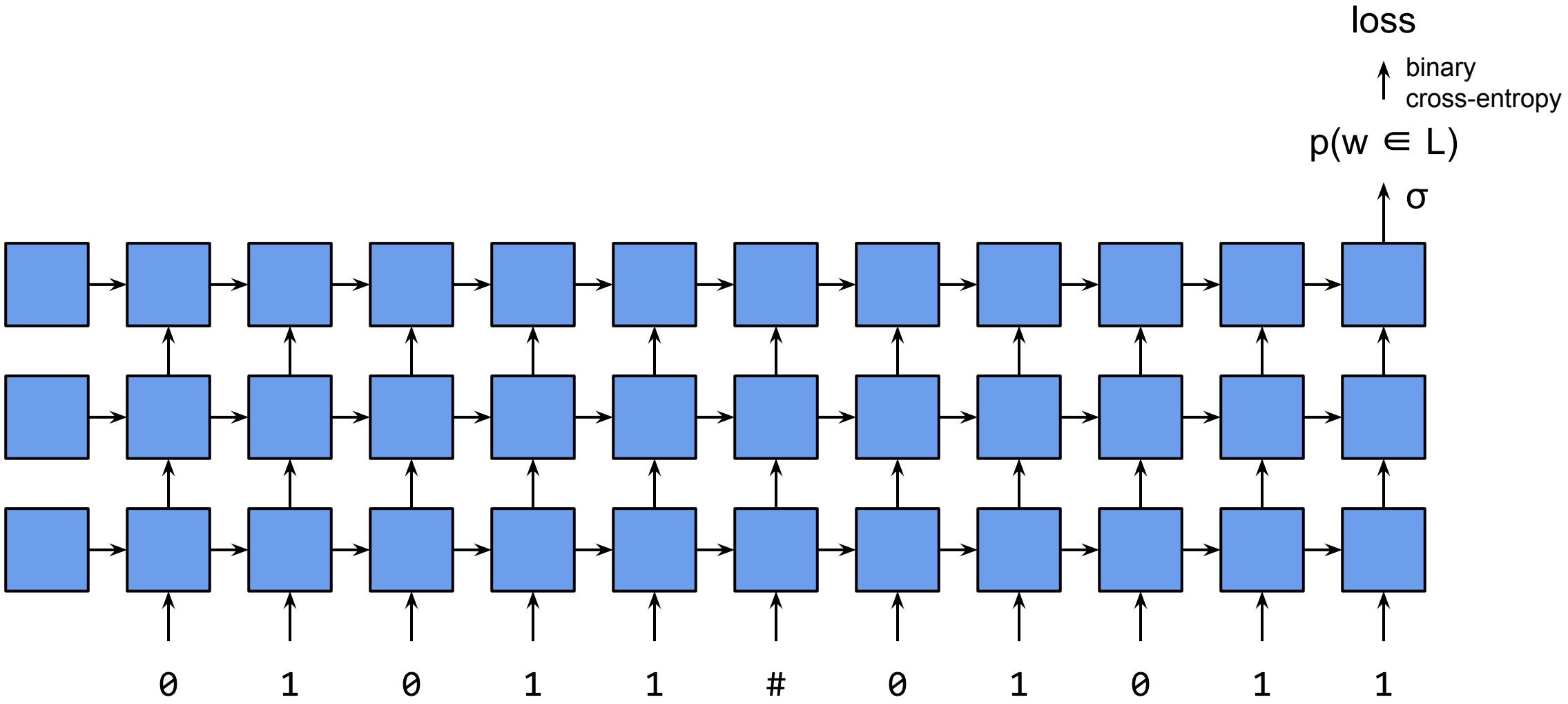
- Transformer
  - Causally masked (don't attend to later positions)
  - Sinusoidal positional encodings
  - Pre-norm instead of post-norm
- Simple RNN
  - tanh activation
  - learned initial state
- LSTM
  - decoupled input and forget gates
  - learned initial state

Every model has 5 layers and about 64k parameters.

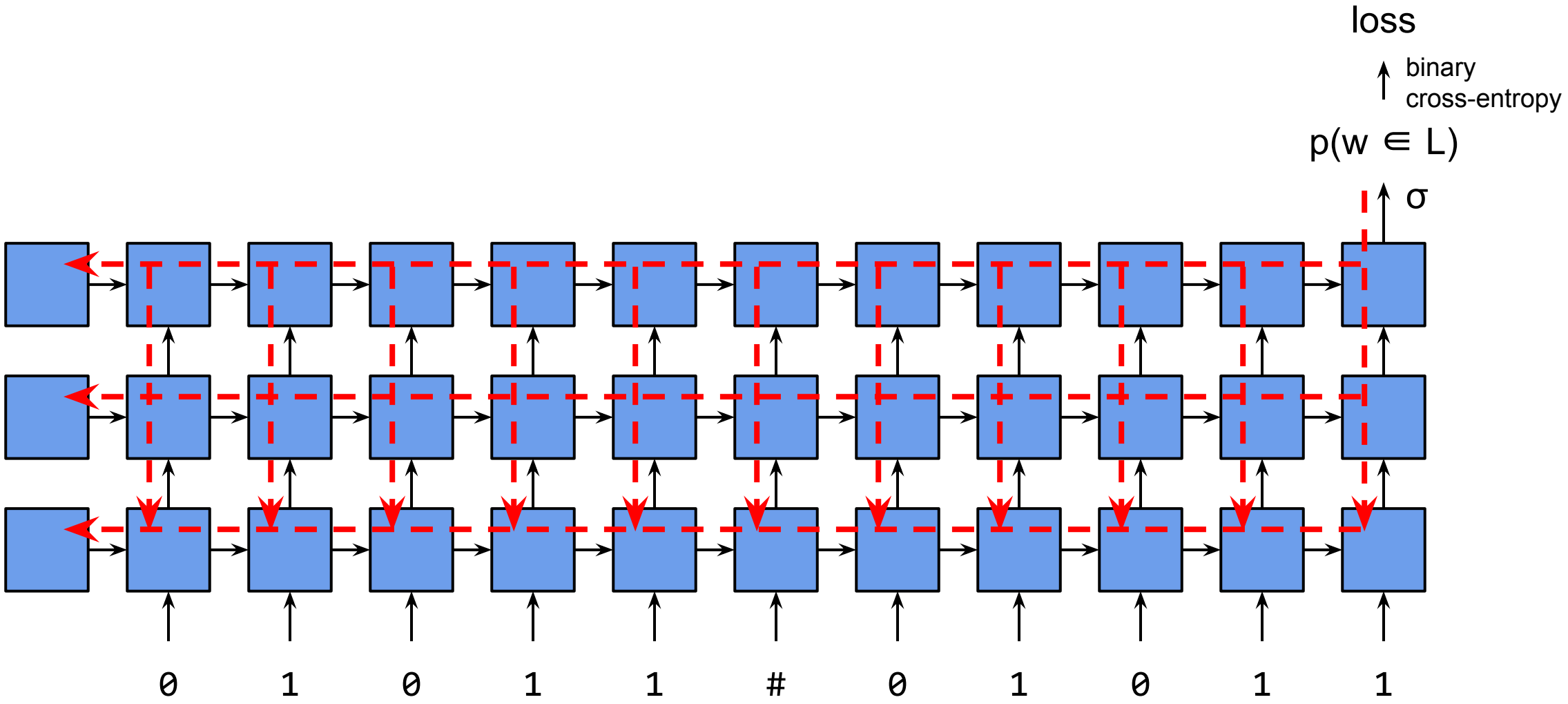
Transformer



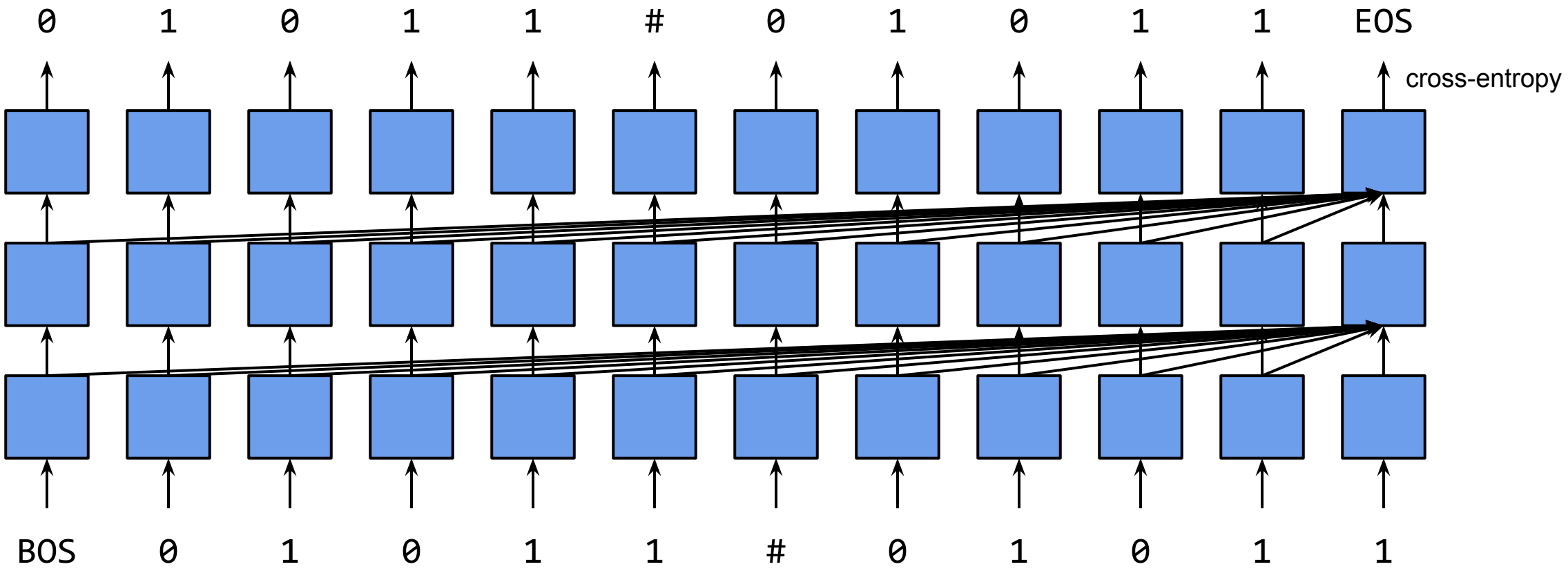
Simple RNN and LSTM



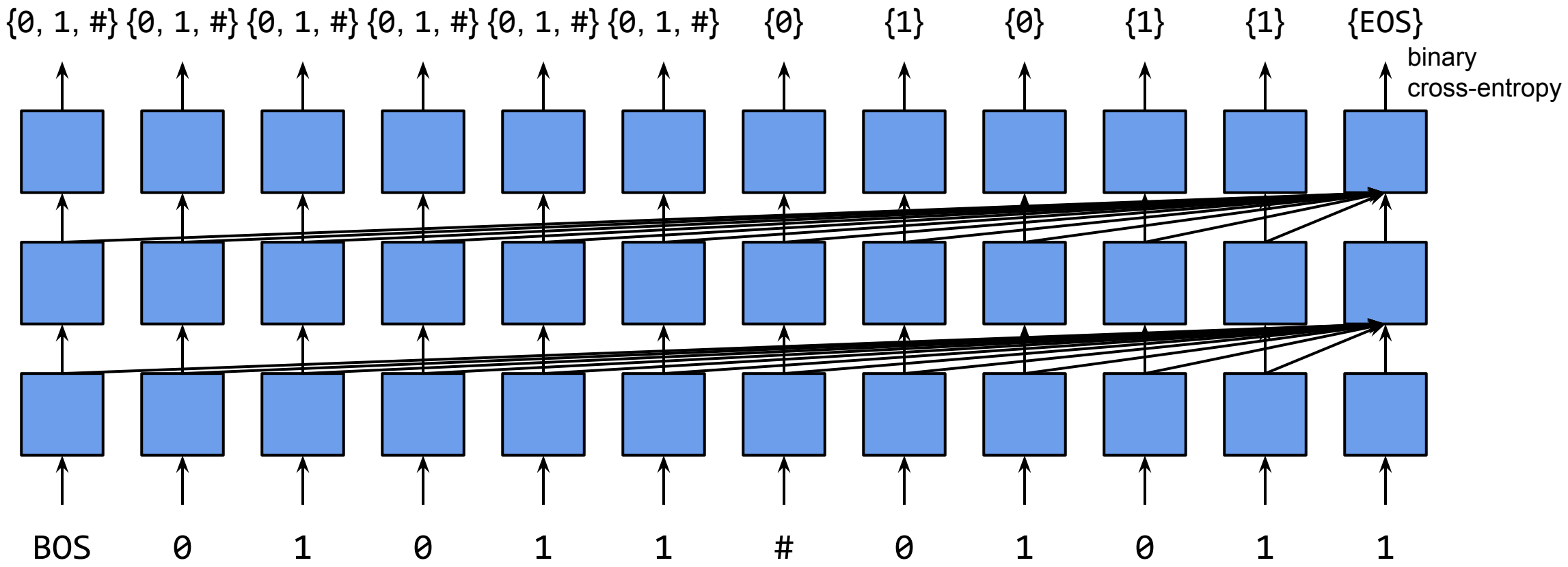
Gradient Problems



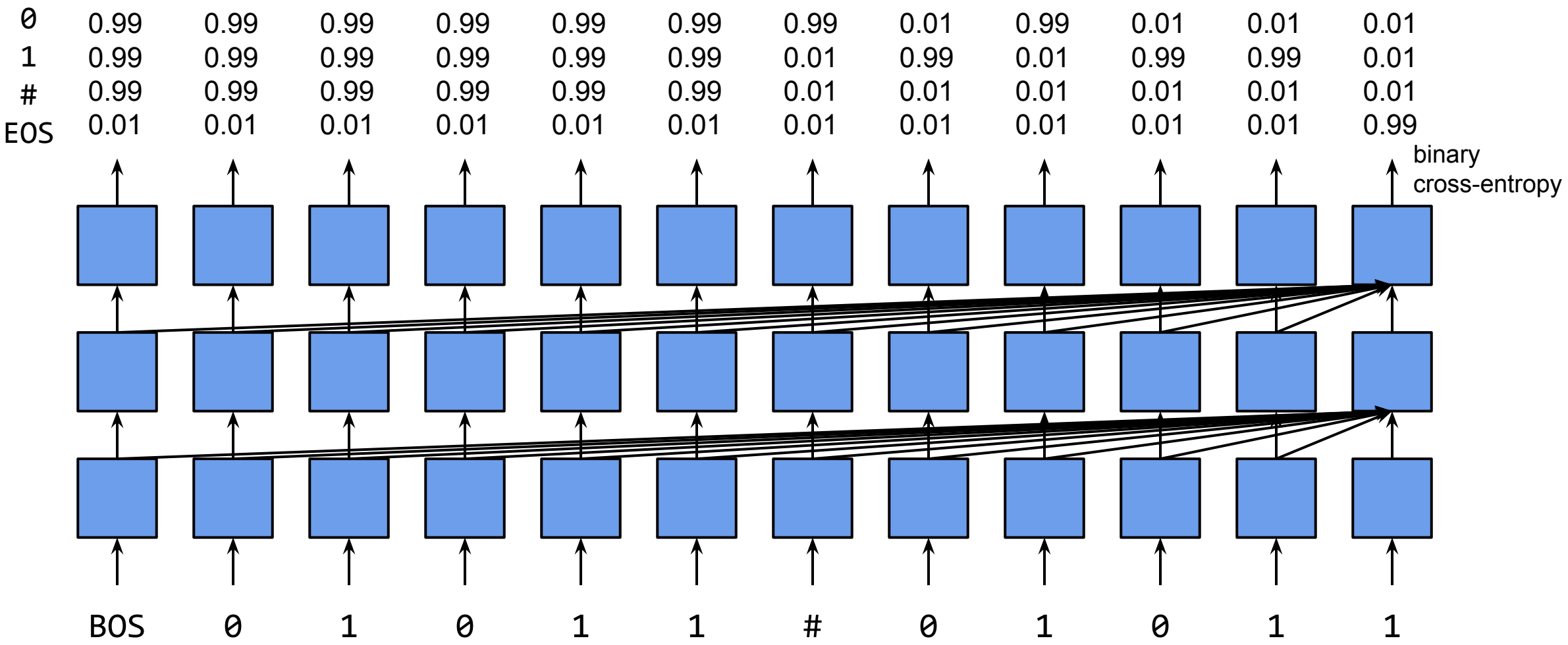
Language Modeling



Next Symbol Prediction



Next Symbol Prediction



Multi-Task Learning

$$\textbf{Loss} = \textbf{Rec. Loss} + \lambda_{LM} \times \textbf{LM Loss} + \lambda_{NS} \times \textbf{NS Loss}$$

- We always include the Rec. Loss term
- We test all 4 combinations of {with, without} × {LM Loss term, NS Loss term}

Experiments

Dataset Splits

- Training data: 10k examples with lengths in [0, 40]
- Test data: 5,010 examples with lengths in [0, 500]
  - Average of 10 examples per length
  - Emphasis on length generalization -- does it learn the algorithm?
- Two variations for validation data (next slides)

Testing Expressivity

- Does a parameter setting exist at all?
- Long validation data set: 1k examples with lengths in [0, 80]
- Encourages model to length-generalize during training via
  - learning rate schedule
  - early stopping
- Report **max** test accuracy of 10 random seeds among all 4 loss functions

Testing Inductive Bias

- What is the architecture's prior over languages?
- How does the network length-generalize without any hints during training?
- Short validation data: 1k examples with lengths in [0, 40]
- Report **mean** test accuracy of 10 random seeds of best loss function

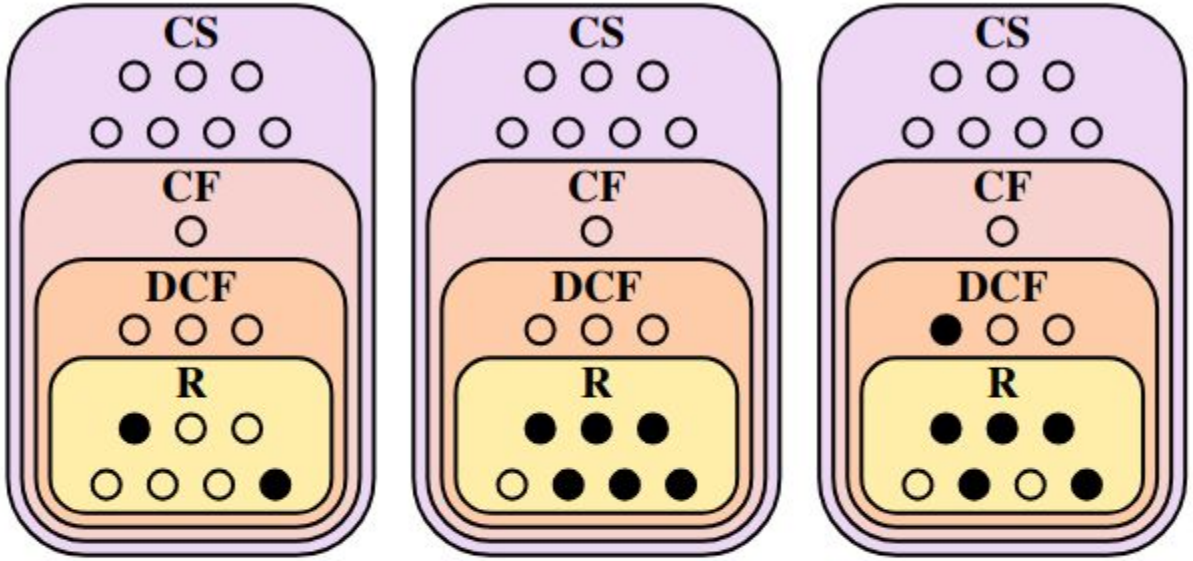
Results

RNN and LSTM outperform Transformer; RNN is surprisingly good

Bigger Language Classes

	Language	Inductive Bias Tf	Inductive Bias RNN	Inductive Bias LSTM	Expressivity Tf	Expressivity RNN	Expressivity LSTM
Regular	Even Pairs	**0.99**$_{\pm 0.01}$	0.60$_{\pm 0.20}$	0.83$_{\pm 0.22}$	**1.00**	**1.00**	**1.00**
	Repeat 01	0.72$_{\pm 0.09}$	0.97$_{\pm 0.07}$	**0.97**$_{\pm 0.07}$	0.86	**1.00**	**1.00**
	Parity	0.56$_{\pm 0.03}$	0.71$_{\pm 0.24}$	**0.90**$_{\pm 0.20}$	0.60	**1.00**	**1.00**
	Cycle Navigation	0.84$_{\pm 0.05}$	**0.93**$_{\pm 0.01}$	0.90$_{\pm 0.04}$	**0.93**	**0.93**	**0.93**
	Modular Arithmetic	0.69$_{\pm 0.11}$	**1.00**$_{\pm 0.00}$	0.98$_{\pm 0.03}$	0.88	**1.00**	**1.00**
	Dyck-$(2,3)$	0.70$_{\pm 0.09}$	**0.95**$_{\pm 0.05}$	0.91$_{\pm 0.10}$	0.82	**1.00**	0.98
	First	**0.98**$_{\pm 0.04}$	0.80$_{\pm 0.24}$	0.94$_{\pm 0.14}$	**1.00**	**1.00**	**1.00**
Deterministic Context-Free	Majority	**0.97**$_{\pm 0.04}$	0.90$_{\pm 0.03}$	0.95$_{\pm 0.04}$	1.00	0.95	**1.00**
	Stack Manipulation	0.66$_{\pm 0.14}$	**0.84**$_{\pm 0.16}$	0.75$_{\pm 0.17}$	0.87	**0.93**	0.91
	Marked Reversal	0.64$_{\pm 0.12}$	0.70$_{\pm 0.18}$	**0.74**$_{\pm 0.17}$	0.87	**0.95**	0.95
Context-Free	Unmarked Reversal	0.58$_{\pm 0.03}$	0.72$_{\pm 0.08}$	**0.76**$_{\pm 0.01}$	0.63	0.81	**0.88**
Context-Sensitive	Marked Copy	0.63$_{\pm 0.11}$	**0.76**$_{\pm 0.15}$	0.69$_{\pm 0.15}$	0.86	**0.95**	**0.95**
	Missing Duplicate	0.66$_{\pm 0.08}$	0.82$_{\pm 0.10}$	**0.85**$_{\pm 0.07}$	0.86	**0.95**	0.94
	Odds First	0.59$_{\pm 0.11}$	**0.79**$_{\pm 0.15}$	0.67$_{\pm 0.14}$	0.86	0.95	**0.96**
	Binary Addition	0.64$_{\pm 0.13}$	0.74$_{\pm 0.12}$	**0.74**$_{\pm 0.12}$	0.88	0.92	**0.92**
	Binary Multiplication	0.70$_{\pm 0.11}$	0.74$_{\pm 0.13}$	**0.78**$_{\pm 0.12}$	**0.92**	0.92	0.92
	Compute Sqrt	0.67$_{\pm 0.10}$	0.78$_{\pm 0.12}$	**0.84**$_{\pm 0.07}$	0.86	**0.89**	**0.89**
	Bucket Sort	0.63$_{\pm 0.08}$	**0.84**$_{\pm 0.09}$	0.69$_{\pm 0.13}$	0.88	**0.96**	0.83

Expressivity Results



(a) Transformer (b) Simple RNN (c) LSTM

Figure 1: Summary of our empirical expressivity results. Dots represent languages, which are listed in Table 1. A filled dot means that the architecture exhibits perfect length generalization (see Table 2 under "Expressivity"). R = regular, DCF = deterministic context-free, CF = context-free, CS = context-sensitive. All architectures are limited to regular languages and the DCF language Majority. The transformer is strictly less expressive than the RNN/LSTM on the languages we tested.

Results

Consistency in rankings between Inductive Bias and Expressivity

Bigger Language Classes

	Language	Inductive Bias Tf	Inductive Bias RNN	Inductive Bias LSTM	Expressivity Tf	Expressivity RNN	Expressivity LSTM
Regular	Even Pairs	**0.99**$_{\pm 0.01}$	0.60$_{\pm 0.20}$	0.83$_{\pm 0.22}$	**1.00**	**1.00**	**1.00**
	Repeat 01	0.72$_{\pm 0.09}$	0.97$_{\pm 0.07}$	**0.97**$_{\pm 0.07}$	0.86	**1.00**	**1.00**
	Parity	0.56$_{\pm 0.03}$	0.71$_{\pm 0.24}$	**0.90**$_{\pm 0.20}$	0.60	**1.00**	**1.00**
	Cycle Navigation	0.84$_{\pm 0.05}$	**0.93**$_{\pm 0.01}$	0.90$_{\pm 0.04}$	**0.93**	**0.93**	**0.93**
	Modular Arithmetic	0.69$_{\pm 0.11}$	**1.00**$_{\pm 0.00}$	0.98$_{\pm 0.03}$	0.88	**1.00**	**1.00**
	Dyck-(2, 3)	0.70$_{\pm 0.09}$	**0.95**$_{\pm 0.05}$	0.91$_{\pm 0.10}$	0.82	**1.00**	0.98
	First	**0.98**$_{\pm 0.04}$	0.80$_{\pm 0.24}$	0.94$_{\pm 0.14}$	**1.00**	**1.00**	**1.00**
Deterministic Context-Free	Majority	**0.97**$_{\pm 0.04}$	0.90$_{\pm 0.03}$	0.95$_{\pm 0.04}$	1.00	0.95	**1.00**
	Stack Manipulation	0.66$_{\pm 0.14}$	**0.84**$_{\pm 0.16}$	0.75$_{\pm 0.17}$	0.87	**0.93**	0.91
	Marked Reversal	0.64$_{\pm 0.12}$	0.70$_{\pm 0.18}$	**0.74**$_{\pm 0.17}$	0.87	**0.95**	0.95
Context-Free	Unmarked Reversal	0.58$_{\pm 0.03}$	0.72$_{\pm 0.08}$	**0.76**$_{\pm 0.01}$	0.63	0.81	**0.88**
	Marked Copy	0.63$_{\pm 0.11}$	**0.76**$_{\pm 0.15}$	0.69$_{\pm 0.15}$	0.86	**0.95**	**0.95**
Context-Sensitive	Missing Duplicate	0.66$_{\pm 0.08}$	0.82$_{\pm 0.10}$	**0.85**$_{\pm 0.07}$	0.86	**0.95**	0.94
	Odds First	0.59$_{\pm 0.11}$	**0.79**$_{\pm 0.15}$	0.67$_{\pm 0.14}$	0.86	0.95	**0.96**
	Binary Addition	0.64$_{\pm 0.13}$	0.74$_{\pm 0.12}$	**0.74**$_{\pm 0.12}$	0.88	0.92	**0.92**
	Binary Multiplication	0.70$_{\pm 0.11}$	0.74$_{\pm 0.13}$	**0.78**$_{\pm 0.12}$	**0.92**	0.92	0.92
	Compute Sqrt	0.67$_{\pm 0.10}$	0.78$_{\pm 0.12}$	**0.84**$_{\pm 0.07}$	0.86	**0.89**	**0.89**
	Bucket Sort	0.63$_{\pm 0.08}$	**0.84**$_{\pm 0.09}$	0.69$_{\pm 0.13}$	0.88	**0.96**	0.83

Results

Transformers do well on low-sensitivity and badly on high-sensitivity (Hahn & Rofin, 2024)

Bigger Language Classes

	Language	Inductive Bias			Expressivity		
		Tf	RNN	LSTM	Tf	RNN	LSTM
Regular	Even Pairs	**0.99**$_{\pm 0.01}$	0.60$_{\pm 0.20}$	0.83$_{\pm 0.22}$	**1.00**	**1.00**	**1.00**
	Repeat 01	0.72$_{\pm 0.09}$	0.97$_{\pm 0.07}$	**0.97**$_{\pm 0.07}$	0.86	**1.00**	**1.00**
	Parity	0.56$_{\pm 0.03}$	0.71$_{\pm 0.24}$	**0.90**$_{\pm 0.20}$	0.60	**1.00**	**1.00**
	Cycle Navigation	0.84$_{\pm 0.05}$	**0.93**$_{\pm 0.01}$	0.90$_{\pm 0.04}$	**0.93**	**0.93**	**0.93**
	Modular Arithmetic	0.69$_{\pm 0.11}$	**1.00**$_{\pm 0.00}$	0.98$_{\pm 0.03}$	0.88	**1.00**	**1.00**
	Dyck-(2, 3)	0.70$_{\pm 0.09}$	**0.95**$_{\pm 0.05}$	0.91$_{\pm 0.10}$	0.82	**1.00**	0.98
	First	**0.98**$_{\pm 0.04}$	0.80$_{\pm 0.24}$	0.94$_{\pm 0.14}$	**1.00**	**1.00**	**1.00**
Deterministic Context-Free	Majority	**0.97**$_{\pm 0.04}$	0.90$_{\pm 0.03}$	0.95$_{\pm 0.04}$	1.00	0.95	**1.00**
	Stack Manipulation	0.66$_{\pm 0.14}$	**0.84**$_{\pm 0.16}$	0.75$_{\pm 0.17}$	0.87	**0.93**	0.91
	Marked Reversal	0.64$_{\pm 0.12}$	0.70$_{\pm 0.18}$	**0.74**$_{\pm 0.17}$	0.87	**0.95**	0.95
Context-Free	Unmarked Reversal	0.58$_{\pm 0.03}$	0.72$_{\pm 0.08}$	**0.76**$_{\pm 0.01}$	0.63	0.81	**0.88**
Context-Sensitive	Marked Copy	0.63$_{\pm 0.11}$	**0.76**$_{\pm 0.15}$	0.69$_{\pm 0.15}$	0.86	**0.95**	**0.95**
	Missing Duplicate	0.66$_{\pm 0.08}$	0.82$_{\pm 0.10}$	**0.85**$_{\pm 0.07}$	0.86	**0.95**	0.94
	Odds First	0.59$_{\pm 0.11}$	**0.79**$_{\pm 0.15}$	0.67$_{\pm 0.14}$	0.86	0.95	**0.96**
	Binary Addition	0.64$_{\pm 0.13}$	0.74$_{\pm 0.12}$	**0.74**$_{\pm 0.12}$	0.88	0.92	**0.92**
	Binary Multiplication	0.70$_{\pm 0.11}$	0.74$_{\pm 0.13}$	**0.78**$_{\pm 0.12}$	**0.92**	0.92	0.92
	Compute Sqrt	0.67$_{\pm 0.10}$	0.78$_{\pm 0.12}$	**0.84**$_{\pm 0.07}$	0.86	**0.89**	**0.89**
	Bucket Sort	0.63$_{\pm 0.08}$	**0.84**$_{\pm 0.09}$	0.69$_{\pm 0.13}$	0.88	**0.96**	0.83

Low-Sensitivity

Even Pairs (begins and ends with same bit)

0 1 0 1 1 0

1 0 1 1

First (begins with 1)

1 0 0 1 0 1

1 1 0 1 1

1 0

High-Sensitivity

Repeat 01 (0 1 repeated any number of times)

0 1 0 1 0 1 0 1

0 1 0 1

Parity (odd number of 1's)

0 1 0 1 1

1 0 1 1

0 0 0 1

Results

Transformers do well on low-sensitivity and badly on high-sensitivity (Hahn & Rofin, 2024)

Bigger Language Classes

	Language	Inductive Bias Tf	Inductive Bias RNN	Inductive Bias LSTM	Expressivity Tf	Expressivity RNN	Expressivity LSTM
Regular	Even Pairs	**0.99**$_{\pm 0.01}$	0.60$_{\pm 0.20}$	0.83$_{\pm 0.22}$	**1.00**	**1.00**	**1.00**
	Repeat 01	0.72$_{\pm 0.09}$	0.97$_{\pm 0.07}$	**0.97**$_{\pm 0.07}$	0.86	**1.00**	**1.00**
	Parity	0.56$_{\pm 0.03}$	0.71$_{\pm 0.24}$	**0.90**$_{\pm 0.20}$	0.60	**1.00**	**1.00**
	Cycle Navigation	0.84$_{\pm 0.05}$	**0.93**$_{\pm 0.01}$	0.90$_{\pm 0.04}$	**0.93**	**0.93**	**0.93**
	Modular Arithmetic	0.69$_{\pm 0.11}$	**1.00**$_{\pm 0.00}$	0.98$_{\pm 0.03}$	0.88	**1.00**	**1.00**
	Dyck-$(2,3)$	0.70$_{\pm 0.09}$	**0.95**$_{\pm 0.05}$	0.91$_{\pm 0.10}$	0.82	**1.00**	0.98
	First	**0.98**$_{\pm 0.04}$	0.80$_{\pm 0.24}$	0.94$_{\pm 0.14}$	**1.00**	**1.00**	**1.00**
Deterministic Context-Free	Majority	**0.97**$_{\pm 0.04}$	0.90$_{\pm 0.03}$	0.95$_{\pm 0.04}$	1.00	0.95	**1.00**
	Stack Manipulation	0.66$_{\pm 0.14}$	**0.84**$_{\pm 0.16}$	0.75$_{\pm 0.17}$	0.87	**0.93**	0.91
	Marked Reversal	0.64$_{\pm 0.12}$	0.70$_{\pm 0.18}$	**0.74**$_{\pm 0.17}$	0.87	**0.95**	0.95
Context-Free	Unmarked Reversal	0.58$_{\pm 0.03}$	0.72$_{\pm 0.08}$	**0.76**$_{\pm 0.01}$	0.63	0.81	**0.88**
Context-Sensitive	Marked Copy	0.63$_{\pm 0.11}$	**0.76**$_{\pm 0.15}$	0.69$_{\pm 0.15}$	0.86	**0.95**	**0.95**
	Missing Duplicate	0.66$_{\pm 0.08}$	0.82$_{\pm 0.10}$	**0.85**$_{\pm 0.07}$	0.86	**0.95**	0.94
	Odds First	0.59$_{\pm 0.11}$	**0.79**$_{\pm 0.15}$	0.67$_{\pm 0.14}$	0.86	0.95	**0.96**
	Binary Addition	0.64$_{\pm 0.13}$	0.74$_{\pm 0.12}$	**0.74**$_{\pm 0.12}$	0.88	0.92	**0.92**
	Binary Multiplication	0.70$_{\pm 0.11}$	0.74$_{\pm 0.13}$	**0.78**$_{\pm 0.12}$	**0.92**	0.92	0.92
	Compute Sqrt	0.67$_{\pm 0.10}$	0.78$_{\pm 0.12}$	**0.84**$_{\pm 0.07}$	0.86	**0.89**	**0.89**
	Bucket Sort	0.63$_{\pm 0.08}$	**0.84**$_{\pm 0.09}$	0.69$_{\pm 0.13}$	0.88	**0.96**	0.83

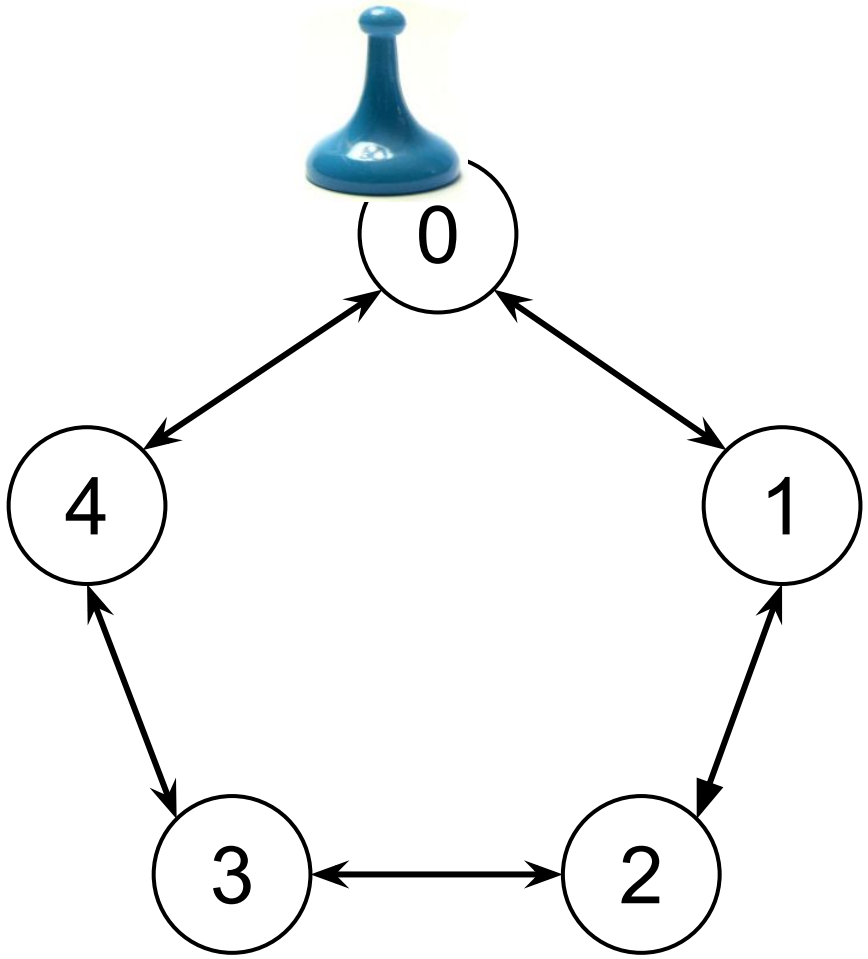
Results

Accuracy ceilings

Bigger Language Classes

	Language	Inductive Bias Tf	Inductive Bias RNN	Inductive Bias LSTM	Expressivity Tf	Expressivity RNN	Expressivity LSTM
Regular	Even Pairs	**0.99**$_{\pm 0.01}$	0.60$_{\pm 0.20}$	0.83$_{\pm 0.22}$	**1.00**	**1.00**	**1.00**
	Repeat 01	0.72$_{\pm 0.09}$	0.97$_{\pm 0.07}$	**0.97**$_{\pm 0.07}$	0.86	**1.00**	**1.00**
	Parity	0.56$_{\pm 0.03}$	0.71$_{\pm 0.24}$	**0.90**$_{\pm 0.20}$	0.60	**1.00**	**1.00**
	Cycle Navigation	0.84$_{\pm 0.05}$	**0.93**$_{\pm 0.01}$	0.90$_{\pm 0.04}$	**0.93**	**0.93**	**0.93**
	Modular Arithmetic	0.69$_{\pm 0.11}$	**1.00**$_{\pm 0.00}$	0.98$_{\pm 0.03}$	0.88	**1.00**	**1.00**
	Dyck-(2, 3)	0.70$_{\pm 0.09}$	**0.95**$_{\pm 0.05}$	0.91$_{\pm 0.10}$	0.82	**1.00**	0.98
	First	**0.98**$_{\pm 0.04}$	0.80$_{\pm 0.24}$	0.94$_{\pm 0.14}$	**1.00**	**1.00**	**1.00**
Deterministic Context-Free	Majority	**0.97**$_{\pm 0.04}$	0.90$_{\pm 0.03}$	0.95$_{\pm 0.04}$	1.00	0.95	**1.00**
	Stack Manipulation	0.66$_{\pm 0.14}$	**0.84**$_{\pm 0.16}$	0.75$_{\pm 0.17}$	0.87	**0.93**	0.91
	Marked Reversal	0.64$_{\pm 0.12}$	0.70$_{\pm 0.18}$	**0.74**$_{\pm 0.17}$	0.87	**0.95**	0.95
Context-Free	Unmarked Reversal	0.58$_{\pm 0.03}$	0.72$_{\pm 0.08}$	**0.76**$_{\pm 0.01}$	0.63	0.81	**0.88**
Context-Sensitive	Marked Copy	0.63$_{\pm 0.11}$	**0.76**$_{\pm 0.15}$	0.69$_{\pm 0.15}$	0.86	**0.95**	**0.95**
	Missing Duplicate	0.66$_{\pm 0.08}$	0.82$_{\pm 0.10}$	**0.85**$_{\pm 0.07}$	0.86	**0.95**	0.94
	Odds First	0.59$_{\pm 0.11}$	**0.79**$_{\pm 0.15}$	0.67$_{\pm 0.14}$	0.86	0.95	**0.96**
	Binary Addition	0.64$_{\pm 0.13}$	0.74$_{\pm 0.12}$	**0.74**$_{\pm 0.12}$	0.88	0.92	**0.92**
	Binary Multiplication	0.70$_{\pm 0.11}$	0.74$_{\pm 0.13}$	**0.78**$_{\pm 0.12}$	**0.92**	0.92	0.92
	Compute Sqrt	0.67$_{\pm 0.10}$	0.78$_{\pm 0.12}$	**0.84**$_{\pm 0.07}$	0.86	**0.89**	**0.89**
	Bucket Sort	0.63$_{\pm 0.08}$	**0.84**$_{\pm 0.09}$	0.69$_{\pm 0.13}$	0.88	**0.96**	0.83

Cycle Navigation



Cycle Navigation

✓ > > > = > < = = > = > > > < < > < > > = < > < = > > < = < = < > < = 0

✗ = > > 2 1

✓ > < = = > = = = < < = > = 0

✗ > = = = > > < < = = = < < > < 3

✓ = > < < < = < = < > < > > = > > = = > > < < > < < 4

✓ < = = > < > = > = > < < = < > = > < = < < = > > = = < > = < < < 2

✗ 1 > 4 0 2 2 1 0 3 4 0 2 > 1 4 > = 4 4 0 3 > < = 2 4 4 =

✗ > = 2 > 1 0 0 = < > 2 1 < 3 4 3 2 0 = 0 4 3 < 2 4 2 < 0 =

✗ < = < < > > < 0 < 2

✓ < > = < > = = < < 3

Cycle Navigation

- Increasing model size doesn't help
- Increasing training data size doesn't help
- Increasing adversarial negative examples doesn't help

Binary Arithmetic

Small digits come first

Addition

0 0 1 0 + 1 1 0 = 1 1 1 0

Multiplication

0 1 0 0 * 0 0 1 = 0 0 0 1

Square Root

0 0 1 = 0 1

Recognition (Ours) vs. Seq-to-Seq (DeepMind)

- Transformer struggles on Parity
- Their RNN/LSTM solved Cycle Navigation, ours did not
- Our RNN performs better than theirs (slightly different architectures)

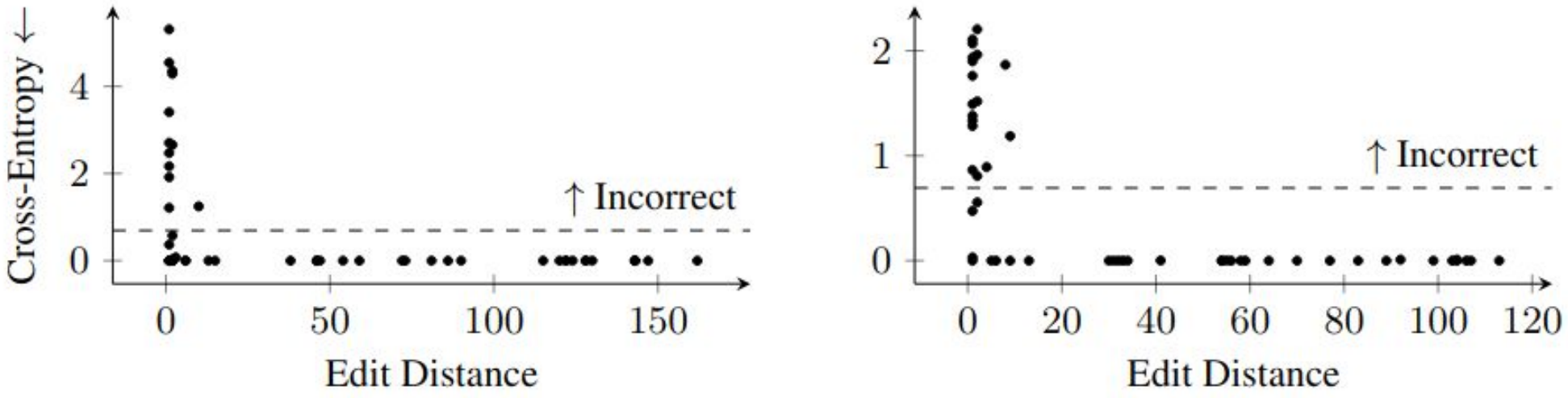
Recognition (Ours) vs. Seq-to-Seq (DeepMind)

Different model rankings on many languages

Bigger Language Classes

		Inductive Bias			Expressivity		
	Language	Tf	RNN	LSTM	Tf	RNN	LSTM
Regular	Even Pairs	**0.99**$_{\pm 0.01}$	0.60$_{\pm 0.20}$	0.83$_{\pm 0.22}$	**1.00**	**1.00**	**1.00**
	Repeat 01	0.72$_{\pm 0.09}$	0.97$_{\pm 0.07}$	**0.97**$_{\pm 0.07}$	0.86	**1.00**	**1.00**
	Parity	0.56$_{\pm 0.03}$	0.71$_{\pm 0.24}$	**0.90**$_{\pm 0.20}$	0.60	**1.00**	**1.00**
	Cycle Navigation	0.84$_{\pm 0.05}$	**0.93**$_{\pm 0.01}$	0.90$_{\pm 0.04}$	**0.93**	**0.93**	**0.93**
	Modular Arithmetic	0.69$_{\pm 0.11}$	**1.00**$_{\pm 0.00}$	0.98$_{\pm 0.03}$	0.88	**1.00**	**1.00**
	Dyck-(2, 3)	0.70$_{\pm 0.09}$	**0.95**$_{\pm 0.05}$	0.91$_{\pm 0.10}$	0.82	**1.00**	0.98
	First	**0.98**$_{\pm 0.04}$	0.80$_{\pm 0.24}$	0.94$_{\pm 0.14}$	**1.00**	**1.00**	**1.00**
Deterministic Context-Free	Majority	**0.97**$_{\pm 0.04}$	0.90$_{\pm 0.03}$	0.95$_{\pm 0.04}$	1.00	0.95	**1.00**
	Stack Manipulation	0.66$_{\pm 0.14}$	**0.84**$_{\pm 0.16}$	0.75$_{\pm 0.17}$	0.87	**0.93**	0.91
	Marked Reversal	0.64$_{\pm 0.12}$	0.70$_{\pm 0.18}$	**0.74**$_{\pm 0.17}$	0.87	**0.95**	0.95
Context-Free	Unmarked Reversal	0.58$_{\pm 0.03}$	0.72$_{\pm 0.08}$	**0.76**$_{\pm 0.01}$	0.63	0.81	**0.88**
Context-Sensitive	Marked Copy	0.63$_{\pm 0.11}$	**0.76**$_{\pm 0.15}$	0.69$_{\pm 0.15}$	0.86	**0.95**	**0.95**
	Missing Duplicate	0.66$_{\pm 0.08}$	0.82$_{\pm 0.10}$	**0.85**$_{\pm 0.07}$	0.86	**0.95**	0.94
	Odds First	0.59$_{\pm 0.11}$	**0.79**$_{\pm 0.15}$	0.67$_{\pm 0.14}$	0.86	0.95	**0.96**
	Binary Addition	0.64$_{\pm 0.13}$	0.74$_{\pm 0.12}$	**0.74**$_{\pm 0.12}$	0.88	0.92	**0.92**
	Binary Multiplication	0.70$_{\pm 0.11}$	0.74$_{\pm 0.13}$	**0.78**$_{\pm 0.12}$	**0.92**	0.92	0.92
	Compute Sqrt	0.67$_{\pm 0.10}$	0.78$_{\pm 0.12}$	**0.84**$_{\pm 0.07}$	0.86	**0.89**	**0.89**
	Bucket Sort	0.63$_{\pm 0.08}$	**0.84**$_{\pm 0.09}$	0.69$_{\pm 0.13}$	0.88	**0.96**	0.83

Performance vs. Edit Distance



(a) Repeat 01

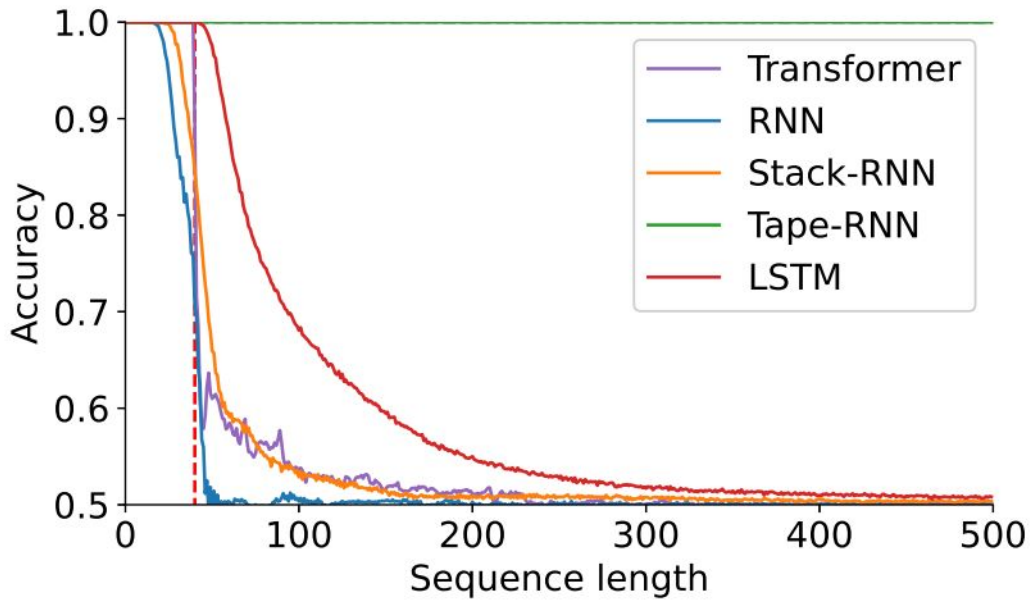
(b) Dyck-$(2, 3)$

Figure 2: Recognition cross-entropy (lower is better) as a function of edit distance for the transformer model shown under "Expressivity" in Table 2, on a separate dataset of 50 negative examples in the length range $[0, 500]$. The dashed lines show $\log 2$, the threshold for incorrect predictions. Despite being trained on a large proportion of negative examples with low edit distance, the transformer still struggles on examples that resemble positive examples.

Performance vs. Length

Seq-to-Seq (DeepMind)

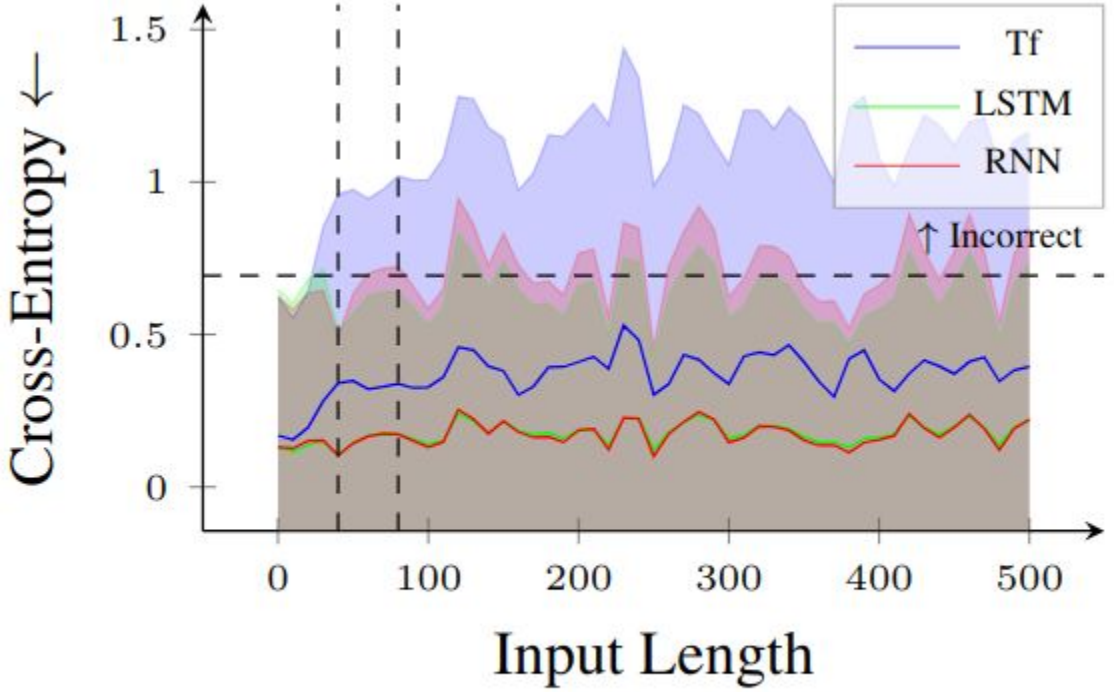
w => w



(i) Duplicate String (CS)

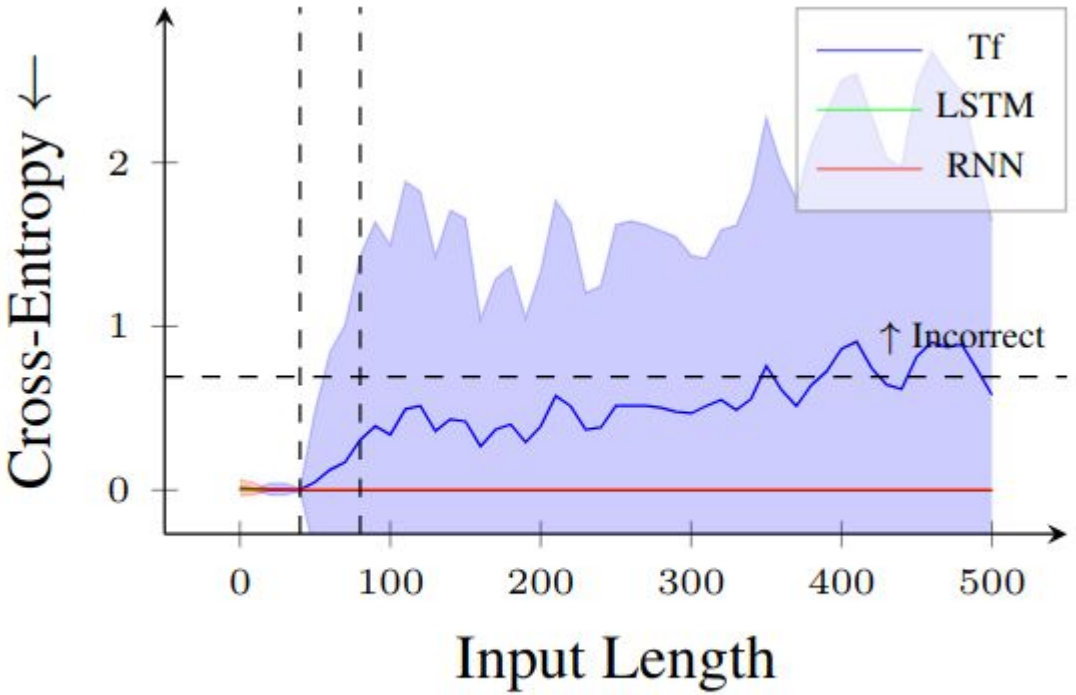
Recognition (Ours)

w#w

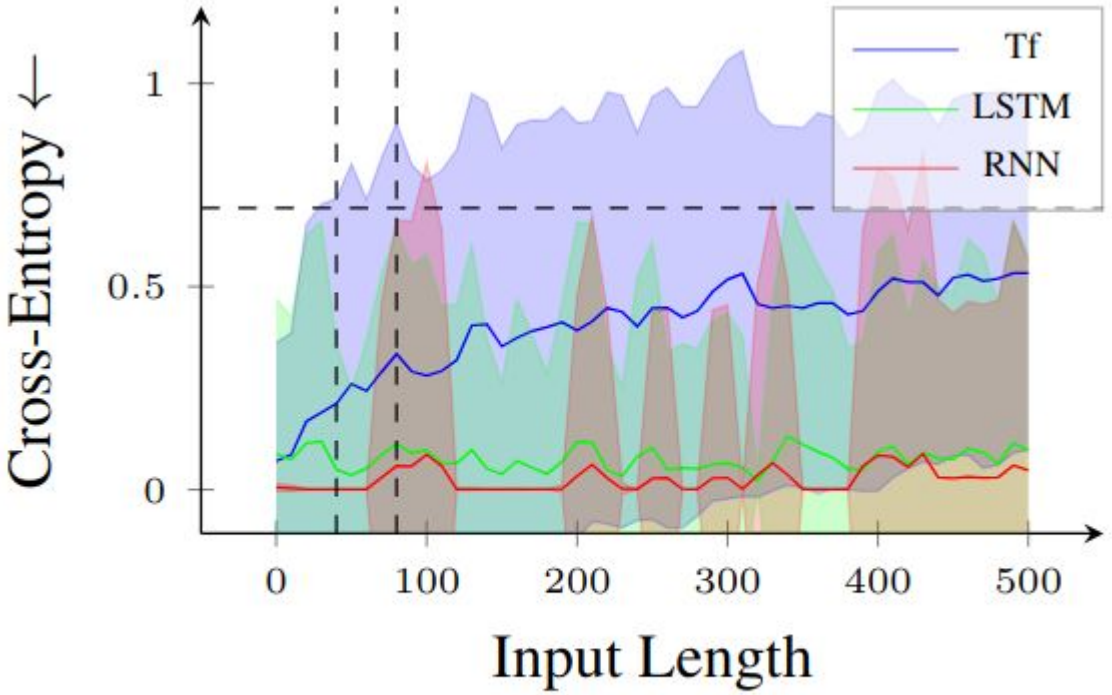


(l) Marked Copy

Performance vs. Length



(b) Repeat 01



(f) Dyck-(2, 3)

Which Loss Function Is Best?

Table 3: The best loss functions, corresponding to the accuracy scores reported in Table 2. "R" = recognition; "LM" = language modeling; "NS" = next symbol prediction. No single loss function consistently results in the best performance; the most frequent winner is just R.

	Inductive Bias			Expressivity		
Language	Tf	RNN	LSTM	Tf	RNN	LSTM
Even Pairs	R	R+LM+NS	R	R	R+NS	R
Repeat 01	R	R+NS	R	R	R	R
Parity	R+NS	R+NS	R+NS	R+NS	R+LM	R
Cycle Navigation	R+LM+NS	R	R	R	R	R
Modular Arithmetic	R	R	R	R+NS	R	R
Dyck-(2, 3)	R+LM	R+LM+NS	R	R+NS	R+NS	R+NS
First	R+NS	R+LM	R+LM	R	R	R
Majority	R+LM	R+NS	R+NS	R+LM	R+LM+NS	R+LM+NS
Stack Manipulation	R	R+NS	R	R+LM+NS	R	R+LM
Marked Reversal	R+NS	R+NS	R	R+LM	R+LM	R+LM
Unmarked Reversal	R	R+NS	R+NS	R	R+NS	R+NS
Marked Copy	R+NS	R+LM	R	R+NS	R	R
Missing Duplicate	R+LM+NS	R	R	R+LM+NS	R+LM+NS	R+LM
Odds First	R	R	R	R+LM+NS	R	R+LM+NS
Binary Addition	R	R+NS	R	R+LM	R+NS	R+NS
Binary Multiplication	R+NS	R	R+NS	R+NS	R+LM	R+NS
Compute Sqrt	R	R	R	R	R	R
Bucket Sort	R	R+LM+NS	R	R+NS	R+NS	R+LM+NS

What Did We Learn?

- Transformers, simple RNNs, and LSTMs rank very low in the Chomsky hierarchy
  - If true, they cannot make grammaticality judgments with 100% accuracy
- Simple RNNs and LSTMs outperform the transformer
- Using just a simple recognition objective is usually effective; auxiliary training objectives help in isolated cases but not uniformly
- Consistency between inductive bias and expressivity
- Results do change from the DeepMind paper
  - e.g., nothing is able to solve Cycle Navigation

Thank You

Questions?

References

- Alexandra Butoi, Ghazal Khalighinejad, Anej Svete, Josef Valvoda, Ryan Cotterell, Brian DuSell. Training Neural Networks as Recognizers of Formal Languages. In Proc. ICLR. Singapore, May 2025.
- Andy Yang, David Chiang, Dana Angluin. Masked Hard-Attention Transformers Recognize Exactly the Star-Free Languages. In Advances in NeurIPS. Vancouver, Canada, December 2024.
- Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, Pedro A Ortega. Neural Networks and the Chomsky Hierarchy. In Proc. ICLR 2023. Kigali, Rwanda, May 2023.
- Michael Hahn, Mark Rofin. Why are Sensitive Functions Hard for Transformers? In Proc. ACL. Bangkok, Thailand, Aug 2024.