NEURO-SYMBOLIC SYNTACTIC PARSING: SHAPING A NEURAL NETWORK WITH THE CYK ALGORITHM

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we show the possibility of a direct injection of algorithms into neural network architecture. We focus on a complex algorithm, that is, Cocke-Younger-Kasami (CYK), for parsing context-free grammars in Chomsky Normal Form, and we propose CYKNN, a simple recurrent neural network architecture for encoding the CYK algorithm in trainable matrix-vector multiplications.We experimented with a very simple grammar with 4 variations showing that our approach outperforms existing LLMs with more than 20B parameters with an in-context learning setting and smaller LLMs of the Qwen family fine-tuned with LoRA. Our attempt paves the way to a different approach to neuro-symbolic methodologies.

1 INTRODUCTION

The ability of transformer-based architectures (Vaswani et al., 2017) to *grok* simple algorithmic capabilities from data is particularly fascinating, in both shallow (Power et al., 2022) and deeper transformer architectures (DeepSeek-AI et al., 2025). The underlying mechanisms driving this phenomenon remain a subject of active debate and are being examined from multiple theoretical and empirical perspectives (Nanda et al., 2023; Tan & Huang, 2024; Huang et al., 2024). Moreover, it appears that these capabilities are acquired within a single training epoch in large language models (Li et al., 2026), whereas *grokking* in specialized shallow transformer architectures typically emerges only after multiple training epochs, subsequent to an initial phase of overfitting on the training data.

Demanding transformer-based architectures to extrapolate algorithmic knowledge from training data is an exaggerated expectation that exceeds what is required to human learners and, in fact, transformer-based Large Language Models do not succeed in learning complex algorithms (Mirzadeh et al., 2025; Shojaee et al., 2025). Indeed, this request calls for the most challenging form of generalization. Extrapolating algorithmic knowledge requires years to human experts and the fact that specific algorithms already exist does not facilitate the task for transformers that have to discover it from training data. Some reserach suggests that these transformers are only large language memories (Zanzotto et al., 2025).

For decades, programmers have instructed machines with algorithms to resolve specific tasks and machine learning was confined to dealing with cases that cannot be handled algorithmically where answers should be found also for possibly unexpected data. These days, machine learning specialists are asking to neural networks to learn what can be programmed. Although understanding this generalization ability of transformers is fascinating, a tantalizing questions arises: if algorithms are known, is there a way to encode directly these algorithms in a neural network like architecture governed by matrix-vector multiplication?

Coding algorithms directly in neural networks is a possible, intriguing alternative to the current trend of passively relying on emerging capabilities of transformers. Indeed, this idea stems from a debate dating back to the late '80s: do neural networks with their way of representing information have extra capabilities with respect to classical symbolic approaches underlying the activity of programming machines? In case of a negative answer, the alternative is viable. Fodor & Pylyshyn (1988) suggest that distributed representations underlying neural network architectures are "only an implementation of the Classical approach" where classical approach is related to discrete symbolic representations. Chalmers (1992) affirms that distributed representations give the important opportunity to reason *holistically* about encoded knowledge. This means that decisions over some specific part of the stored

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

knowledge can be taken without retrieving the specific part but acting on the whole representation. Substantially, neural networks and symbolic approaches are somehow similar even if large language models realized with transformer-based neural network architectures are astonishing.

In this paper, we pave the way for the direct injection of algorithms into neural network architecture by showing that we can encode at least a complex algorithm, that is, Cocke-Younger-Kasami (CYK) for parsing context-free grammars in Chomsky Normal Form. We stem from the initial studies of (Zanzotto et al., 2020) and we propose CYKNN, a simple recurrent neural network architecture for encoding the Cocke-Younger-Kasami (CYK) in trainable matrix-vector multiplications. In some sense, the injection of the algorithm shapes the architecture of the CYKNN. Stemming from a way to represent data treated by the algorithm, the algorithmic operations are translated into matrix-vector multiplications, which *holistically* change the status of the data. More concretely, holistically means changing *for*-cycles in a single matrix-vector operation. We experimented with a very simple grammar with 4 variations showing that our approach outperforms existing LLMs with more than 20B parameters with an in-context learning setting (OpenAI et al., 2025; Team et al., 2025) and smaller LLMs of the Qwen family (Qwen et al., 2025; Yang et al., 2025) fine-tuned with LoRA (Hanindhito et al., 2025).

2 Background and Notation

In this section, we introduce the basics about the CYK algorithm and overview a class of distributed representations called holographic reduced representation.

2.1 Why CKY parsing with context-free-grammars is important and what is the CYK algorithm?

Parsing algorithms based on context-free grammars (CFGs) have been central to computer science since the 1960s: these algorithms are the base for parsing programming languages as well as for parsing natural language. Hence, parsing algorithms fall into two broad classes, depending on whether or not they handle ambiguous grammars. For unambiguous grammars generally used for programming language compilers, specialized techniques such as LL and LR (Sippu & Soisalon-Soininen, 1990) parse input efficiently in *linear time* (Aho et al., 2006). For ambiguous grammars, general parsing methods construct compact parse forests representing all parse trees accepted by the grammar for a given input string. These are generally dynamic-programming algorithms that run in *polynomial time* in the input length (see (Graham & Harrison, 1976) for an overview). Parsing algorithms for ambiguous CFGs are widely used in areas such as natural language processing, where different parse trees correspond to different semantic interpretations (Nederhof & Satta, 2010).

The Cocke–Younger–Kasami algorithm (CYK) (Cocke, 1969; Younger, 1967; Kasami, 1965) was the first parsing method for ambiguous CFGs to be discovered. This algorithm is at the core of many common algorithms for natural language parsing, where it is used in combination with probabilistic methods and other machine learning techniques to estimate the probabilities for grammar rules based on large datasets (Charniak, 1996), as well as to produce a parse forest of all parse trees for an input string and retrieve the trees having the overall highest probability (Huang & Chiang, 2005).

The CYK algorithm is a classical dynamic-programming algorithm for parsing and recognition based on context-free grammars (CFGs). We provide here a brief description of the algorithm in order to introduce the notation used in later sections; we closely follow the presentation in (Graham & Harrison, 1976), and we assume the reader is familiar with the formalism of CFG ((Hopcroft et al., 2001), Chapter 5). The algorithm requires CFGs in Chomsky normal form, where each rule has the form $A \to BC$ or $A \to a$, where A, B, and C are nonterminal symbols and a is a terminal symbol. We write R to denote the set of all rules of the grammar and NT to denote the set of all its nonterminals.

Given an input string $w = a_1 a_2 \cdots a_n$, $n \geq 1$, where each a_i is an alphabet symbol, the algorithm uses a two-dimensional table P of size $(n+1) \times (n+1)$, where each entry stores a set of nonterminals representing partial parses of the input string. More precisely, for $0 \leq i < j \leq n$, a nonterminal A belongs to set $P[i, j]$ if and only if there exists a parse tree with root A generating the substring $a_{i+1} \cdots a_j$ of w. Thus, w can be parsed if the initial nonterminal of the grammar S is added to $P[0, n]$. Algorithm 1 shows how table P is populated. P is first initialized using unary rules, at

Line 3. Then, each entry $P[i, j]$ is filled at Line 9 by looking at pairs $P[i, k]$ and $P[k, j]$ and by using binary rules.

Algorithm 1 CYK(string $w = a_1 a_2 \cdots a_n$, rule set R) **return** table P.

```
1: for i ← 1 to n do
2:     for each A → a_i in R do
3:         add A to P[i − 1, i]
4: for j ← 2 to n do
5:     for i ← j − 2 to 0 do
6:         for k ← i + 1 to j − 1 do
7:             for each A → BC in R do
8:                 if B ∈ P[i, k] and C ∈ P[k, j] then
9:                     add A to P[i, j]
```

Grammar rules R

$$\begin{array}{lcl} S & \rightarrow & DE \\ S & \rightarrow & DS \\ D & \rightarrow & a \\ E & \rightarrow & b \end{array}$$

Table P

D		S
(0,1)	(0,2)	(0,3)
	D	S
	(1,2)	(1,3)
		E
		(2,3)

Figure 1: The Cocke–Younger–Kasami (CYK) parsing algoritm, a simple context-free grammar (CFG), the CYK parsing table P for the input string $w = aab$

A running example is presented in Figure 1, showing a set R of grammar rules along with the table P produced by the algorithm when processing the input string $w = aab$. For instance, S is added to $P[1, 3]$, since $D \in P[1, 2]$, $E \in P[2, 3]$, and $(S \rightarrow DE) \in R$. Since $S \in P[0, 3]$, we conclude that w can be generated by the grammar.

2.2 Holographic Reduced Representations to Encode Symbolic Reasoning in Neural Networks

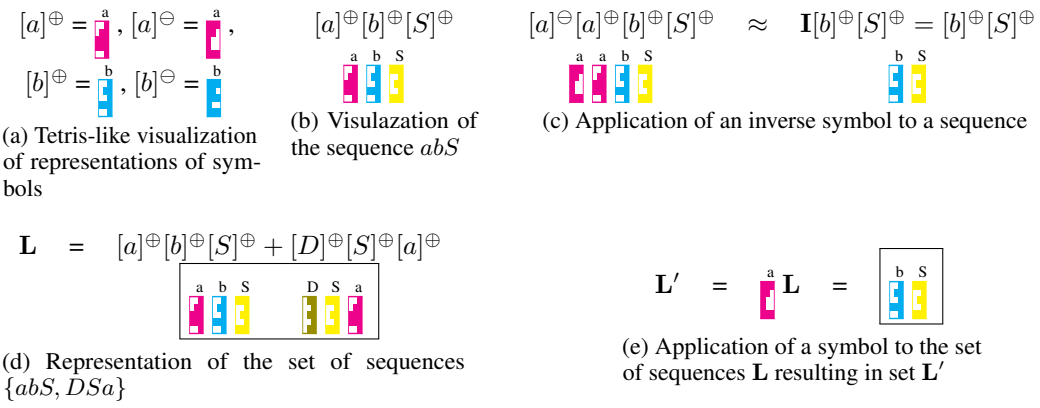
Holographic Reduced Representations (HRRs), originally proposed in (Plate, 1995) and later extended in (Zanzotto & Dell'Arciprete, 2012), are distributed representations that are particularly suitable for our goal of encoding the two-dimensional CYK parsing table P and for implementing the operation that retrieves the content of its cells $P[i, j]$. In what follows, we present the operations we employ, together with a graphical notation to illustrate their properties based on Tetris-like pieces.

In line with Zanzotto et al. (2020), we use a revised version of extended HRRs that exploits the shuffled circular convolution in the description of the symbols. Each symbol a is represented as a matrix $[a]^{\oplus} = \mathrm{A}_{\circ}\Phi$ in $\mathbb{R}^{d \times d}$ for some constant d, where $\mathrm{A}_{\circ}$ is the circulant matrix of a and Φ is a permutation matrix. Basically, given a symbol a, $[a]^{\oplus}$ is the matrix representing its random vector $\vec{a}$ ready for the shuffled circular convolution. Along with $[a]^{\oplus}$, it is possible to define the inverse representation $[b]^{\ominus} = \Phi^T {\mathrm{A}_{\circ}}^T$ of a symbol b. The matrix multiplication $[a]^{\oplus}[b]^{\ominus}$ has then the following property:

$$[a]^{\oplus}[b]^{\ominus} \approx \begin{cases} \mathbf{I} & \text{if } \vec{a} = \vec{b} \\ \mathbf{0} & \text{if } \vec{a} \neq \vec{b} \end{cases}$$

(see Appendix C for details).

The above representations have the interesting feature that they delete symbols in a chain of matrix multiplications whenever representations of symbols appear next to their inverse representations. In fact, any two adjacent inverse representations of symbols are replaced by the nearly identity matrix, which is invariant for matrix multiplication. As this behavior is reminiscent of what happens in Tetris, where pieces eliminate lines whenever their shapes perfectly fill the gaps, Zanzotto et al. (2020) adopted a Tetris-like visualization for the above operations (see Fig. 2.(a)). In this visualization, sequences of symbols correspond to sequences of pieces (see Fig. 2.(b)). Then, as in Tetris, elements whose shapes complement each other cancel out and are removed from the sequence (see Fig. 2.(c)). Collections of sequences of symbols (sums of matrix products representing sequences) are depicted inside boxes (see Fig. 2.(d)). In addition to the usual Tetris rules, we assume here that an element with a certain shape selects from a box only elements with the complementary shape facing it (see Fig. 2.(e)).



(a) Tetris-like visualization of representations of symbols

(b) Visulazation of the sequence abS

(c) Application of an inverse symbol to a sequence

(d) Representation of the set of sequences $\{abS, DSa\}$

(e) Application of a symbol to the set of sequences $\mathbf{L}$ resulting in set $\mathbf{L}'$

Figure 2: Tetris-like representation of matrix representations of symbols

Using the Tetris metaphor, we describe our new model to encode P tables as matrices of real numbers $\mathbf{P}$, and how we can implement CFG rule applications by means of matrix multiplication.

3 Encoding the CYK algorithm in a Neural Network

Our goal is to demonstrate that a symbolic algorithm can be expressed in terms of linear algebraic operations, that is, matrix-vector multiplication. By using this new formulation, we can systematically transform a symbolic algorithm into an equivalent formulation that induces a corresponding neural network architecture. To demonstrate this possibility, we will use the well-known CYK algorithm.

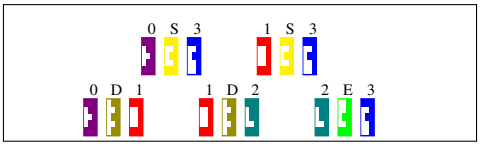
In the following, we first describe the inspiring idea and then the corresponding neural network, the CYKNN, designed to perform the CYK algorithm steps.

3.1 The Inspiring Idea

Our methodology to build a neural network shaped by a CYK algorithm stems from a simple, game-changing idea: associating the representation of the cells of CKY table with a convenient way of representing rules so that the update operation of the table is simple and can be done holistically. Let's represent symbols in cells $A \in P[i, j]$ in a distribued matrix $\mathbf{P}$ as:

$$[i]^{\oplus}[A]^{\oplus}[j]^{\oplus}$$

In this case, the overall matrix $\mathbf{P}$ for the running example in Figure 1 is:



If this representation is associated with a correlated formalization of grammar rules, the update operation may become very simple. Binary grammar rules $A \rightarrow BC$ can be represented in the following way:

$$\mathbf{W} = \sum_{A \rightarrow BC \in R} [B]^{\ominus}[A]^{\oplus}[C]^{\ominus}$$

Then the matrix representation $\mathbf{R}$ of set R is the following:

$$\mathbf{R} = \sum_{k=1}^{n} [k]^{\ominus} \mathbf{W} [k]^{\ominus}$$

Algorithm 2 ProtoCKYNN($\mathbf{P}_0$) **return** table P

1: $\mathbf{P} \leftarrow \mathbf{P}_0$
2: **for** $i \leftarrow 0$ to $n-1$ **do**
3: $\quad \mathbf{P}' \leftarrow \mathbf{PRP}$
4: $\quad \mathbf{P} \leftarrow \mathbf{P} + \mathbf{P}'$

The matrix $\mathbf{R}$ along with the matrix $\mathbf{P}$, which represents the CYK matrix, can be used to determine the next step starting from a previous status. If the matrix $\mathbf{P}$ is filled with all the constituents up to a given length, the product $\mathbf{P}' = \mathbf{PRP}$ will operate on addends that can result in the following way. For each i, k, j and for every rule$A \rightarrow BC$ there will be addends of this kind:

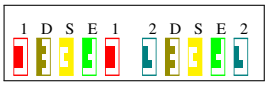
$$[i]^{\oplus}[B]^{\oplus}[k]^{\oplus}[k]^{\ominus}[B]^{\ominus}[A]^{\oplus}[C]^{\ominus}[k]^{\ominus}[k]^{\oplus}[C]^{\oplus}[j]^{\oplus} = [i]^{\oplus}[A]^{\oplus}[j]^{\oplus}$$

that will add to $\mathbf{P}'$ constituents of length $j - i$. The CKY parser will then be simplified into the cycle described in Algorithm 2. At the end of the cycle, the matrix $\mathbf{P}$ will contain all the constituents related to the input sentence. The proposed algorithm is overly simplified and, as such, cannot be expected to be fully accurate. In particular, the reduction in computational complexity from $O(n^3)$ to $O(n)$ is achieved at the cost of introducing an approximation at each iteration of the processing cycle. To mitigate this limitation, in the neural network implementation, we incorporated a signal amplification and refinement mechanism for the representations of constituent symbols, which is applied at every step of the procedure.

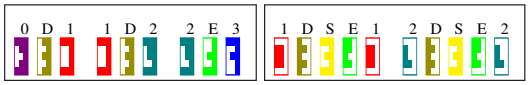
To clarify the underlying concept, we employ our ongoing example in Fig. 1. We begin with a CYK matrix that contains only the first diagonal, namely $CYK_{\text{matrix}} = \{(0, 1, D), (1, 2, D), (2, 3, E)\}$. In the distributed representation, this matrix is expressed as follows:

$\mathbf{P} =$ 

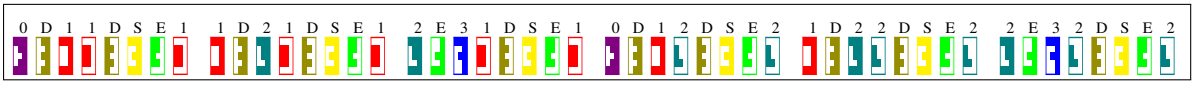
In addition to the status of the matrix $\mathbf{P}$, a simplified rule matrix $\mathbf{R}$ representing only the binary rule $S \rightarrow DE$ is given by:

$\mathbf{R} =$ 

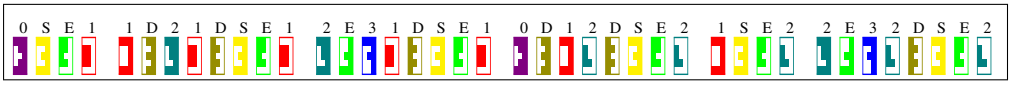
The application of the principal operation $P' = PRP$ within the iterative cycle of Algorithm 2 yields the following effect:

$\mathbf{P}' = \mathbf{PRP} =$  $\mathbf{P} =$

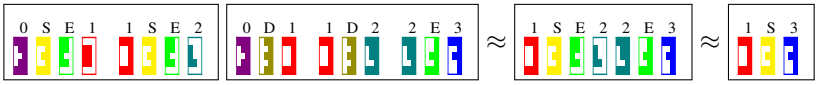
Focusing on the first $\mathbf{PR}$, the matrix multiplication creates all the following sums of matrices:

$=$  $\mathbf{P} =$

Some of these addends contain adjacent matrices and their inverses, which can be algebraically canceled:

$=$  $\mathbf{P} \approx$

Upon removal of extended matrix sequences, identified as noise, from the visualization, the following behavior is observed:

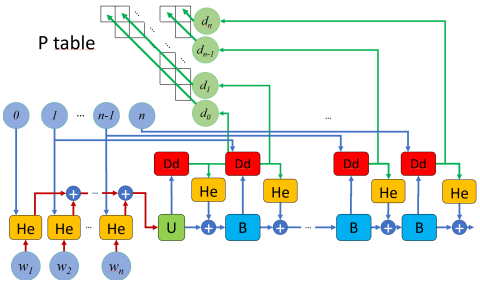
$\approx$ 

Then, finally, the application of this single operation generates the filling of the new diagonal, taking into account longer sequences, that is, $\mathbf{P}'$ contains $(1, 3, S)$. This is a clear example of *holistic reasoning*.

With respect to what is proposed in (Zanzotto et al., 2020), our approach proposes an extremely different algorithm that requires only one representation matrix instead of two very complex matrices representing the left-hand and the right-hand sides. Moreover, the algorithm in (Zanzotto et al., 2020) has never evolved in a real neural network where backprop can be applied given the complex operations implied by the left and the right matrices representing the CYK matrix. Our approach is extremely more simple.

3.2 CYK Neural Network (CYKNN)

The inspiring idea represented by the Algorithm 2 may be translated in a more formal recurrent neural network architecture; see Fig. 3(a). Indeed, *for* cycles in the algorithm call for a recurrent architecture or a transformer-like architecture where all the operations are done in a single pass.



(a) Recurrent CYKNN Architecture

Algorithm 3 Dd(**P**,v) **return** ($\mathbf{P_v}$,d_v)

1: **for** $i \leftarrow 0$ to $n-1-v$ **do**
2: $\quad \vec{o} \leftarrow \sigma(\mathbf{W}[i]^{\ominus}\mathbf{P}[i+v+1]^{\ominus}\vec{1})$
3: $\quad$ **for** $s \in NT$ **do**
4: $\quad\quad$ **if** $\vec{o}_{index(s)} > t$ **then**
5: $\quad\quad\quad \mathbf{P_v} \leftarrow \mathbf{P_v} + [i]^{\oplus}[s]^{\oplus}[i+v+1]^{\oplus}$

(b) Diagonal decoder algorithm: module **Dd**

Figure 3: The CYK in a Recurrent Neural Network (CYKNN)

The overall architecture, called CYKNN, contains then **Holographic Encoder (He)**, the **Unary Module** (**U**), the **Binary Module**, and the **Diagonal Decoder (Dd)**. These modules are hereafter defined.

The **Holographic Encoder (He)** represents the module that transforms input tokens w_i into an initial matrix $\mathbf{P}_0$ representing all the tokens. Each token is represented along with its span from i to $i+1$, that is:

$$o_i = [i]^{\oplus}[w_i]^{\oplus}[i+1]^{\oplus}$$

As depicted in the architecture in Fig. 3, these o_i are summed up as a single $\mathbf{P}_0$ matrix:

$$\mathbf{P}_0 = \sum_{i=0}^{n-1} [i]^{\oplus}[w]^{\oplus}[i+1]^{\oplus}$$

The **Unary Module** (**U**) is responsible for applying unary rules within the CYKNN architecture. It applies the unary grammar once to obtain the first diagonal of the explicit CYK matrix in a distributed manner P_1. The *holistic* operation is:

$$\mathbf{P}_1 = \mathbf{P}_0\mathbf{R}_U$$

that exploits the matrix $\mathbf{P}_0$ and the grammar of the unitary rules encoded in $\mathbf{R}_U$. The grammar is initialized with:

$$\mathbf{W}_U = \sum_{A \to a \in R} [a]^{\ominus}[A]^{\oplus}$$

Then, for a sentence of length n, R_U is obtained in the following way:

$$\mathbf{R}_U = \sum_{i=1}^{n} [i]^{\ominus}\mathbf{W}_U[i]^{\oplus}$$

After the initialization, the grammar matrix $\mathbf{W}_U$ is trainable.

The **Binary Module** (**B**) is the module that implements the main operation of the Algorithm 2, that is:

$$\mathbf{P}_{i+1} = \mathbf{P}_i\mathbf{R}_B\mathbf{P}_i$$

In this case, the initialization of the matrix $\mathbf{W}_B$ representing the binary rules of the grammar is:

$$\mathbf{W}_B = \sum_{A \to BC \in R} [B]^{\ominus}[A]^{\oplus}[C]^{\ominus}$$

For a sentence of length n, $\mathbf{R}_B$ is obtained in the following way:

$$\mathbf{R}_B = \sum_{i=1}^{n} [i]^{\ominus} \mathbf{W}_B [i]^{\ominus}$$

As for the unary rule grammar $\mathbf{W}_U$, also the grammar matrix $\mathbf{W}_B$ is trainable after the initialization.

The **Diagonal Decoder (Dd)** is the most complex module as it extract cells of the diagonals from the encoded CYK matrix $\mathbf{P}_i$. This is done for two reasons. The first reason is the construction of the final symbolic CYK matrix. The second reason is to purify the signal to be inserted in the $\mathbf{P}_{i+1}$ matrix so that symbols are better presented in the novel matrix. The key component of the module is the following operation:

$$\vec{o} = \sigma(\mathbf{W}_{NT}([i]^{\ominus}\mathbf{P}[j]^{\ominus})_0)$$

where $\mathbf{W}_{NT}^T$ is the decoding matrix for non terminal symbols (NT), $\sigma(\ )$ is a sigmoid function and $([i]^{\ominus}\mathbf{P}[j]^{\ominus})_0$ is the first column of the circulant matrix of $[i]^{\ominus}\mathbf{P}[j]^{\ominus}$ that represents the sum of the representations of the symbols in the cell (i, j). The first column of the circulant matrix is the vectorized version of the sum. Then, each dimension o_i of the vector $\vec{o}$ represent the presence of the symbol $s \in NT$ where $index(s) = i$ in the cell. The **Dd** algorithm is reported in Figure 3(b), where the output is d_v that contains the cell of the CYK table of the diagonal v and $\mathbf{P_V}$ representing its distributed representation version.

3.3 Experimental investigation

In this section, we aim to demonstrate CYKNN's ability to perform CYK parsing and we compare it with current transformer-based approaches to perform the same task.

3.3.1 Experimental set-up

To experiment with the parsing ability of existing large language models and of our novel neural network architecture, CYKNN, that realizes a CYK with differentiable matrix-vector multiplications, we first set up a very simple grammar $G_0 = (T, NT, S, R_U \cup R_B)$ (see Appendix A) and we parsed 5,414 randomly-generated sequences with a traditional symbolic CYK (see Appendix B for an example of the dataset). The dataset has been then split in the three classical parts: training with 3,000 sequences, testing with 2,000 sequences, and development with 414 sequences.

Evaluation measures assess the ability of each method to reconstruct the oracle CYK matrix. Then, we used the classical precision, recall, and f-measure over the CYK matrices produced by the systems. Just for completeness, we here report the definition of these 3 measures on the CYK matrices P_{oracle} and P_{system}. Defining two sets O and S, respectively, for the oracle and the system P matrices as $O = \{(i, j, Sym)|Sym \in P_{oracle}[i, j]\}$ and $S = \{(i, j, Sym)|Sym \in P_{system}[i, j]\}$, precision, recall, and f-measure are classically defined as $Prec = |O \cap S|/|S|$, $Rec = |O \cap S|/|O|$, and $fmeasure = 2Prec \cdot Rec/(Prec + Rec)$. For the evaluation of the larger language models, we produced a subset *Small Test* of the testing set that contains only sequences of lengths 5,6,7, and 8. The complete test dataset is referred as *Full Test*.

In these experiments, we aim to determine whether current transformer-based LLMs can solve this neuro-symbolic task. For this reason, we experimented with two classes of approaches: (1) a finetuning approach for relatively large language models; (2) an in-context learning approach with larger language models. For the first class, we used two models of the Qwen family: Qwen2.5-1.5B-Instruct (Qwen et al., 2025) and Qwen3-8B (Yang et al., 2025). Both Qwen2.5-1.5B-Instruct and Qwen3-8B were fine-tuned with LoRA (Hanindhito et al., 2025), using 20% of the parameters, for 3 training epochs on the Train set for training. Testing has been performed on the Full Test Set and extrapolated for the Reduced Test Set. For the second class, we used two models, gpt-oss-20B (OpenAI et al., 2025) and Gemma-3-27B-it, referred as Gemma-27B (Team et al., 2025). In this setting, no additional training was performed; instead, we designed a few-shot prompt (see Appendix

Table 1: Precision, recall, and f-measure on the reduced and the full test set for CYKNN, for two medium-sized pre-trained LLMs with fine-tuning (Qwen2.5-1.5B and Qwen3-8B), and for two large-sized LLMs with in-context learning (Gemma-27B and gpt-oss-20B)

	Small Test			Full Test		
Model	*Prec*	*Rec*	*f-measure*	*Prec*	*Rec*	*f-measure*
Qwen2.5-1.5B-Instruct	0.574	0.036	0.067	0.521	0.216	0.305
Qwen3-8B	0.378	0.004	0.007	0.689	0.219	0.333
gpt-oss-20B	**0.940**	0.589	0.724	-	-	-
Gemma-27B	0.258	0.355	0.299	-	-	-
CYKNN G=Random	0.936	0.264	0.412	**0.900**	0.173	0.290
CYKNN without CF	0.749	0.583	0.656	0.865	0.394	0.541
CYKNN with G_0	0.777	**0.837**	**0.806**	0.785	**0.765**	**0.775**

D) to guide the models toward producing a well-formed CYK parse matrix, and evaluated them on strings of length 5, 6, 7, and 8 drawn from the same test set used for the other models.

To evaluate the ability of the CYKNN to model grammars in matrices, we experimented with 3 additional grammars, G_1, G_2, and G_3 built by augmenting G_0 with additional rules to make more complex the encoding of the grammars in the distributed matrices W_B and W_U representing the binary and the unary rules of the grammar itself. Grammars G_1, G_2, and G_3 have 20 unary rules and, respectively 5, 8, and 21 binary rules. Then, grammar G_1 increases only the number of unary rules with respect to grammar G_0.

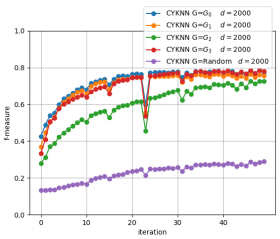
CYKNNs have been tested with different sizes d of the embedding vectors: 100, $1,000$, and $2,000$. We trained CYKNNs with randomly initialized grammar matrices W_U and W_B and with these matrices initialized to express G_0, G_1, G_2, and G_3. The version with random intialized matrices is referred to as CYKNN G=Random. Versions with initialized grammar matrices are referred to as CYKNN G=G_i.

Finally, we also experimented with *CYKNN without CF* that is basically an implementation of an RNN, where P' = RP. We call it "without context-free (CF)" because our CYKNN can implement a context-free parser with a push-down automaton, whereas a normal Recurrent Neural Network can implement a Finite State Automaton.

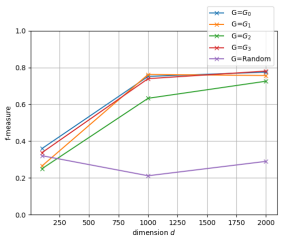
Experiments with the CYKNN family have been performed on a server with a single NVIDIA GeForce GTX 1070 with 8 Gb of on-board RAM. Experiments run for 3 days. Fine-tuning of Qwen family has been performed on a server with two NVIDIA RTX A6000 with 48Gb of on-board RAM.

3.3.2 Results and Discussion

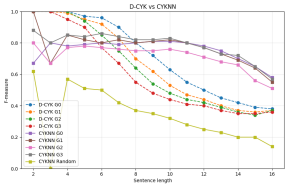
CYKNN is a good model for performing CYK parsing as a symbolic task with a neural network architecture (see Table 1). Indeed, CYKNN with grammar G_0 outperforms all the models both in the Small and in the Full Sets.



(a) Learning curve of CKYNNs with differentgrammars and with dimension $d = 2,000$



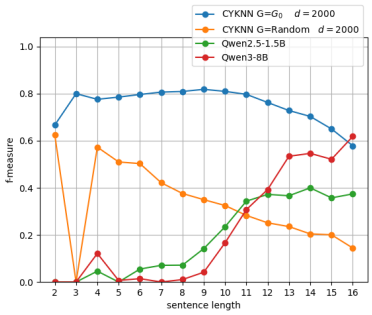
(b) Performance of CYKNNs with different dimensions d of the representation space



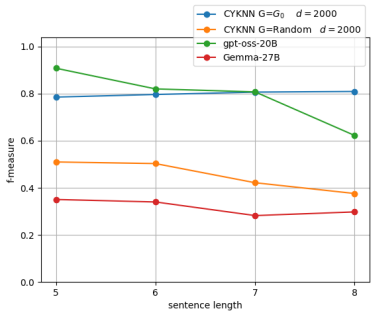
(c) Performance of CYKNNs vs. sequence lengths and commparison with D-CYK (Zanzotto et al., 2020)

Figure 4: Learning abilities of the CYKNN model family

The neuro-symbolic task of parsing with the CYK algorithm seems to be a difficult task for existing LLMs (see Table 1). The fine-tuned Qwen2.5-1.5B-Instruct and Qwen3_8b, and the in-context learned Gemma-27B perform poorly on the Small Set. Only the in-context learned gpt-oss-20B has higher performance in terms of f-measure. This high f-measure is mainly due to a high precision, as recall is quite low. Then, apparently, gpt-oss-20B is conservative when filling the CYK matrix. Very strangely, performances of Qwen2.5-1.5B-Instruct and Qwen3-8B perform are lower on shorter sequences of the Small Test than on longer sentences included in the Full Test (cf. Fig. 4(a)).



(a) Performance over sequence lengths of two CYKNNs with dimension $d = 2,000$ with grammar G_0 and a random grammar matrix vs. two finetuned *qwen* models



(b) Performance over sequences of length 5,6,8, and 9 of two CYKNNs with dimension $d = 2,000$ with grammar G_0 and a random grammar matrix vs. two larger LLMs

Figure 5: CKYNNs with respect to finetuned and existing large LLMs with in-context learning: Performance in terms of f-measure with respect to the oracle on sequences of the test set.

There also seems to be a deeper issue: the context-free, algorithmic nature of the CYK parsing task is not captured by a transformer architecture equipped with attention (Vaswani et al., 2017). Fine-tuning of pre-trained LLMs does not help the transformers underlying Qwen2.5-1.5B-Instruct and Qwen3-8B to learn how to produce CYK matrices. Fine-tuned Qwen models seem to learn the association of the prefixes of sequences to the initial part of the CYK matrices. Fine-tuned Qwen models outperform CYKNN with random grammar matrix only for sequences longer than 12 tokens. This fact, along with the extremely low performance for sequences smaller than 10 tokens, suggests that the underlying transformers are somehow only memorizing and are not inducing any algorithm from data.

Within the CYKNN family, the role of the complexity of the initialization of the grammar (G_0 vs. G_1, G_2, and G_3) is relevant with respect to the learning curves (see Fig. 4(a)). When the grammar has more distractors (e.g., G_2, the learning is slower. Nevertheless, the low growth of the CYKNN initialized with a random matrix shows that it is crucial to inject existing knowledge coding of the grammar into matrices. It is particularly relevant the size of the encoding dimension d of the symbols (see Fig. 4(b)). Moreover, in contrast with fine-tuned LLMs, the CYKNN family behaves in a more intuitive way with respect to the length of the sequences (see Fig. 5(a)), even with more complex grammars (see Fig. 4(c)). The context-free boost is extremely relevant, as the model without it (CYKNN without CF) performs worse (see Tab. 1). Finally, our model outperforms the previous closest model D-CYK (Zanzotto et al., 2020) for long sequences (see Fig. 4(c)).

4 Conclusions

The goal of this paper is to show that there is a radically different way to use neural networks when performing already-known algorithms: leveraging the parallelism of classical symbolic approaches and representations within neural networks to shape the neural network directly according to the algorithm. We showed the feasibility of the methodology by designing CYKNN as a neural network architecture devoted to perform the CYK parsing algorithm. CYKNN positively exploits encoded grammars and outperforms existing transformer-based LLMs that have inherent limits in performing the neuro-symbolic task of parsing with context-free grammars. Overall, results open the avenue to novel neuro-symbolic approaches where algorithms shape neural networks.

5 LIMITATIONS

As this paper presents a innovative way of thinking, it has a number of limitations. First, the used grammar is extremely simple. Yet, we have shown that it is already enough complex for current transformer-based LLMs. Second, the code of the CYKNN family is not optimized, thus it leaves the impression that the computation is very slow and, consequently, the computational complexity very high. On the contrary, the computational complexity of the algorithm reduces the computational complexity of the classical CYK algorithm.

REFERENCES

Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA, 2006. ISBN 0321486811.

David J. Chalmers. *Syntactic Transformations on Distributed Representations*, pp. 46–55. Springer Netherlands, Dordrecht, 1992. ISBN 978-94-011-2624-3. doi: 10.1007/978-94-011-2624-3_3. URL https://doi.org/10.1007/978-94-011-2624-3_3.

Eugene Charniak. *Statistical Language Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1996. ISBN 0262531410.

John Cocke. *Programming Languages and Their Compilers: Preliminary Notes*. Courant Institute of Mathematical Sciences, New York University, 1969. ISBN B0007F4UOA.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Jun-Mei Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiaoling Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bing-Li Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dong-Li Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, JingChang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Jiong Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, M. Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, Ruiqi Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shao-Kang Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wen-Xia Yu, Wentao Zhang, Wangding Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyu Jin, Xi-Cheng Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yi Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yu-Jing Zou, Yujia He, Yunfan Xiong, Yu-Wei Luo, Yu mei You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yao Li, Yi Zheng, Yuchen Zhu, Yunxiang Ma, Ying Tang, Yukun Zha, Yuting Yan, Zehui Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhen guo Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zi-An Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645:633 – 638, 2025. URL https://api.semanticscholar.org/CorpusID:275789950.

Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1):3 – 71, 1988. ISSN 0010-0277. doi: https://doi.org/10.1016/0010-0277(88)90031-5. URL http://www.sciencedirect.com/science/article/pii/0010027788900315.

S. L. Graham and M. A. Harrison. Parsing of general context free languages. In *Advances in Computers*, volume 14, pp. 77–185. Academic Press, New York, NY, 1976.

Bagus Hanindhito, Bhavesh Patel, and Lizy K. John. Large language model fine-tuning with low-rank adaptation: A performance exploration. In *Proceedings of the 16th ACM/SPEC International Conference on Performance Engineering*, ICPE '25, pp. 92–104, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400710735. doi: 10.1145/3676151.3719377. URL https://doi.org/10.1145/3676151.3719377.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 2nd edition*. Addison-Wesley, Reading, MA, 2001.

Liang Huang and David Chiang. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pp. 53–64, Vancouver, British Columbia, October 2005. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W05-1506.

Yufei Huang, Shengding Hu, Xu Han, Zhiyuan Liu, and Maosong Sun. Unified View of Grokking, Double Descent and Emergent Abilities: A Perspective from Circuits Competition, February 2024. URL http://arxiv.org/abs/2402.15175. arXiv:2402.15175.

William B Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics*, 26(January 1984):189–206, 1984. ISSN 1042-9832. doi: 10.1090/conm/026/737400. URL https://www.researchgate.net/profile/William{_}Johnson16/publication/235008656{_}Extensions{_}of{_}Lipschitz{_}maps{_}into{_}a{_}Hilbert{_}space/links/55e9abf908aeb65162649527.pdf.

Tadao Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. Technical report, Air Force Cambridge Research Lab, Bedford, MA, 1965.

Ziyue Li, Chenrui Fan, and Tianyi Zhou. Grokking in LLM Pretraining? Monitor Memorization-to-Generalization without Test, February 2026. URL http://arxiv.org/abs/2506.21551. arXiv:2506.21551.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. In *Proceedings of ICLR*, 2025. doi: 10.48550/arXiv.2410.05229. URL http://arxiv.org/abs/2410.05229. arXiv:2410.05229.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *Proceedings of ICLR*, 2023. doi: 10.48550/arXiv.2301.05217. URL http://arxiv.org/abs/2301.05217. arXiv:2301.05217.

Mark-jan Nederhof and Giorgio Satta. Theory of parsing. In Alexander Clark, Chris Fox, and Shalom Lappin (eds.), *The Handbook of Computational Linguistics and Natural Language Processing*, chapter 4, pp. 105–130. Wiley, 2010.

OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, Che Chang, Kai Chen, Mark Chen, Enoch Cheung, Aidan Clark, Dan Cook, Marat Dukhan, Casey Dvorak, Kevin Fives, Vlad Fomenko, Timur Garipov, Kristian Georgiev, Mia Glaese, Tarun Gogineni, Adam Goucher, Lukas Gross, Katia Gil Guzman, John Hallman, Jackie Hehir, Johannes Heidecke, Alec Helyar, Haitang Hu, Romain Huet, Jacob Huh, Saachi Jain, Zach Johnson, Chris Koch, Irina Kofman, Dominik Kundel, Jason Kwon, Volodymyr Kyrylov, Elaine Ya Le, Guillaume Leclerc, James Park Lennon, Scott Lessans, Mario Lezcano-Casado, Yuanzhi Li, Zhuohan Li, Ji Lin, Jordan Liss, Lily, Liu, Jiancheng Liu, Kevin Lu, Chris Lu, Zoran Martinovic, Lindsay McCallum, Josh McGrath, Scott McKinney, Aidan McLaughlin, Song Mei, Steve Mostovoy, Tong Mu, Gideon Myles, Alexander Neitz, Alex Nichol, Jakub Pachocki, Alex Paino, Dana Palmie, Ashley Pantuliano, Giambattista Parascandolo, Jongsoo Park, Leher Pathak, Carolina Paz, Ludovic Peran, Dmitry Pimenov, Michelle Pokrass, Elizabeth Proehl, Huida Qiu, Gaby Raila, Filippo Raso, Hongyu

Ren, Kimmy Richardson, David Robinson, Bob Rotsted, Hadi Salman, Suvansh Sanjeev, Max Schwarzer, D. Sculley, Harshit Sikchi, Kendal Simon, Karan Singhal, Yang Song, Dane Stuckey, Zhiqing Sun, Philippe Tillet, Sam Toizer, Foivos Tsimpourlas, Nikhil Vyas, Eric Wallace, Xin Wang, Miles Wang, Olivia Watkins, Kevin Weil, Amy Wendling, Kevin Whinnery, Cedric Whitney, Hannah Wong, Lin Yang, Yu Yang, Michihiro Yasunaga, Kristen Ying, Wojciech Zaremba, Wenting Zhan, Cyril Zhang, Brian Zhang, Eddie Zhang, and Shengjia Zhao. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

Tony A. Plate. Holographic Reduced Representations. *IEEE Transactions on Neural Networks*, 6(3): 623–641, 1995. ISSN 19410093. doi: 10.1109/72.377968.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets, January 2022. URL http://arxiv.org/abs/2201.02177. arXiv:2201.02177.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Magnus Sahlgren. An Introduction to Random Indexing. *Proceedings of the Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering, TKE 2005*, pp. 1–9, 2005. ISSN 1570-7075. doi: 10.1.1.96.2230. URL http://krextown.googlecode.com/svn-history/r41/trunk/infomap/RI{_}intro.pdf.

Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. The Illusion of Thinking: Understanding the Strengths and Limitations of Reasoning Models via the Lens of Problem Complexity. In *Proceedings of NeurIPS*, 2025. doi: 10.48550/arXiv.2506.06941. URL http://arxiv.org/abs/2506.06941. arXiv:2506.06941.

S. Sippu and E. Soisalon-Soininen. *Parsing Theory: $LR(k)$ and $LL(k)$ Parsing*, volume 2. Springer-Verlag, Berlin, Germany, 1990.

Zhiquan Tan and Weiran Huang. Understanding Grokking Through A Robustness Viewpoint, February 2024. URL http://arxiv.org/abs/2311.06597. arXiv:2311.06597.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma,

Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Daniel H. Younger. Recognition and parsing of context-free languages in time on^3. *Information and Control*, 10(2):189–208, 1967. ISSN 00199958. doi: 10.1016/S0019-9958(67)80007-X. URL http://www.sciencedirect.com/science/article/pii/S001999586780007X.

Fabio Massimo Zanzotto, Giorgio Satta, and Giordano Cristini. Cyk parsing over distributed representations. *Algorithms*, 13(10), 2020. ISSN 1999-4893. doi: 10.3390/a13100262. URL https://www.mdpi.com/1999-4893/13/10/262.

Fabio Massimo Zanzotto, Elena Sofia Ruzzetti, Giancarlo A. Xompero, Leonardo Ranaldi, Davide Venditti, Federico Ranaldi, Cristina Giannone, Andrea Favalli, and Raniero Romagnoli. Position paper: MeMo: Towards language models with associative memory mechanisms. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 15169–15180, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.785. URL https://aclanthology.org/2025.findings-acl.785/.

F.M. Zanzotto and L. Dell'Arciprete. Distributed tree kernels. In *Proceedings of International Conference on Machine Learning*, pp. 193–200, 2012. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84867126965&partnerID=40&md5=0d51c0ed7070baf730f887c818a8c177.

A THE SIMPLE GRAMMAR OF THE EXPERIMENTS

Grammar $G_0 = (T, NT, S, R_U \cup R_B)$ is very simple as current transformer-based LLMs are not able to model the neuro-symbolic task of performing CYK parsing. The grammar is defined as follows:

T	a,b,c
NT	S,A,B,C
R_U	A $\rightarrow$ a
	B $\rightarrow$ b
	C $\rightarrow$ c
R_B	S $\rightarrow$ A B
	S $\rightarrow$ B C
	A $\rightarrow$ B B
	B $\rightarrow$ C A
	C $\rightarrow$ C B

B A SMALL SAMPLE OF THE DATASET

Samples in the dataset appear as follows:

sequence	*Oracle CYK Matrix*
c c b a	(0,1,['C']) (0,2,[]) (0,3,[]) (0,4,['C']) (1,2,['C']) (1,3,['C']) (1,4,['B']) (2,3,['B']) (2,4,[]) (3,4,['A'])
c a c	(0,1,['C']) (0,2,['B']) (0,3,['S']) (1,2,['A']) (1,3,[]) (2,3,['C'])

C PROPERTIES OF HOLOGRAPHIC REDUCED REPRESENTATIONS

The starting point of a distributed representation and, hence, of HRR is how to encode symbols into vectors: symbol a is encoded using a random vector $\vec{a} \in \mathbb{R}^d$ drawn from a multivariate normal distribution $\vec{a} \sim N(0, \mathbf{I}\frac{1}{\sqrt{d}})$. These are used as basis vectors for the Johnson–Lindenstrauss transform (Johnson & Lindenstrauss, 1984), as well as for random indexing (Sahlgren, 2005). The main property of these random vectors is the following:

$$\vec{a}^T\vec{b} \approx \begin{cases} 1 & \text{if } \vec{a} = \vec{b} \\ 0 & \text{if } \vec{a} \neq \vec{b} \end{cases}$$

In this paper, we use the matrix representation of the shuffled circular convolution introduced in (Zanzotto & Dell'Arciprete, 2012). In this way, symbols are represented in a way where composition is just matrix multiplication and the inverse operation is matrix transposition. Given the above symbol encoding, we can define a basic operation $[\,]^{\oplus}$ and its approximate inverse $[\,]^{\ominus}$. These operations take as input a symbol and provide a matrix in $\mathbb{R}^{d\times d}$ and are the basis for our encoding and decoding. The first operation is defined as:

$$[a]^{\oplus} = \mathrm{A}_{\circ}\Phi \,,$$

where Φ is a permutation matrix to obtain the shuffling (Zanzotto & Dell'Arciprete, 2012) and $\mathrm{A}_{\circ}$ is the circulant matrix of the vector $\vec{a}^T = (a_0 \quad a_1 \quad \dots \quad a_{d-1})$. , that is:

$$\mathrm{A}_{\circ} = \begin{bmatrix} a_0 & a_{d-1} & \dots & a_1 \\ a_1 & a_0 & \dots & a_2 \\ \vdots & \ddots & \vdots & \\ a_{d-2} & a_{d-3} & \dots & a_{d-1} \\ a_{d-1} & a_{d-2} & \dots & a_0 \end{bmatrix} = \begin{bmatrix} | & | & \dots & | \\ s_0(\vec{a}) & s_1(\vec{a}) & \dots & s_{d-1}(\vec{a}) \\ | & | & \dots & | \end{bmatrix}$$

while $s_i(\vec{a})$ is the circular shifting of i positions of the vector $\vec{a}$. Circulant matrices are used to describe circular convolution. In fact, $\vec{a} * \vec{b} = \mathrm{A}_{\circ}\vec{b} = \mathrm{B}_{\circ}\vec{a}$ where $*$ is circular convolution. This operation has a nice approximated inverse in:

$$[a]^{\ominus} = \Phi^T\mathrm{A}_{\circ}^{\mathbf{T}} \,.$$

We then have:

$$[a]^{\oplus}[b]^{\ominus} \approx \begin{cases} \mathbf{I} & \text{if } \vec{a} = \vec{b} \\ \mathbf{0} & \text{if } \vec{a} \neq \vec{b} \end{cases}$$

since Φ is a permutation matrix and therefore $\Phi\Phi^T = I$ and since:

$$\mathrm{A}_{\circ}^{\mathbf{T}}\mathrm{B}_{\circ} \approx \begin{cases} \mathbf{I} & \text{if } \vec{a} = \vec{b} \\ \mathbf{0} & \text{if } \vec{a} \neq \vec{b} \end{cases}$$

due to the fact that $\mathrm{A}_{\circ}$ and $\mathrm{B}_{\circ}$ are circulant matrices based on random vectors $\vec{a}, \vec{b} \sim N(0, \mathbf{I}\frac{1}{\sqrt{d}})$; hence, $s_i(\vec{a})^T s_j(\vec{b}) \approx 1$ if both $i = j$ and $\vec{a} = \vec{b}$, and $s_i(\vec{a})^T s_j(\vec{b}) \approx 0$ otherwise. Finally, the permutation matrix Φ is used to enforce non-commutativity in the matrix product such as $[a]^{\oplus}[b]^{\oplus}[c]^{\oplus}$.

D Experiments with LLMs as CYK table generators

D.1 Few-shot prompting for CYK table generation

We adopt a standard few-shot prompting setup in which the model is given a fixed CNF grammar together with two input strings and their gold CYK tables, and is then asked to generate the CYK table for a new string in the same format. The prompt is reported below.

```
Task: Given a context-free grammar in Chomsky Normal Form (CNF) and an input string w,
compute the CYK table following the format of the examples and return it.

Grammar:
S -> A B
S -> B C
A -> B B
B -> C A
C -> C B
A -> a
B -> b
C -> c

Below some example of the matrix I want you to return. Follow the exact same format.

Example_1

String_1: "cacbbcca"
CYK Matrix : (0,1,['C']) (0,2,['B']) (0,3,['S']) (0,4,['S']) (0,5,['S','A']) (0,6,[]) (0,7,[]) (0,8,[])
                 (1,2,['A']) (1,3,[]) (1,4,[]) (1,5,['S']) (1,6,[]) (1,7,[]) (1,8,[])
                         (2,3,['C']) (2,4,['C']) (2,5,['B','C']) (2,6,['S']) (2,7,[]) (2,8,['S'])
                                 (3,4,['B']) (3,5,['A']) (3,6,[]) (3,7,[]) (3,8,[])
                                         (4,5,['B']) (4,6,['S']) (4,7,[]) (4,8,['S'])
                                                 (5,6,['C']) (5,7,[]) (5,8,['C'])
                                                         (6,7,['C']) (6,8,['B'])
                                                                 (7,8,['A'])

Example_2

String_2: "cababcac"
CYK Matrix: (0,1,['C']) (0,2,['B']) (0,3,['A']) (0,4,[]) (0,5,[]) (0,6,[]) (0,7,[]) (0,8,[])
                 (1,2,['A']) (1,3,['S']) (1,4,[]) (1,5,[]) (1,6,[]) (1,7,[]) (1,8,[])
                         (2,3,['B']) (2,4,[]) (2,5,[]) (2,6,[]) (2,7,[]) (2,8,[])
                                 (3,4,['A']) (3,5,['S']) (3,6,[]) (3,7,[]) (3,8,[])
                                         (4,5,['B']) (4,6,['S']) (4,7,['A']) (4,8,[])
                                                 (5,6,['C']) (5,7,['B']) (5,8,['S'])
                                                         (6,7,['A']) (6,8,[])
                                                                 (7,8,['C'])

Now compute:

String_3: "{current_string}"
CYK Matrix :
```

D.2 Instruction tuning on CYK supervision

To further evaluate whether explicit supervision improves algorithmic behaviour, we fine-tune instruction-following models on a CYK training set. We preprocess each example into an instruction–

input–output format, where the `instruction` specifies the fixed CNF grammar, the `input` contains a query asking for the CYK matrix of a given string, and the `output` consists of the corresponding gold CYK table serialized in the target linearized format. An example is shown below:

```
{
  'instruction': 'Compute the CYK matrix for the grammar:
                  S -> A B
                  S -> B C
                  A -> B B
                  B -> C A
                  C -> C B
                  A -> a
                  B -> b
                  C -> c,

  'input': 'Tell me the CYK matrix of cacbaccbcba',

  'output': "(0,1,['C']) (0,2,['B']) (0,3,['S']) (0,4,['S']) (0,5,['A']) (0,6,[]) (0,7,[]) (0,8,[]) (0,9,[]) (0,10,[]) (0
             (1,2,['A']) (1,3,[]) (1,4,[]) (1,5,['S']) (1,6,[]) (1,7,[]) (1,8,[]) (1,9,[]) (1,10,[]) (1,11,
             (2,3,['C']) (2,4,['C']) (2,5,['B']) (2,6,['S']) (2,7,[]) (2,8,[]) (2,9,[]) (2,10,[]) (2,11,['S'
             (3,4,['B']) (3,5,[]) (3,6,[]) (3,7,[]) (3,8,[]) (3,9,[]) (3,10,[]) (3,11,[])
             (4,5,['A']) (4,6,[]) (4,7,[]) (4,8,[]) (4,9,[]) (4,10,[]) (4,11,[])
             (5,6,['C']) (5,7,[]) (5,8,[]) (5,9,[]) (5,10,[]) (5,11,['C'])
             (6,7,['C']) (6,8,['C']) (6,9,[]) (6,10,[]) (6,11,['B','C'])
             (7,8,['B']) (7,9,['S']) (7,10,['S']) (7,11,['A'])
             (8,9,['C']) (8,10,['C']) (8,11,['B'])
             (9,10,['B']) (9,11,[])
             (10,11,['A'])"
}
```

These serialized instruction examples are then used to fine-tune medium-sized instruct models with LoRA, allowing the models to adapt to the CYK table-generation task while keeping the number of trainable parameters limited. At test time, evaluation is performed in a zero-shot setting, where the fine-tuned model receives only the `instruction` and the `input`, and is required to generate the CYK matrix without any additional demonstrations.