Where to Approximate in Neurosymbolic Inference?

Samy Badreddine[1,2,3] **Emile van Krieken**[4] **Luciano Serafini**[2,3] **Antonio Vergari**[5]
[1]Sony AI [2]Fondazione Bruno Kessler [3]University of Trento
[4] Vrije Universiteit Amsterdam [5]University of Edinburgh
samy.badreddine@sony.com e.van.krieken@vu.nl
luciano.serafini@fbk.eu avergari@ed.ac.uk

Abstract

Probabilistic neurosymbolic methods rely on weighted model counting (WMC) to combine neural predictors with symbolic constraints. Exact WMC is #P-hard and typically scales poorly, so many methods resort to approximations. By unifying existing approaches under a three-step bottom-up *approximate compilation* pipeline, we study *where* approximation budget is best allocated. Inspired by tensor networks, we instantiate two steps with tensor train decompositions and derive new pipelines that exactly multiply approximated factors and recompress their products via SVD- or interpolation-based schemes. Across six tasks in Sudoku, arithmetic reasoning, and scene understanding, and across increasing problem sizes, our pipelines produce WMC estimates orders-of-magnitude more accurate than prior methods. This translates into stronger downstream learning on tasks where reasoning relies on tight interactions between symbolic factors, where cruder approximations ignore these dependencies. Our pipelines open novel design space for effective approximate compilation of challenging probabilistic reasoning tasks.

1 Introduction

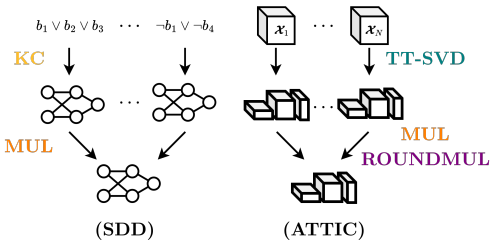
Probabilistic neurosymbolic (NeSy) systems combine neural perception with symbolic constraints, to enable learning from weak supervision [34, 67] or to inject symbolic knowledge into structured prediction pipelines [1]. Inference involves solving the *weighted model count* (WMC) problem [7]: computing the probability that a neural distribution over discrete concepts satisfies a constraint. Computing the WMC exactly is #P-hard in general [56]. One approach for solving practical instances tractably is *circuit compilation* which compiles the constraint into a logical circuit, a process also called *knowledge compilation* [15], while the neural distribution is turned into a probabilistic circuit [63, 9, 1]. Then, under certain structural properties, these circuits can be efficiently multiplied and summed over to yield the exact WMC [51, 64]. The same structural properties and multiplication operation are also behind popular bottom-up strategies for knowledge compilation [46, 14].

However, compilation is itself hard, and exactly multiplying several compiled circuits blows up exponentially in the number of factors [64]. This implies that compilation strategies can hit a wall on many real-world NeSy problems. Even compiling a 9×9 Sudoku board, often used in benchmarks [4, 66], as a constraint is out of reach for all exact compilers. Without surprise, many NeSy methods focused on approximate computation of the WMC [59, 53, 8, 16]. However, these approaches have carried over with different perspectives and from different communities. As such, the problem of understanding *where* and *when* to approximate in a systematic way is still open.

In this paper, we provide a unified view over many NeSy pipelines and we identify three stages at which approximations can enter (Figure 1): **(I)** compiling each factor exactly or with a relaxed circuit, **(II)** multiplying factors exactly or via a recompression that controls size growth, and **(III)** combining remaining factors under an independence assumption. Many existing methods fit in this categorization, sometimes mixing several approximations. *Independence* (fuzzy) approximations are by far the most

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Baseline	SI	SII	SIII
Exact methods [14, 67]	KC	MUL	–
Fuzzy methods [5, 58]	–	–	INDEP
Sem Strength [2]	KC	MUL	INDEP
CTSKETCH [8]	TT-SVD	–	INDEP
APRICOT A	PREMUL, TT-SVD	MUL	INDEP
APRICOT B	PREMUL, TT-SVD	MUL, ROUNDMUL	INDEP
APRICOT C	PREMUL, TT-SVD	MUL, ROUNDMUL	–



Exact operations: Approximate operations:

Figure 1: **Where can we approximate** when compiling structured-decomposable circuits? We unify bottom-up approaches in a three-step pipeline: **(Step I)** compiles each factor ϕ_j into an exact or **relaxed** circuit, **(Step II)** multiplies the resulting N circuits down to K **exactly** or using **recompression strategies**, and **(Step III)** combines any $K > 1$ remaining circuits with an **independence** (fuzzy) approximation. Existing methods correspond to specific combinations of exact or approximate operations, but the framework also allows for new TT-based APRICOT strategies yet to be explored.

common [5, 58, 2] and are well known to support effective learning [58, 32], yet their inference fidelity is not well-studied. Stages (I) and (II) are far less explored. Drawing on the close connection between tensor networks and circuits [30], we instantiate stages (I) and (II) using tensor train (TT) decompositions. TTs have been used in NeSy to approximate and compress isolated factors [8] but not in a full compilation pipeline. We show how approximate tensor trains can be used for inference and compilation, deriving novel **APRICOT** pipelines.

Contributions. **(i)** We unify existing bottom-up compilation methods under a three-stage approximate-compilation pipeline and analyze how error and cost compose across strategies. **(ii)** Within this framework, we derive new approximation strategies based on TN recompressions, previously unexplored in NeSy or compilation. **(iii)** We map the compilation design space on Sudoku and evaluate weighted inference and learning on arithmetic and scene understanding tasks. We show that APRICOT pipelines scale beyond exact compilation, produce WMC estimates orders of magnitude more accurate than prior methods by preserving dependencies, which also translates to better learning in programs where factors are structurally tightly connected. Code and instructions for reproducing all experiments are available at github.com/sbadredd/apricot.

2 Probabilistic NeSy Models

A probabilistic NeSy model combines (i) a parameterized (neural) model $p_{\boldsymbol{\theta}}(\mathbf{c} \mid \mathbf{x})$ which maps the input $\mathbf{x}$ to a vector of discrete, symbolic *concepts* $\mathbf{c} = [c_i]_{i=1,\ldots,d}$, $c_i \in \mathcal{C}_i$, with (ii) a symbolic *constraint* $\phi : \mathcal{C}_1 \times \cdots \times \mathcal{C}_d \to \{0, 1\}$ that assesses the validity of the predicted concepts. In Sudoku, for example, $\mathbf{x}$ is the image of the board, $\mathbf{c}$ are the predicted digits in each cell, and ϕ checks that the predicted digits satisfy the Sudoku rules. This involves computing $p_{\boldsymbol{\theta}}(\phi|\mathbf{x})$ which maps the input $\mathbf{x}$ to the probability of satisfying the constraint ϕ. This is known as a conditional *weighted model count* (WMC) problem, defined as [1]

$$p_{\boldsymbol{\theta}}(\phi \mid \mathbf{x}) := \sum\nolimits_{\mathbf{c} \in \mathcal{C}} p_{\boldsymbol{\theta}}(\mathbf{c} \mid \mathbf{x})\, \phi(\mathbf{c}). \tag{1}$$

The problem can also be stated with $\phi(\mathbf{c}, \mathbf{y})$ being a program that maps concepts $\mathbf{c}$ to a supervised structured output $\mathbf{y}$, in which case the WMC derives $p_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x})$ (sometimes up to a normalization constant, see Section H.1 for details).

Often, ϕ factorizes into a conjunction of subformulas $\phi_1(\mathbf{c}_1) \wedge \cdots \wedge \phi_N(\mathbf{c}_N)$, where each factor ϕ_j has scope $\mathbf{c}_j \subseteq \mathbf{c}$. For example, the 9×9 Sudoku rules decompose into $N = 810$ factors that check pairwise constraints $c_i \neq c_j$ for cells i, j in the same row, column, or box. Programs may additionally

[1] The problem can also be stated with $\phi(\mathbf{c}, \mathbf{y})$ being a program that maps concepts $\mathbf{c}$ to a structured output $\mathbf{y}$, in which case the WMC derives $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$. See the *Sudoku completion* example in Section 5.

introduce *auxiliary symbolic variables*, unweighted and unsupervised, to derive the factorization. For example, the arithmetic rules of addition, often explored in NeSy [11, 31], factorize using auxiliary carries between digits. This is further discussed in Section H.1.

Computing the WMC is #P-hard in general [56], and this is also due to the large combinatoric space of NeSy problems. E.g., in a 9×9 Sudoku there are 9^{81} possible configurations, out of which *only* 6.67×10^{21} are valid boards [19]. To solve this issue, probabilistic NeSy methods must use compilation techniques and/or relaxation strategies to make the WMC tractable.

2.1 Exact circuit compilation

A prominent approach to computing the WMC tractably is to represent both ϕ and $p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})$ as circuits and combine them [27, 34, 67, 1]. A circuit is a directed acyclic computational graph of *weighted sum* and *product operations* over tractable input functions [9, 64]. We focus on *smooth* and *decomposable* circuits, which support marginalization in time linear in their size. For tractable multiplication, circuits additionally need compatible decomposition structures: a circuit is *structured-decomposable* when its product units respect a common hierarchical partition of the variables (called a vtree), and two such circuits are *compatible* when these decompositions agree on shared variables [64]. Compatible circuits can be multiplied in polynomial time while preserving smoothness and decomposability. We defer formal definitions to Section A.

Knowledge compilation (KC) transforms a logical constraint ϕ into a logical circuit, typically expressed using AND/OR gates [15]. For deterministic circuits, whose OR children have disjoint supports, AND/OR gates can be interpreted as product/sum operations without changing the represented Boolean function [25]. A deterministic compiled circuit for ϕ can be multiplied tractably with a compatible probabilistic circuit $p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})$.

Sentential decision diagrams (SDDs) [14] are a popular class of deterministic, structured-decomposable logical circuits. They can also be constructed bottom-up by **multiplying** circuit representations of factors ϕ_j. Let $\mathrm{KC}(\phi_j)$ denote the circuit obtained by compiling factor ϕ_j. We express bottom-up compilation using MUL:

$$\sum\nolimits_{\mathbf{c}\in\mathcal{C}} p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})\mathrm{MUL}\big(\mathrm{KC}(\phi_1),\ldots,\mathrm{KC}(\phi_N)\big)(\mathbf{c}). \tag{2}$$

This is the classical recipe behind many approaches such as semantic loss [67] or DeepProbLog [34].

The main limitation is constructing a sufficiently compact circuit for ϕ with the required structural properties. KC is at least NP-hard [15, 7], and, to the best of our knowledge, existing compilers cannot produce a polynomial-size circuit in reasonable time for a 9×9 Sudoku instance, even when relaxing structured decomposability [12, 28]. [2] Therefore, a common practice is to consider *approximations*. We focus on approximations which (i) apply to bottom-up circuit compilation, (ii) are differentiable. In Section B, we discuss other approaches to WMC in NeSy, such as amortized or sampling-based methods, and how they relate to our work.

2.2 Approximations

Independence approximations. A common approximation when multiplying two circuits is to assume they are independent, even if they are defined over the same set of variables. If we assume that, the WMC of their conjunction factorizes into the product of their WMCs. More formally,

$$\sum\nolimits_{\mathbf{c}\in\mathcal{C}} p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})\mathrm{MUL}(\mathrm{KC}(\phi_1),\ldots,\mathrm{KC}(\phi_N))(\mathbf{c}) \overset{\mathrm{I}}{\approx} \prod\nolimits_j \left(\sum\nolimits_{\mathbf{c}_j\in\mathcal{C}_j} p_{\boldsymbol{\theta}}(\mathbf{c}_j|\mathbf{x})\mathrm{KC}(\phi_j)(\mathbf{c}_j)\right). \tag{3}$$

Often, $|\mathcal{C}_j| \ll |\mathcal{C}|$, which makes the factor WMCs much easier to compute. *Fuzzy* NeSy approaches with product semantics [5, 17, 58] can be understood as probabilistic approaches relaxing *all* dependencies [57]. [3] The approximation is only correct when the factors ϕ_j are independent, i.e., when

[2] Current boolean KC software also commonly requires categorical variables to be propositionalized. This requires introducing mutual exclusivity constraints which further increases the size of the formula and complicates compilation. We discuss the resulting encoding overhead in Section A.

[3] In the rest of the paper, we sometimes refer to the independence approximation as "fuzzy", though it is important to note that not all fuzzy approaches use product semantics understood as an independence relaxation of probabilistic NeSy.

the scopes $\mathbf{c}_j$ are pairwise disjoint. In general, this is not the case and the approximation can be arbitrarily bad, as we will show in our experiments. Still, in the context of differentiable learning, fuzzy relaxations are known to be effective in practice [58].

Fuzzy relaxations can be used in combination with exact circuit compilation. Semantic Strengthening [2] compiles SDD circuits for factor groups under a certain budget, prioritizing groups of factors with high mutual information, and applies the independence approximation between groups. SDDs are particularly suitable for this approach as they can iteratively multiply and analyze factors, making informed compilation or stopping decisions within the budget.

Factor approximations. Instead of approximating the marginalization step, one can approximate the factors themselves. This requires approximate compilation (AC) step, i.e.,

$$\text{MUL}(\text{KC}(\phi_1), \ldots, \text{KC}(\phi_N)) \overset{\text{F}}{\approx} \text{MUL}(\text{AC}(\phi_1), \ldots, \text{AC}(\phi_N)) \tag{4}$$

where AC compiles an *approximated circuit* in the real semiring for factor ϕ_j, i.e., a circuit that does not encode a Boolean function but closely approximates it up to an ϵ-error. Crucially, the circuit retains all the structural properties needed to tractably compute MUL. In Section 3, we discuss how each factor can be approximated with *tensor trains* (TTs) [43], a special family of tensorized circuits [30] for which there is a rich literature on function approximation [41]. Such a TT-approximated factor $\text{AC}(\phi_j)$ can be a representation that is significantly smaller than the exact circuit $\text{KC}(\phi_j)$, given that it can introduce negative parameters [29]. Additionally, TTs inherently handle categorical variables and do not need Boolean encoding.

CTSKETCH [8] is a recent approach that approximates factors ϕ_j with low-rank TTs. However, CTSKETCH only considers programs that follow a certain tree-like independence structure, where any variable c_i can appear in at most one factor. This applies to simple problems such as MNISTADD [34], but most programs, like Sudoku boards, have overlapping factors and violate this assumption. Therefore, CTSKETCH mixes factor approximations with fuzzy independence approximations $\prod_j (\sum_{\mathbf{c}_j} p_{\boldsymbol{\theta}}(\mathbf{c}_j|\mathbf{x})\text{AC}(\phi_j)(\mathbf{c}_j))$. Section D details this. In Section 3, we show how to exactly multiply TT-approximated factors under more complex dependence structures.

Product approximations. Finally, we consider approximating the multiplication step directly, without going through intermediate approximations of the factors themselves, i.e.,

$$\text{MUL}(\text{KC}(\phi_1), \ldots, \text{KC}(\phi_N)) \overset{\text{P}}{\approx} \text{ROUNDMUL}(\text{KC}(\phi_1), \ldots, \text{KC}(\phi_N)). \tag{5}$$

Drawing on TN theory, we show that it is possible to derive algebraically-principled approximations which compress the product of circuits into smaller circuits. We denote such operations ROUNDMUL for *rounded multiplications*, inspired by the TT-ROUND [43] algorithm discussed in Section 3.3. This direction is, to the best of our knowledge, unexplored in the NeSy or compilation literature, and can prove useful to approximate many operations on circuits.

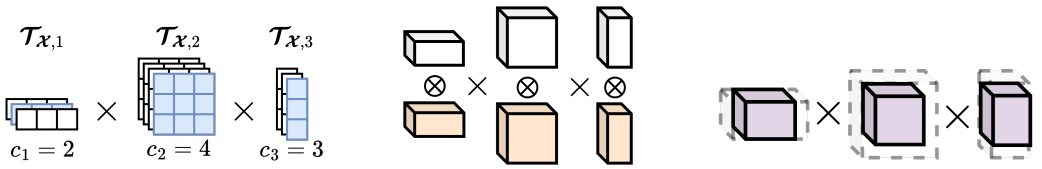
3 Approximate Tensor Trains...

The tensor perspective applies to WMC by representing the constraint ϕ as a Boolean-valued tensor $\boldsymbol{\mathcal{X}} \in \{0,1\}^{n_1 \times \cdots \times n_d}$, where $n_i = |\mathcal{C}_i|$ and $\boldsymbol{\mathcal{X}}(\mathbf{c}) := \phi(\mathbf{c})$. This is an order-$d$ tensor: each mode corresponds to one variable c_i, and its size n_i is the number of values that variable can take. Thus, each tensor entry records whether a particular assignment satisfies the constraint. The dense representation contains $\prod_{i=1}^{d} n_i$ entries, which quickly becomes infeasible to materialize as the number of variables grows. For example, the full 9×9 Sudoku constraint has $d = 81$ variables, each with 9 possible values, giving an order-81 tensor with 9^{81} entries.

Tensor trains (TTs) [43, 45] represent such tensors through a sequence of smaller tensors. They are a special case of tensor networks and can also be viewed as structured-decomposable circuits [30]. A TT represents an approximation $\tilde{\boldsymbol{\mathcal{X}}}$ of an order-d tensor $\boldsymbol{\mathcal{X}}$ using one *core* per mode:

$$\boldsymbol{\mathcal{X}}(c_1, \ldots, c_d) \approx \tilde{\boldsymbol{\mathcal{X}}}(c_1, \ldots, c_d) = \boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},1}(c_1) \cdots \boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},d}(c_d).$$

Each core $\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},k}$ is an order-3 tensor of shape $r_{k-1} \times n_k \times r_k$. Fixing the variable value c_k selects a matrix slice $\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},k}(c_k) := \boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},k}(:, c_k, :)$. The dimensions r_k are the *TT ranks*, with $r_0 = r_d = 1$ so that multiplying all selected matrices returns a scalar. Writing $n = \max_k n_k$ and $r = \max_k r_k$,



(a) $\mathcal{X}(2, 4, 3)$ (b) $\mathcal{X}_j * \mathcal{X}_l$ (c) TT-ROUND

Figure 2: Closed algebra over TTs. TTs (a) can **approximate discrete functions** under an error criterion and (b) are closed under several **exact operations** including multiplications. As multiplications involve Kronecker products $\otimes$ of the cores resulting in an exponential increase in rank, algebraic recompression strategies (c) can **approximate the result of the operations**.

TT storage is $\mathcal{O}(dnr^2)$, compared with up to $\mathcal{O}(n^d)$ entries in the dense tensor. The representation is therefore compact when the ranks remain small. We now use this representation to approximate individual constraint factors, multiply the resulting TTs, and recompress their products.

3.1 TT Factor Approximations

A factor ϕ_j with scope $\mathbf{c}_j$ of size d_j corresponds to an order-d_j tensor $\mathcal{X}_j$. For example, each pairwise Sudoku constraint $c_i \neq c_l$ is an order-2 tensor: a 9×9 matrix with zeros on the diagonal and ones elsewhere. While the full constraint tensor is generally too large to materialize, these factor tensors are often small enough to construct explicitly and approximate using TT-SVD.

We approximate each dense factor tensor using TT-SVD [43], as in CTSKETCH [8]. The algorithm constructs a TT $\tilde{\mathcal{X}}_j$ under either a prescribed Frobenius error tolerance $\|\mathcal{X}_j - \tilde{\mathcal{X}}_j\|_F \leq \epsilon_j$ or a maximum TT rank R. For prescribed ranks, TT-SVD is quasi-optimal: its error is within a factor $\sqrt{d_j - 1}$ of the best approximation with those ranks. Importantly, TT-SVD requires its dense input to be materialized, which motivates applying it to the small factor tensors rather than to the full constraint.The resulting TTs must then be combined (i.e., multiplied) while accounting for dependencies between their overlapping scopes.

3.2 Exact multiplication of TTs

TTs are closed under element-wise (Hadamard) multiplication, which allows us to compile TT-approximated factors without introducing new approximations. The operation requires matching scopes and mode orderings. We fix a global variable ordering and arrange each factor's modes accordingly. Before multiplying factors with different scopes, we *broadcast* their TTs to the union of their scopes by inserting identity cores for missing variables. The construction and a heuristic for choosing the global ordering are detailed in Section F.2.

Consider two TTs $\mathcal{X}_j$ and $\mathcal{X}_l$ after broadcasting to a common scope of q variables, locally indexed $c_1, \ldots, c_q$. Their product $\mathcal{X}^* = \mathcal{X}_j * \mathcal{X}_l$ has cores

$$\mathcal{T}_{\mathcal{X}^*,k}(c_k) = \mathcal{T}_{\mathcal{X}_j,k}(c_k) \otimes \mathcal{T}_{\mathcal{X}_l,k}(c_k), \qquad k = 1, \ldots, q, \tag{6}$$

where $\otimes$ denotes the matrix Kronecker product. This is a special case of the MUL operation for compatible structured-decomposable circuits which maintains a TT structure. Because of the Kronecker, the product ranks are $r^*_k = r^j_k r^l_k$. Thus, two TTs with ranks bounded by r can produce a TT with ranks up to r^2 and storage $\mathcal{O}(qnr^4)$. Repeatedly multiplying m such TTs can require $\mathcal{O}(qnr^{2m})$ storage in the uncompressed representation. This matches the worst-case exponential complexity of circuit multiplication [64]. Exact multiplication therefore preserves dependencies between factors, but its rank growth motivates recompression.

3.3 Approximate multiplications of TTs

To control rank growth, we approximate the product $\mathcal{X}^*$ by a TT with a target maximum rank R. Unlike factor approximation with TT-SVD, this *product recompression* operates directly on the input TTs, without materializing the dense product. Depending on the method, the recompression can

alternatively be controlled by a truncation tolerance ϵ. We consider four strategies, distinguished by their compression mechanism (SVD or interpolation) and their use of deterministic or randomized compression directions. Algorithmic details are deferred to Section I.

TT-ROUND [43] *(SVD, deterministic/site-randomized)* uses successive SVD-based truncations to reduce TT ranks. For Hadamard products, we exploit the Kronecker structure to interleave products and compressions without materializing the Kronecker product cores. We consider both deterministic and randomized per-site SVDs [23].

HATT [55] *(SVD, randomized)* uses a random TT sketch of the product to identify compression subspaces, then applies SVDs to small sketched matrices. Its randomization acts at the TT level, rather than independently within each local SVD.

TCI [39, 42] *(interpolation, deterministic)* constructs a TT approximation from selected entries of the product, choosing representative rows and columns through maxvol-based cross interpolation [20].

RSI [38] *(interpolation, randomized)* combines random sketching with interpolative decompositions. It replaces TCI's maxvol-based initialization with sketched environments built from the input TTs. We extend its original single left-to-right sweep to alternating left-to-right and right-to-left sweeps, using the connection to TCI (details in Section I.4).

3.4 From tensor error to WMC error

The Frobenius guarantees provided by TT-SVD and SVD-based recompression translate to a bound on the final inference error. Let $\tilde{\boldsymbol{\mathcal{X}}}$ approximate the full constraint tensor $\boldsymbol{\mathcal{X}}$, with $\|\tilde{\boldsymbol{\mathcal{X}}} - \boldsymbol{\mathcal{X}}\|_F \leq \epsilon$, and let $\boldsymbol{\mathcal{P}}(\mathbf{c}) = p_{\boldsymbol{\theta}}(\mathbf{c} \mid \mathbf{x})$. Since the WMC $p_{\boldsymbol{\theta}}(\phi|\mathbf{x})$ in Equation (1) is the Frobenius inner product of these tensors, Cauchy-Schwarz gives

$$\left|\mathrm{W}\,\widehat{\mathrm{MC}} - \mathrm{WMC}\right| \leq \epsilon\,\|\boldsymbol{\mathcal{P}}\|_F \leq \epsilon. \tag{7}$$

The last inequality follows because $\boldsymbol{\mathcal{P}}$ is a normalized probability tensor. More details are given in Section C, including how the factor errors ϵ propagate under multiplications.

4 ...for Inference and Compilation

We now combine these ingredients into the TT-based framework APRICOT: **AP**proximate **R**epresentations for **I**nference and **CO**mpilation using **T**ensor trains. We describe instantiations of this framework within a unified bottom-up pipeline that also accommodates exact circuit compilation and some pre-existing approximation strategies (Figure 1). Given a constraint $\phi = \phi_1 \wedge \cdots \wedge \phi_N$, the pipeline has three stages: compiling factors, multiplying their circuit representations, and evaluating the remaining circuits. The choices at each stage decide whether **factor**, **product**, and/or **independence** approximations are involved.

Step I: Factor compilation. Each factor is represented by a circuit. Exact compilation constructs $\mathrm{KC}(\phi_j)$, for which we use SDDs [14]. Alternatively, APRICOT applies **TT-SVD** to the dense factor tensor $\boldsymbol{\mathcal{X}}_j$, with an initial rank budget R_{init} or a Frobenius error tolerance (Section 3.1). Before TT decomposition, small factors may optionally be grouped and **pre-multiplied** exactly in dense tensor space. This captures dependencies within each group before introducing a factor approximation, as long as the grouped tensor fits in memory. Let M denote the number of circuits resulting from this stage. Without grouping, $M = N$. All circuits use compatible decomposition structures (in the case of TTs, variable orderings) so that they can subsequently be multiplied.

Step II: Multiplication ($M \to K$). The M circuits are reduced to K circuits through pairwise multiplications, where $1 \leq K \leq M$. **Exact multiplication** uses SDD Apply for SDDs or the TT product of Section 3.2. For APRICOT, the propagation of existing factor errors is analyzed in Section C. To control the growth of intermediate TTs, APRICOT can instead use **recompressed multiplication**, with any of the strategies introduced in Section 3.3. Recompression introduces additional approximation error in exchange for smaller representations. The multiplication schedule, recompression strategy, and stopping criterion are all parameters of the pipeline. For example, K may be prescribed or determined by the available resources and approximation conditions. Our experimental choices are specified in Section 5.

Step III: Final independent reduction ($K \to 1$). Let $\tilde{\phi}_1, \ldots, \tilde{\phi}_K$ be the remaining circuits, with scopes $\mathbf{c}_1, \ldots, \mathbf{c}_K$. We compute their individual WMCs and combine them as

$$\widehat{p}_{\boldsymbol{\theta}}(\phi \mid \mathbf{x}) = \prod_{k=1}^{K} \left(\sum_{\mathbf{c}_k \in \mathcal{C}_k} p_{\boldsymbol{\theta}}(\mathbf{c}_k \mid \mathbf{x})\, \tilde{\phi}_k(\mathbf{c}_k) \right),$$

where $\mathcal{C}_k$ is the assignment space of the k-th residual scope. For $K > 1$, this applies the **independence** approximation between remaining factors (Equation (3)). The introduced error is difficult to predict and depends on the specific factors remaining after Step II (Section E).

Figure 1 maps existing approaches and new APRICOT variants as different choices in each step of the pipeline. Exact compilation and multiplication to $K = 1$ is exact inference, while stopping earlier gives the grouped independence strategy used, for example, by Semantic Strengthening. TT factor approximation followed directly by independent reduction gives the CTSKETCH-style approximation for overlapping factors, a special case of APRICOT. Variants can additionally combine them: APRICOT A combines TT factors through exact multiplications before an independence reduction, APRICOT B additionally uses product recompression up to a budget, while APRICOT C avoids the final independence approximation. Notice that each row in Figure 1 identifies a family rather than a fixed configuration: ranks, multiplication schedules, and reduction targets can vary across tasks. In particular, we detail different multiplication schedules we evaluate in Section G.2.

5 Experiments

We first explore the compilation design space on Sudoku, a popular NeSy benchmark [4, 66] with a complex dependence structure, and evaluate in-depth where to introduce approximation during compilation. We then compare representative pipelines on arithmetic and scene-understanding tasks of increasing difficulty. Finally, we address whether accurately compiling dependencies improves neurosymbolic learning across these settings.

Tasks. For the compilation ablation, we use Sudoku boards of size 4×4, 6×6, and 9×9, where known model counts derived analytically [19] allow us to evaluate unweighted approximation quality. For learning, we use two Visual Sudoku tasks. **ViSudo-Classification** predicts whether a board of MNIST images satisfies the Sudoku rules, using the dataset from Augustine et al. [4]. **ViSudo-Completion** predicts missing cells from the observed digit images, using the SatNet dataset [66] with the data-leak correction of Chang et al. [6]. We additionally evaluate four tasks for both weighted inference and learning. **MNISTAdd** [34] and **MNISTMul** predict the sum or product of two L-digit numbers represented by MNIST images, with supervision only on the result. **CLEVR-Diff** predicts whether all objects in a CLEVR scene [24] have different attribute tuples (color, shape, ...), while **CLEVR-CountDiff** predicts the number of distinct attribute tuples in the scene. We construct the CLEVR tasks reusing scenes from Stammer et al. [54].

An important difference between these tasks is how their programs factorize. In particular, the arithmetic and distinct-counting programs introduce *auxiliary symbolic variables* such as carries and running counts to express the constraint with smaller factors (see Section H.3 for details). In comparison, our Sudoku and CLEVR-Diff factorizations directly compare perceptual concepts and do not introduce such intermediate symbols. This will prove meaningful when analyzing the NeSy learning results.

Baselines. We use the family names introduced in Figure 1. The Sudoku compilation ablation explores many configurations within these families, while the learning experiments evaluate only selected representatives. On the arithmetic and CLEVR tasks, we compare three strategies: full independence reduction (LTN/DFL-style), full exact compilation (Semantic Loss/DeepProbLog-style), and full TT compilation allowing product recompression (APRICOT C). As explained in Section 4, the names identify families; the exact configurations (ranks, multiplication schedules) depend on the specific task programs and are detailed in Section H.5.

We implement all TT algorithms in PyTorch [44], use PySDD [37] for SDD compilation, and KLay [33] for GPU inference on SDDs. Results are averaged over ten runs with errors bars for standard deviation. Hardware and training protocols are given in Section H.5.

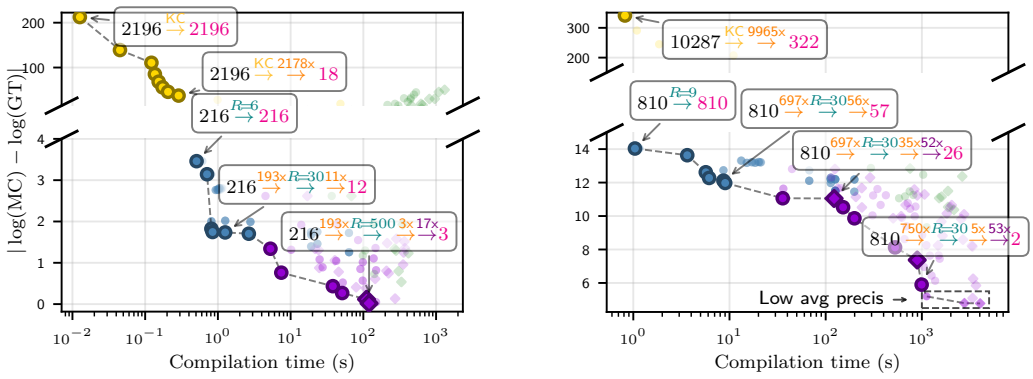


Figure 3: Pareto front of absolute log inference error vs. compilation time for Sudoku 6x6 (left) and 9x9 (right). SDD pipelines are in yellow ●. APRICOT pipelines are in blue ● when they do not use recompressions, in violet when they use SVD-based recompressions (TT-ROUND ● and HATT ◆) and in green for interpolation-based strategies (TCI ● and RSI ◆).

5.1 Sudoku compilation ablation

Figure 3 compares compilation time and model-count error on the larger boards. We vary the number and type of the multiplications (exact or recompressed) as well as the rank for the initial TT-SVD. Each point in the plot is a candidate compilation pipeline; details on our grid search and on the tags we use to identify points are in Section H.2. Below, we highlight trends in the results.

We measure absolute errors on a logarithmic scale $|\log_{10} \mathrm{MC}/\mathrm{MC}_{\mathrm{ref}}|$, with zero indicating an exact count. The model count estimates are obtained by calculating the WMC under a uniform distribution, which gives the probability of an assignment being a model, multiplied by the size of the assignment space. The ground-truth counts are 288, 28,200,960, and approximately 6.67×10^{21} for the three board sizes. 4×4 serves as a sanity check, with both exact SDD compilation and suitable TT configurations recovering its count to numerical precision (Section H.2.3). 6×6 is already challenging: compilation is only possible with the best *non*-structured-decomposable compilers like d4 [28], not with SDDs. To the best of our knowledge, 9×9 is unsolved by any existing compilers.

On independence approximations. As expected from our error analysis and discussion on fuzzy bounds (Section C), the *fuzzy approximations of the WMC have unpredictable behaviors* which vary with the encoding used. Even at matched K, SDD (yellow) and APRICOT behave very differently. In practice, the MC estimate of SDD+Fuzzy is approximately zero, making it hardly usable.

On exact multiplications. Combining factors before independence reduction always *significantly improves fidelity*. However, both SDD size and TT ranks limit how far exact multiplication scales. **Factor** approximations can compress initial factors, but their products still grow exponentially.

On approximate multiplications. Recompression allows further multiplications and provides the strongest accuracies. The *SVD-based strategies are more reliable* than the interpolation-based alternatives, with no significant difference between randomization strategies. In both cases, however, attempting too many multiplications under a limited rank budget can also degrade the representation because of expressivity limitations. To distinguish count accuracy from constraint fidelity, we additionally evaluate how well each circuit separates satisfying assignments from nonsatisfying assignments. We measure this using average precision on 1000 model and 1000 non-model sudoku samples, evaluating the product of the current factors indexed on each sample. On 9×9, we find points (marked "low AP") which, after much recompression, achieve good aggregate error but miss the actual models. This highlights the importance of halting compilation based on the accumulated Frobenius error bound (Section C) once it exceeds a certain tolerance.

5.2 Weighted inference

We next evaluate whether the compilation strategies remain effective when querying the circuits with nonuniform weights. We compare circuit estimates with exact compilation or task-specific

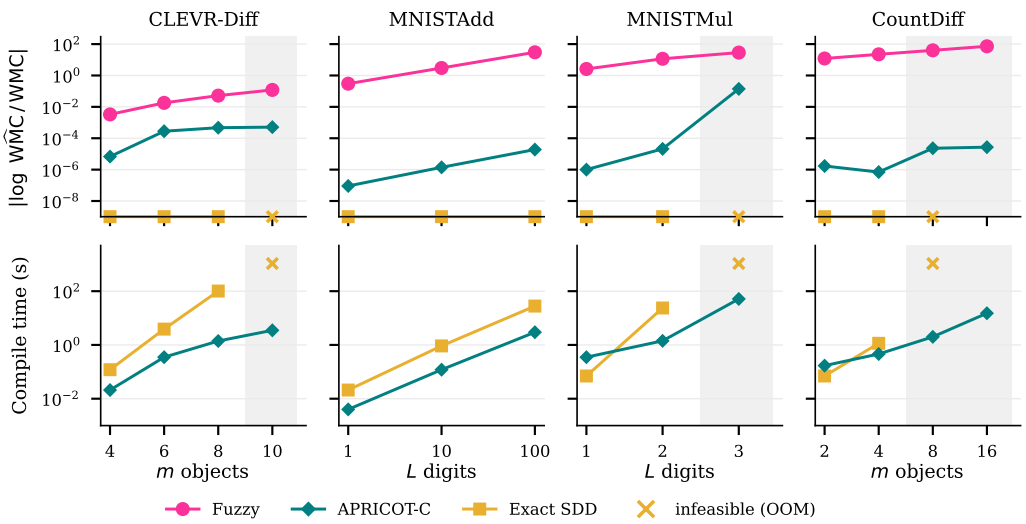


Figure 4: Median absolute log WMC error and compilation time across tasks and problem sizes. Exact SDD compilation becomes infeasible on harder problems. Full TT reduction (APRICOT C) can scale further while producing estimates orders of magnitude more accurate than fuzzy.

exact references under 64 shared random weightings. For each weighting, we compute the absolute base-10 log-ratio between the estimated and reference WMC, and report the median of these 64 errors alongside compilation time (Figure 4). That is, a reported error of 1 means that, for a typical sampled weighting, the estimated WMC differs from the reference by a factor of 10. Details are specified in Section H.4.

On MNISTAdd, both SDDs with a suitable vtree and APRICOT exploit the carry-chain structure to scale efficiently with the number of digits. On the more challenging CLEVR-Diff, MNISTMul, and CLEVR-CountDiff, APRICOT C scales beyond the evaluated SDD compiler. At the same time, its WMC estimates are substantially more accurate than those obtained by independence reduction. Independence reduction is inexpensive at the cost of inference fidelity, but TT recompression provides a practical trade-off, retaining these dependencies approximately in a manageable representation.

5.3 Learning

Previous works have found that faithful inference is not always a requirement for competitive downstream learning. Indeed, despite ignoring dependencies, fuzzy systems provide good differentiability in many use cases [5] and can behave surprisingly well as gradient estimators [32]. We explore whether this holds across more complex settings.

We measure the concept accuracy as the fraction of individual concepts $\mathbf{c}$ predicted correctly (digits or object keys) and the output accuracy $\mathbf{y}$ as a per-instance exact accuracy (e.g., all missing cells must be correct for Sudoku completion, all output digits must be correct for arithmetic). Notice that output accuracy can decline with problem size even when individual concepts and reasoning remain accurate: for two inputs with L digits, independent digit errors at 99% accuracy gives a probability of all inputs being correct 0.99^{2L} (about 13% at $L = 100$), which is necessary for per-instance output accuracy. For ViSudo-Classification and both CLEVR tasks, we evaluate concept accuracy up to a permutation given that the exact concepts $\mathbf{c}$ are not identifiable from the label signal [35, 36]. More details are provided in Section H.5.

We find a clear contrast depending on the task constraint. Independence reduction is competitive on Visual Sudoku and CLEVR-Diff (first three columns in Figure 5) with all baselines achieving close to 99% concept accuracy, leaving little room for improvement. However, preserving dependencies improves learning on the arithmetic and CLEVR-CountDiff programs (last three columns). We hypothesize that this is due to *factorizations introducing* ***auxiliary symbolic variables***. For example,

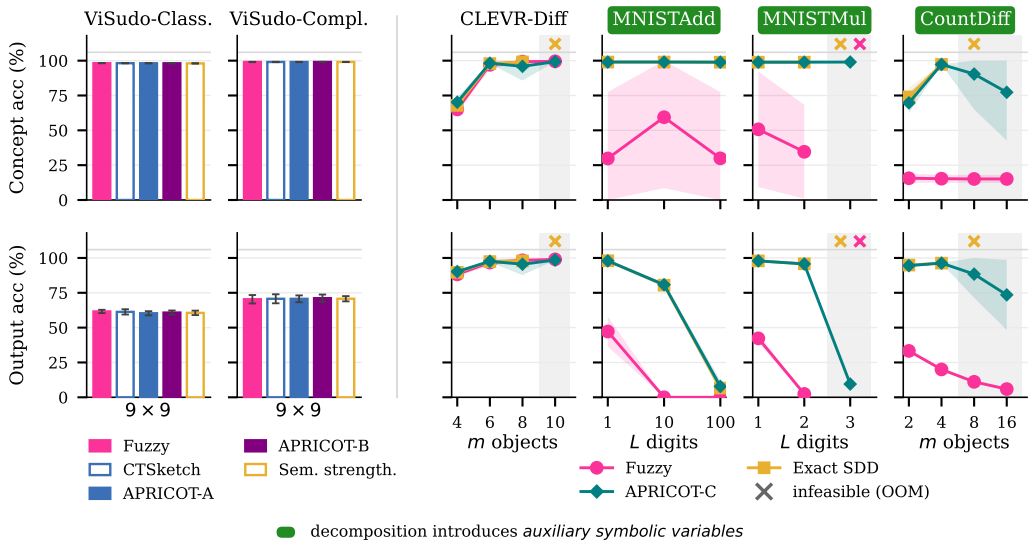


Figure 5: Concept and output accuracy across tasks and problem sizes. APRICOT matches exact compilation when the latter is feasible, and outperforms fuzzy in larger regimes.

in MNISTAdd, the constraint (detailed in Section H.3) has factors at each position k

$$A_k + B_k + C_{k-1} = Y_k + 10C_k$$

where the perception network predicts input digits A_k, B_k, derives an output Y_k, and introduces carries C_k. The carry C_k is an unweighted auxiliary symbol that denotes the part of the result at position k which exceeds 10 and rolls over position $k+1$. Its value must agree between the factors at these two positions, and jointly marginalizing them enforces this agreement. On the other hand, independently marginalizing the factors permits inconsistent carries which weakens the propagation of the supervised signal from Y_k to the perceived A_k, B_k. This matters for learning concepts and inferring outputs in more complicated settings (arithmetic as N grows, or CountDiff for any m).

In such settings, exact compilation and APRICOT C preserves agreements between factor computations and vastly outperforms the fuzzy baseline. Notice that this gap is not domain dependent as exemplified by the contrast between CLEVR-Diff and CLEVR-CountDiff, highlighting the importance of the constraint structure.

Larger error bars sometimes reflect optimization failures. For example, on CountDiff with $m = 8$ and $m = 16$, respectively one and three of ten APRICOT runs fail to converge, or on arithmetic nearly half of the fuzzy seeds. Task-specific tuning may reduce these failures, but the aggregate results support the trends discussed above. We also note that on MNISTMul, fuzzy uses a different factorization because the finer encoding used by SDD and APRICOT would give vanishing gradients (details in Section H.3.1). Its dense initial factors are thus larger and exceed GPU memory at $L = 3$.

6 Conclusions

We unify existing bottom-up compilation methods for WMC under a modular pipeline (Figure 1) and instantiate previously unexplored configurations via tensor-train decompositions (APRICOT). APRICOT combines TT factor representations with exact and recompressed multiplication, improving inference fidelity while extending the scale of feasible compilation. Our learning experiments show that independence reduction can be effective, but that preserving dependencies matters when local computations must relate shared auxiliary variables.

Limitations. Most TT decompositions and recompressions – particularly TCI and RSI– are not specifically tailored to sparse Boolean tensors. Future work should develop methods specific to this regime. Our results also motivate an extended study on how program factorization (particularly auxiliary-variable structure) impacts gradient-based learning beyond compiled NeSy methods.

Acknowledgements

We would like to thank Lorenzo Loconte, Christoph Staudt, Andreas Grivas and Joan Serrà for fruitful discussions during the writing of this paper. Emile van Krieken was funded by the NWO AiNed project "Human-Centric AI Agents with Common Sense" (NGF.1607.22.044). Luciano Serafini was funded by the NRRP project Future AI Research (FAIR - PE00000013) under the NRRP MUR program, funded by NextGenerationEU. Antonio Vergari was supported by the "UNREAL: Unified Reasoning Layer for Trustworthy ML" project (EP/Y023838/1) selected by the ERC and funded by UKRI EPSRC.

Author contributions. The core ideas were born when SB visited the University of Edinburgh under the supervision of AV. AV ideated approximating factors with circuits which SB implemented with tensor trains, further exploring product approximations. EvK and LS provided continuous and invaluable feedback throughout the project. SB conceived and implemented the experiments with guidance from AV and EvK. EvK also helped in running baseline experiments. All authors contributed to writing.

AI use statement. We used generative AI tools to assist in implementing the MNIST arithmetic and CLEVR experiments, for plotting, for designing the groundtruth dynamic programming solutions for the weighted inference experiments (as detailed in Section H.4), and for reviewing the manuscript before submission. The manuscript was written by the authors. APRICOT baselines were also manually implemented by the authors, only using AI for reviewing the code. We take responsibility for the final content of this work, including text and artifacts produced with the aid of generative AI.

References

[1] Kareem Ahmed, Stefano Teso, Kai-Wei Chang, Guy Van den Broeck, and Antonio Vergari. Semantic probabilistic layers for neuro-symbolic learning. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, volume 35, pages 29944–29959. Curran Associates, Inc., 2022.

[2] Kareem Ahmed, Kai-Wei Chang, and Guy Van den Broeck. Semantic strengthening of neuro-symbolic learning, 2023. URL https://arxiv.org/abs/2302.14207.

[3] Jonathan E. Atkins, Erik G. Boman, and Bruce Hendrickson. A spectral algorithm for seriation and the consecutive ones problem. *SIAM Journal on Computing*, 28(1):297–310, 2003. ISSN 0097-5397. doi: 10.1137/s0097539795285771. URL https://doi.org/10.1137/s0097539795285771.

[4] Eriq Augustine, Connor Pryor, Charles Dickens, Jay Pujara, William Yang Wang, and Lise Getoor. Visual sudoku puzzle classification: A suite of collective neuro-symbolic tasks. In *International Workshop on Neural-Symbolic Learning and Reasoning (NeSy)*, Windsor, United Kingdom, 2022.

[5] Samy Badreddine, Artur S. d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *CoRR*, abs/2012.13635, 2020. URL https://arxiv.org/abs/2012.13635.

[6] Oscar Chang, Lampros Flokas, Hod Lipson, and Michael Spranger. Assessing satnet's ability to solve the symbol grounding problem, 2023. URL https://arxiv.org/abs/2312.11522.

[7] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence.*, 172(6-7):772–799, 2008.

[8] Seewon Choi, Alaia Solko-Breslin, Rajeev Alur, and Eric Wong. CTSketch: Compositional Tensor Sketching for Scalable Neurosymbolic Learning, 2025. URL https://arxiv.org/abs/2503.24123. Version Number: 1.

[9] YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic modeling. Technical report, University of California, Los Angeles (UCLA), 2020.

[10] Hussam Al Daas, Grey Ballard, Paul Cazeaux, Eric Hallman, Agnieszka Miedlar, Mirjeta Pasha, Tim W. Reid, and Arvind K. Saibaba. Randomized algorithms for rounding in the tensor-train format, 2021. URL https://arxiv.org/abs/2110.04393.

[11] Alessandro Daniele, Tommaso Campari, Sagar Malhotra, and Luciano Serafini. Deep Symbolic Learning: Discovering Symbols and Rules from Perceptions. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 3597–3605, Macau, SAR China, August 2023. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-1-956792-03-4. doi: 10.24963/ijcai.2023/400. URL https://www.ijcai.org/proceedings/2023/400.

[12] Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM (JACM)*, 48: 608–647, 2001.

[13] Adnan Darwiche. A differential approach to inference in bayesian networks. *Journal of the ACM (JACM)*, 50:280–305, 2003.

[14] Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[15] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research (JAIR)*, 17:229–264, 2002.

[16] Lucile Dierckx, Alexandre Dubray, Siegfried Nijssen, and J Kwok. Learning from logical constraints with lower-and upper-bound arithmetic circuits. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pages 5012–5020. International Joint Conferences on Artificial Intelligence, 2025.

[17] Ivan Donadello, Luciano Serafini, and Artur d'Avila Garcez. Logic tensor networks for semantic image interpretation, 2017. URL https://arxiv.org/abs/1705.08968.

[18] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 2010. ISSN 0008-414X. doi: 10.4153/cjm-1965-045-4. URL https://doi.org/10.4153/cjm-1965-045-4.

[19] Bertram Felgenhauer and Frazer Jarvis. Mathematics of sudoku i. 2006. URL https://api.semanticscholar.org/CorpusID:122734246.

[20] S. A. Goreinov, I. V. Oseledets, D. V. Savostyanov, E. E. Tyrtyshnikov, and N. L. Zamarashkin. How to find a good submatrix. In Vadim Olshevsky and Eugene Tyrtyshnikov, editors, *Matrix Methods: Theory, Algorithms and Applications*, pages 247–256. World Scientific, 2010. doi: 10.1142/9789812836021_0015. URL https://www.worldscientific.com/doi/abs/10.1142/9789812836021_0015.

[21] Johnnie Gray and Garnet Kin-Lic Chan. Hyperoptimized approximate contraction of tensor networks with arbitrary geometry. *Physical Review X*, 14(1), 2024. ISSN 2160-3308. doi: 10.1103/physrevx.14.011009. URL https://doi.org/10.1103/physrevx.14.011009.

[22] Johnnie Gray and Stefanos Kourtis. Hyper-optimized tensor network contraction. *Quantum*, 5: 410, 2021. ISSN 2521-327X. doi: 10.22331/q-2021-03-15-410. URL https://doi.org/10.22331/q-2021-03-15-410.

[23] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, 2010. URL https://arxiv.org/abs/0909.4061.

[24] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, 2017.

[25] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. Algebraic model counting. *Journal of Applied Logic*, 22:46–62, 2017.

[26] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR)*, 2015.

[27] Doga Gizem Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *International Conference on Principles of Knowledge Representation and Reasoning*, 2014.

[28] Jean-Marie Lagniez and Pierre Marquis. An improved decision-dnnf compiler. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 667–673, 2017. doi: 10.24963/ijcai.2017/93. URL https://doi.org/10.24963/ijcai.2017/93.

[29] Lorenzo Loconte, M. Sladek Aleksanteri, Stefan Mengel, Martin Trapp, Arno Solin, Nicolas Gillis, and Antonio Vergari. Subtractive mixture models via squaring: Representation and learning. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[30] Lorenzo Loconte, Antonio Mari, Gennaro Gala, Robert Peharz, Cassio de Campos, Erik Quaeghebeur, Gennaro Vessio, and Antonio Vergari. What is the relationship between tensor factorizations and circuits (and how can we exploit it)? *arXiv preprint arXiv:2409.07953*, 2024.

[31] Jaron Maene and Luc De Raedt. Soft-Unification in Deep Probabilistic Logic. *Advances in Neural Information Processing Systems*, 36:60804–60820, December 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/bf215fa7fe70a38c5e967e59c44a99d0-Abstract-Conference.html.

[32] Jaron Maene, Vincent Derkinderen, and Luc De Raedt. On the hardness of probabilistic neurosymbolic learning. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 34203–34218. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/maene24a.html.

[33] Jaron Maene, Vincent Derkinderen, and Pedro Zuidberg Dos Martires. KLay: Accelerating arithmetic circuits for neurosymbolic AI. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Zes7Wyif8G.

[34] Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *arXiv preprint arXiv:1805.10872*, 2018.

[35] Emanuele Marconato, Stefano Teso, and Andrea Passerini. Neuro-symbolic reasoning shortcuts: Mitigation strategies and their limitations, 2023. URL https://arxiv.org/abs/2303.12578.

[36] Emanuele Marconato, Samuele Bortolotti, Emile van Krieken, Paolo Morettin, Elena Umili, Antonio Vergari, Efthymia Tsamoura, Andrea Passerini, and Stefano Teso. Symbol grounding in neuro-symbolic ai: A gentle introduction to reasoning shortcuts, 2026. URL https://arxiv.org/abs/2510.14538.

[37] Wannes Meert and Arthur Choi. Pysdd. In *Recent Trends in Knowledge Compilation*, number 17381 in Dagstuhl Reports, 2017.

[38] Zhaonan Meng, Yuehaw Khoo, Jiajia Li, and E. Miles Stoudenmire. Recursive sketched interpolation: Efficient hadamard products of tensor trains, 2026. URL https://arxiv.org/abs/2602.17974.

[39] Yuriel Núñez Fernández, Marc K Ritter, Matthieu Jeannin, Jheng-Wei Li, Thomas Kloss, Thibaud Louvet, Satoshi Terasaki, Olivier Parcollet, Jan von Delft, Hiroshi Shinaoka, and Xavier Waintal. Learning tensor networks with tensor cross interpolation: New algorithms and libraries. *SciPost Physics*, 18(3), March 2025. ISSN 2542-4653. doi: 10.21468/scipostphys.18.3.104. URL http://dx.doi.org/10.21468/SciPostPhys.18.3.104.

[40] Ryoma Onaka, Kengo Nakamura, Masaaki Nishino, and Norihito Yasuda. Tensor decomposition meets knowledge compilation: A study comparing tensor trains with obdds, 2025. URL https://arxiv.org/abs/2502.03702.

[41] Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2013.

[42] Ivan Oseledets and Eugene Tyrtyshnikov. TT-cross approximation for multidimensional arrays. *Linear Algebra and its Applications*, 432(1):70–88, January 2010. ISSN 00243795. doi: 10.1016/j.laa.2009.07.024. URL https://linkinghub.elsevier.com/retrieve/pii/S0024379509003747.

[43] Ivan. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33: 2295–2317, 2011.

[44] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019. URL http://arxiv.org/abs/1912.01703.

[45] David Pérez-García, F. Verstraete, Michael M. Wolf, and Juan Ignacio Cirac. Matrix product state representations. *Quantum Information and Computing*, 7(5):401–430, 2007. ISSN 1533-7146.

[46] Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *23rd Conference on Artificial Intelligence (AAAI)*, volume 8, pages 517–522, 2008.

[47] Alex Rogozhnikov. Einops: Clear and reliable tensor manipulations with einstein-like notation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=oapKSVM2bcj.

[48] Dmitry V. Savostyanov. Quasioptimality of maximum-volume cross interpolation of tensors. *Linear Algebra and its Applications*, 458:217–244, October 2014. ISSN 0024-3795. doi: 10.1016/j.laa.2014.06.006. URL http://dx.doi.org/10.1016/j.laa.2014.06.006.

[49] Sebastian Schlag, Tobias Heuer, Lars Gottesbüren, Yaroslav Akhremtsev, Christian Schulz, and Peter Sanders. High-quality hypergraph partitioning, 2021. URL https://arxiv.org/abs/2106.08696.

[50] Ulrich Schollwoeck. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326:96–192, 2010.

[51] Yujia Shen, Arthur Choi, and Adnan Darwiche. Tractable operations for arithmetic circuits of probabilistic models. In *Advances in Neural Information Processing Systems 29 (NeurIPS)*. Curran Associates, Inc., 2016.

[52] Lennert De Smet, Pedro Zuidberg Dos Martires, Robin Manhaeve, Giuseppe Marra, Angelika Kimmig, and Luc De Raedt. Neural probabilistic logic programming in discrete-continuous domains. *ArXiv*, abs/2303.04660, 2023.

[53] Lennert De Smet, Emanuele Sansone, and Pedro Zuidberg Dos Martires. Differentiable sampling of categorical distributions using the catlog-derivative trick. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=AQyqxXctsN.

[54] Wolfgang Stammer, Patrick Schramowski, and Kristian Kersting. Right for the Right Concept: Revising Neuro-Symbolic Concepts by Interacting with their Explanations. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3618–3628, June 2021. doi: 10.1109/CVPR46437.2021.00362. URL https://ieeexplore.ieee.org/document/9578154. ISSN: 2575-7075.

[55] Zhonghao Sun, Jizu Huang, Chuanfu Xiao, and Chao Yang. Hatt: Hadamard avoiding tt recompression, 2025. URL https://arxiv.org/abs/2410.04385.

[56] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[57] Emile van Krieken, Erman Acar, and Frank van Harmelen. Semi-supervised learning using differentiable reasoning. *CoRR*, abs/1908.04700, 2019. URL http://arxiv.org/abs/1908.04700.

[58] Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy logic operators. *Artif. Intell.*, 302:103602, 2022.

[59] Emile van Krieken, Thiviyan Thanapalasingam, Jakub M. Tomczak, F. V. Harmelen, and Annette ten Teije. A-nesi: A scalable approximate method for probabilistic neurosymbolic inference. *ArXiv*, abs/2212.12393, 2022.

[60] Emile van Krieken, Pasquale Minervini, Edoardo M. Ponti, and Antonio Vergari. On the independence assumption in neurosymbolic learning, 2024. URL https://arxiv.org/abs/2404.08458.

[61] Emile van Krieken, Pasquale Minervini, Edoardo Ponti, and Antonio Vergari. Neurosymbolic reasoning shortcuts under the independence assumption, 2025. URL https://arxiv.org/abs/2507.11357.

[62] Emile van Krieken, Pasquale Minervini, Edoardo Ponti, and Antonio Vergari. Neurosymbolic diffusion models, 2025. URL https://arxiv.org/abs/2505.13138.

[63] Antonio Vergari, Nicola Di Mauro, and Guy Van den Broeck. Tractable probabilistic models: Representations, algorithms, learning, and applications. *Tutorial at the 35th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2019.

[64] Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A compositional atlas of tractable circuit operations for probabilistic inference. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, pages 13189–13201. Curran Associates, Inc., 2021.

[65] Victor Verreet, Lennert De Smet, Luc De Raedt, and Emanuele Sansone. Explain, agree, learn: scaling learning for neural probabilistic logic. In *ECAI 2024: 27th European Conference on Artificial Intelligence, 19–24 October 2024, Santiago de Compostela, Spain–Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024)*, pages 1349–1356. SAGE Publications Pvt. Ltd 1 Oliver's Yard, 55 City Road, London, EC1Y 1SP, 2024.

[66] Po-Wei Wang, Priya L. Donti, Bryan Wilder, and J. Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. *CoRR*, abs/1905.12149, 2019. URL http://arxiv.org/abs/1905.12149.

[67] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Broeck. A semantic loss function for deep learning with symbolic knowledge. In *37th International conference on machine learning (ICML)*, pages 5502–5511. PMLR, 2018.

A Circuits

A.1 Background

We first define circuits abstractly over a semiring, which allows us to distinguish the logical and probabilistic representations used in NeSy AI.

Definition A.1 (Circuit [9, 64, 25])**.** Let $(\mathcal{S}, \oplus, \otimes, 0, 1)$ be a commutative semiring. A *circuit* $f(\mathbf{z})$ is a directed acyclic computational graph whose intermediate units compute $\oplus$ and $\otimes$ operations over tractable input functions of variables $\mathbf{z}$.

Different choices of semiring recover the circuit representations commonly used in probabilistic inference and knowledge compilation. *Probabilistic circuits* (PCs) [13, 63, 9] operate over the non-negative real semiring $(\mathbb{R}_{\geq 0}, +, \times, 0, 1)$ and encode non-negative functions; their sum-unit weights are typically constrained to be non-negative [29, 30]. In contrast, *logical circuits* are conventionally defined over the Boolean semiring $(\{0, 1\}, \vee, \wedge, 0, 1)$ as DAGs whose internal units are logical OR/AND operations and whose inputs represent logical literals [15].

Tractable computation of multiplication and marginalization over arithmetic circuits requires certain structural properties, which we briefly discuss next. We refer the reader to Choi et al. [9] for details.

We focus on circuits that are *smooth*, i.e., all inputs of sum units are defined over the same variables, and *decomposable*, i.e., inputs of product units are defined over disjoint sets of variables. These properties enable tractable marginalization in time linear in the circuit size [9].

For tractable circuit multiplication, stronger structural constraints are required. A decomposable circuit is *structured-decomposable* if its product units respect a common hierarchical partition of the variables, typically represented by a vtree. Two structured-decomposable circuits are *compatible* [64] when their decompositions agree on their shared variables. Compatible circuits can be multiplied in time polynomial in their sizes while preserving smoothness and decomposability.

A.2 Probabilistic NeSy reasoning

To use a logical circuit together with a PC, we first cast the logical circuit into the real sum-product semiring. This conversion is sound for *deterministic* logical circuits, whose OR units have children with disjoint supports. For such circuits, at most one child of each OR unit evaluates to 1 for any assignment, and hence Boolean OR can be replaced by real addition and AND by multiplication without changing the encoded Boolean function. The resulting arithmetic circuit has unit weights equal to 1 at the former OR units.

Consequently, if ϕ is represented by a smooth, decomposable, deterministic logical circuit, its arithmetic representation is compatible with the PC $p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})$, and the two can be multiplied tractably, then Equation (15) can be computed exactly and efficiently. A first requirement is therefore constructing from ϕ a logical circuit with the required structural properties, a task known as **knowledge compilation** (KC) and which is at least NP-hard [15, 7].

Most widely used KC software operates on propositional formulas and outputs circuits over Boolean variables.[4] Thus, the propositionalization overhead reported here reflects our use of available Boolean KC implementations, rather than a fundamental requirement of circuit representations.

Sentential decision diagrams (SDDs) [14] are a popular class of deterministic and structured-decomposable logical circuits for KC. After interpreting their deterministic OR/AND units in the real sum-product semiring, they can be combined with compatible PCs $p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})$. They can also be constructed bottom-up by **multiplying** circuit representations of factors ϕ_j.

Let $\mathrm{KC}(\phi_j)$ denote the arithmetic circuit obtained by compiling factor ϕ_j into a deterministic logical circuit and interpreting it in the sum-product semiring. We rewrite the WMC using the symbolic

[4] This Boolean encoding is a property of the knowledge-compilation tools considered here, rather than an intrinsic limitation of circuits. Circuits can in principle be defined directly over finite-domain categorical variables, for example through multi-valued decision diagrams or suitable categorical input functions. When using propositional KC software, however, a categorical variable c_i can be encoded using Boolean indicators $b_{i,j} = \mathbb{1}[c_i = j]$ together with constraints enforcing that exactly one indicator is true. The size increase depends strongly on the chosen encoding. With the direct one-hot encoding used in our experiments, a 9×9 Sudoku changes from $d = 81$ categorical variables and $N = 810$ factors to $d = 729$ Boolean variables and $N = 10287$ factors after propositionalization and addition of mutual-exclusivity constraints.

operator MUL, which takes a set of compatible circuits as input and returns a circuit representing their product:

$$\sum_{\mathbf{c}\in\mathcal{C}} p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})\mathrm{MUL}\big(\mathrm{KC}(\phi_1),\ldots,\mathrm{KC}(\phi_N)\big)(\mathbf{c}). \tag{8}$$

This is the classical recipe underlying many NeSy approaches, including semantic loss [67] and DeepProbLog [34].

B Related Work

Knowledge compilation. Compilers more permissive than SDDs handle larger formulas at the cost of compositionality with other circuits. d4 [28] is an exact compiler for d-DNNF, a broader class than SDDs that drops structured-decomposability. It scales to formulas no SDD compiler can handle (e.g., 6x6 Sudoku, though still not 9x9), but the resulting circuits can only be weighted by fully-factorized probabilities, an independence assumption known to lack expressivity in practice [1, 60, 61]. Dierckx et al. [16] propose an approximate variant of d-DNNF compilation, which enumerates a budget-limited set of models into a DNF to derive lower- and upper-bound circuits. While these provide provable intervals, the resulting circuits are again not structured-decomposable, making them orthogonal to our target. Onaka et al. [40] treat TTs themselves as a target for **exact factor compilation**, using them in the knowledge-compilation map alongside NNF subsets and showing they are strictly more succinct than OBDDs while supporting the same polytime operations. This corresponds to combining TT factors via exact multiplications without **TT-SVD approximation** or **product recompression**, and inherits the same exponential blow-up under multiplication that motivates the approximate strategies we study.

Scalable inference in NeSy. Other approaches avoid circuit compilation entirely. A-NeSI [59] trains a *neural network* to amortize WMC inference, which are more expressive than circuits. Like APRICOT (but unlike SDD pipelines), A-NeSI can also learn the logic of black-box programs using input-output examples. However, the neural approach offers no formal error guarantees and gives a representation that does not support downstream tractable operations such as multiplication with other circuits or conditioning on evidence. *Monte Carlo methods* [32, 52, 53, 65, 62] sample concept assignments to estimate gradients, which also extends to continuous settings, but Maene et al. [32] show that the underlying gradient estimation from samples is itself #P-hard and can require an exponential number of samples to converge. Naturally, they also do not produce circuits that can be reused for downstream operations, targeting a different use case than ours.

C Error bounds

We investigate how the different approximations impact the final WMC error. The **independence** approximation is crude, with an unbounded error. This approximation depends not only on the program semantics but also on how it is represented, which can lead to unpredictable behavior. As we will see in our results and as we derive analytically in Section E, the independence approximation for Sudoku derives estimates that vary by many orders of magnitude depending on the encoding used (categorical or binary).

The other approximations have more controlled error propagation. First, the ϵ error bound on the **factor** approximation introduced by TT-SVD in APRICOT pipelines propagates linearly with the number of factors when **multiplying them exactly**.

Theorem C.1. *Let $\boldsymbol{\mathcal{X}}^* = \boldsymbol{\mathcal{X}}_j * \boldsymbol{\mathcal{X}}_l$ the element-wise product of two tensors. Let $\tilde{\boldsymbol{\mathcal{X}}}^* = \tilde{\boldsymbol{\mathcal{X}}}_j * \tilde{\boldsymbol{\mathcal{X}}}_l$, where $\|\boldsymbol{\mathcal{X}}_j - \tilde{\boldsymbol{\mathcal{X}}}_j\|_F \leq \epsilon_j$ and $\|\boldsymbol{\mathcal{X}}_l - \tilde{\boldsymbol{\mathcal{X}}}_l\|_F \leq \epsilon_l$. Then, $\|\boldsymbol{\mathcal{X}}^* - \tilde{\boldsymbol{\mathcal{X}}}^*\|_F \leq \|\boldsymbol{\mathcal{X}}_j\|_\infty \epsilon_l + \|\boldsymbol{\mathcal{X}}_l\|_\infty \epsilon_j + \epsilon_j\epsilon_l$ where $\|\boldsymbol{\mathcal{X}}_j\|_\infty = \max_{c_1,\ldots,c_d} |\boldsymbol{\mathcal{X}}_j(c_1,\ldots,c_d)|$ is the max norm.*

Proof. Let $\Delta_j = \tilde{\boldsymbol{\mathcal{X}}}_j - \boldsymbol{\mathcal{X}}_j$ and $\Delta_l = \tilde{\boldsymbol{\mathcal{X}}}_l - \boldsymbol{\mathcal{X}}_l$, so that $\|\Delta_j\|_F \leq \epsilon_j$ and $\|\Delta_l\|_F \leq \epsilon_l$. Expanding the product,

$$\begin{aligned}\boldsymbol{\mathcal{X}}^* - \tilde{\boldsymbol{\mathcal{X}}}^* &= \boldsymbol{\mathcal{X}}_j * \boldsymbol{\mathcal{X}}_l - (\boldsymbol{\mathcal{X}}_j + \Delta_j) * (\boldsymbol{\mathcal{X}}_l + \Delta_l) \\ &= -\boldsymbol{\mathcal{X}}_j * \Delta_l - \Delta_j * \boldsymbol{\mathcal{X}}_l - \Delta_j * \Delta_l.\end{aligned}$$

By the triangle inequality,

$$\|\boldsymbol{\mathcal{X}}^* - \tilde{\boldsymbol{\mathcal{X}}}^*\|_F \leq \|\boldsymbol{\mathcal{X}}_j * \Delta_l\|_F + \|\Delta_j * \boldsymbol{\mathcal{X}}_l\|_F + \|\Delta_j * \Delta_l\|_F.$$

For any two tensors $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}$ of matching shape, the Hadamard product satisfies $\|\boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}\|_F \leq \|\boldsymbol{\mathcal{A}}\|_\infty \|\boldsymbol{\mathcal{B}}\|_F$ – follows from $\sum_{\mathbf{c}} (\boldsymbol{\mathcal{A}}(\mathbf{c})\boldsymbol{\mathcal{B}}(\mathbf{c}))^2 \leq \max_{\mathbf{c}} \boldsymbol{\mathcal{A}}(\mathbf{c})^2 \sum_{\mathbf{c}} \boldsymbol{\mathcal{B}}(\mathbf{c})^2 = \|\boldsymbol{\mathcal{A}}\|_\infty^2 \|\boldsymbol{\mathcal{B}}\|_F^2$. Applied to the first two terms,

$$\|\boldsymbol{\mathcal{X}}_j * \Delta_l\|_F \leq \|\boldsymbol{\mathcal{X}}_j\|_\infty \|\Delta_l\|_F \leq \|\boldsymbol{\mathcal{X}}_j\|_\infty \, \epsilon_l,$$
$$\|\Delta_j * \boldsymbol{\mathcal{X}}_l\|_F \leq \|\boldsymbol{\mathcal{X}}_l\|_\infty \|\Delta_j\|_F \leq \|\boldsymbol{\mathcal{X}}_l\|_\infty \, \epsilon_j.$$

For the third term, the L^p-norm inequality $\|\Delta_l\|_\infty \leq \|\Delta_l\|_F$ (treating the tensor as a vector) gives

$$\|\Delta_j * \Delta_l\|_F \leq \|\Delta_l\|_\infty \|\Delta_j\|_F \leq \epsilon_l \, \epsilon_j.$$

Summing the three bounds yields the claim. □

As we study boolean tensors, small ϵ_j implies $\|\boldsymbol{\mathcal{X}}_j\|_\infty \approx 1$ and thus the error on $\boldsymbol{\mathcal{X}}^*$ is $\mathcal{O}(\epsilon_i + \epsilon_j)$. By extension, for N factors, the error on $\boldsymbol{\mathcal{X}}^*$ is $\mathcal{O}(N\epsilon)$ where $\epsilon = \max_j \epsilon_j$.

Recompression after multiplication (Section 3.3) introduces additional errors specific to each strategy. Deterministic TT-ROUND, in particular, can maintain guarantees on the Frobenius error of the recompressed product, which adds to the error of the exact result [43].

Reusing Theorem C.1 and considering the weight tensor $\boldsymbol{\mathcal{P}}$ such that $\boldsymbol{\mathcal{P}}(\mathbf{c}) = p_{\boldsymbol{\theta}}(\mathbf{c} \mid \mathbf{x})$, we first bound the error before the final contraction.

Corollary C.2 (Weighted-tensor error)**.** *Let $\tilde{\boldsymbol{\mathcal{X}}}$ be an ϵ-approximation of $\boldsymbol{\mathcal{X}}$, and let $\boldsymbol{\mathcal{P}}$ be a tensor of weights. Then*

$$\|\boldsymbol{\mathcal{P}} * \tilde{\boldsymbol{\mathcal{X}}} - \boldsymbol{\mathcal{P}} * \boldsymbol{\mathcal{X}}\|_F \leq \|\boldsymbol{\mathcal{P}}\|_\infty \epsilon. \tag{9}$$

This controls the full weighted tensor and therefore applies before any marginalization or contraction. Since the WMC is the contraction of this tensor to a scalar, we can derive a separate bound directly. For tensors of matching shape, let us denote $\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \rangle_F = \sum_{\mathbf{c}} \boldsymbol{\mathcal{A}}(\mathbf{c})\boldsymbol{\mathcal{B}}(\mathbf{c})$ for their Frobenius inner product.

Corollary C.3 (WMC error)**.** *Let $\tilde{\boldsymbol{\mathcal{X}}}$ be an approximation of the constraint tensor $\boldsymbol{\mathcal{X}}$ such that $\|\tilde{\boldsymbol{\mathcal{X}}} - \boldsymbol{\mathcal{X}}\|_F \leq \epsilon$, and let $\boldsymbol{\mathcal{P}}$ be any tensor of nonnegative weights. Then*

$$\left| \mathrm{W}\widehat{\mathrm{M}}\mathrm{C} - \mathrm{WMC} \right| \leq \epsilon \, \|\boldsymbol{\mathcal{P}}\|_F. \tag{10}$$

If $\boldsymbol{\mathcal{P}}$ is a normalized probability tensor, then $\|\boldsymbol{\mathcal{P}}\|_F \leq 1$ and consequently $|\, \mathrm{W}\widehat{\mathrm{M}}\mathrm{C} - \mathrm{WMC} \,| \leq \epsilon$.

Proof. Let $\Delta = \tilde{\boldsymbol{\mathcal{X}}} - \boldsymbol{\mathcal{X}}$. Since the exact and approximate weighted model counts are $\langle \boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{X}} \rangle_F$ and $\langle \boldsymbol{\mathcal{P}}, \tilde{\boldsymbol{\mathcal{X}}} \rangle_F$, respectively,

$$\begin{aligned} \left| \mathrm{W}\widehat{\mathrm{M}}\mathrm{C} - \mathrm{WMC} \right| &= |\langle \boldsymbol{\mathcal{P}}, \Delta \rangle_F| \\ &\leq \|\boldsymbol{\mathcal{P}}\|_F \, \|\Delta\|_F \leq \epsilon \, \|\boldsymbol{\mathcal{P}}\|_F, \end{aligned}$$

where the first inequality is Cauchy-Schwarz. And, if $\boldsymbol{\mathcal{P}}$ is normalized, we have $\|\boldsymbol{\mathcal{P}}\|_F \leq \|\boldsymbol{\mathcal{P}}\|_1 = 1$. □

D CTSKETCH and Program Dependencies

CTSKETCH [8] approximates each factor ϕ_j with a tensor train (Section 3.1), computes the factor WMC independently, and multiplies them progressively following a "tree-like" structure. We formalize this using factor graphs and show that the tree-like assumption is violated by many NeSy programs, including the Sudoku tasks explored by Choi et al. [8], where we find that CTSKETCH introduces independence approximations (Equation (3)).

D.1 Tree-like factorizations

Example D.1 (sum_4 [8])**.** Consider the program sum_4 which takes as input four digits $c_1, c_2, c_3, c_4 \in \{0, \dots, 9\}$ and outputs their sum $y = c_1 + c_2 + c_3 + c_4$. This program can be decomposed using two intermediate sums $y_1 = c_1 + c_2$ and $y_2 = c_3 + c_4$, followed by a final addition $y = y_1 + y_2$. The constraint factorizes as

$$\phi(\mathbf{c}, y) = \sum_{y_1, y_2} \underbrace{\mathbb{1}[c_1 + c_2 = y_1]}_{\phi_1(c_1, c_2, y_1)} \underbrace{\mathbb{1}[c_3 + c_4 = y_2]}_{\phi_2(c_3, c_4, y_2)} \underbrace{\mathbb{1}[y_1 + y_2 = y]}_{\phi_3(y_1, y_2, y)},$$

with the corresponding factor graph shown in Figure 6a.

Let $\mathbf{c}_1 = \{c_1, c_2\}$ and $\mathbf{c}_2 = \{c_3, c_4\}$ denote the two disjoint subsets of $\mathbf{c}$ associated with ϕ_1 and ϕ_2 respectively. The WMC rewrites as:

$$p_{\boldsymbol{\theta}}(y|\mathbf{x}) = \sum_{\mathbf{c}} p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x}) \left(\sum_{y_1, y_2} \phi_1(\mathbf{c}_1, y_1)\, \phi_2(\mathbf{c}_2, y_2)\, \phi_3(y_1, y_2, y) \right).$$

Assuming $p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x}) = \prod_i p_{\boldsymbol{\theta}}(c_i|\mathbf{x})$, CTSKETCH simplifies this marginalization in two steps. **First**, since ϕ_3 does not depend on $\mathbf{c}$ and $p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})$ does not depend on y_1, y_2, the two sums commute and ϕ_3 can be moved outside:

$$\begin{aligned} p_{\boldsymbol{\theta}}(y|\mathbf{x}) &= \sum_{y_1, y_2} \phi_3(y_1, y_2, y) \left(\sum_{\mathbf{c}} p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x}) \phi_1(\mathbf{c}_1, y_1)\, \phi_2(\mathbf{c}_2, y_2) \right) \\ &= \sum_{y_1, y_2} \phi_3(y_1, y_2, y) \left(\sum_{\mathbf{c}} p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x}) \prod_{j=1}^{2} \phi_j(\mathbf{c}_j, y_j) \right) \end{aligned}$$

Second, the tree structure of the factor graph and disjoint scopes of ϕ_1, ϕ_2 let us push inner marginalizations down to the leaves.

$$p_{\boldsymbol{\theta}}(y|\mathbf{x}) = \sum_{y_1, y_2} \phi_3(y_1, y_2, y) \prod_{j=1}^{2} \Big(\sum_{\mathbf{c}_j} p_{\boldsymbol{\theta}}(\mathbf{c}_j|\mathbf{x})\, \phi_j(\mathbf{c}_j, y_j) \Big)$$

Read in isolation, this second step has exactly the form of the independence approximation of Equation (3): it computes a per-factor WMC and multiplies the results. The crucial difference is that here the equality is *exact* because the leaf $\mathbf{c}$-scopes are disjoint and the marginals are genuinely independent.

CTSKETCH additionally approximates each ϕ_j as a tensor train, but the same factorization trick applies directly to the dense factor tensors $\boldsymbol{\mathcal{X}}_j$. MNISTADD [34] satisfies this tree-like assumption, as do other programs that decompose into trees of intermediate variables.

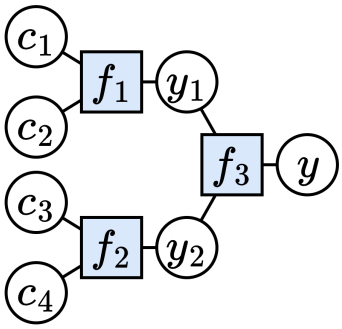
D.2 Overlapping factors

In most NeSy programs the factor graph is not a tree: variables appear in multiple factors and the corresponding scopes overlap substantially.

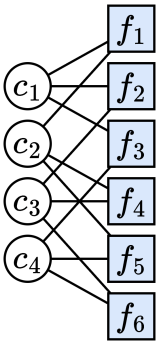
Example D.2 (Sudoku 4x4)**.** Consider the 4×4 Sudoku classification task, where the program takes as input 16 concepts $c_i \in \{1, 2, 3, 4\}$ representing the digits in each cell and outputs whether the grid is valid. The program decomposes into 56 pairwise inequality constraints across rows, columns and blocks. For example, the first row alone factorizes as

$$\begin{aligned} \phi_{\text{row1}}(c_1, c_2, c_3, c_4) = &\underbrace{\mathbb{1}[c_1 \neq c_2]}_{\phi_1} \underbrace{\mathbb{1}[c_1 \neq c_3]}_{\phi_2} \underbrace{\mathbb{1}[c_1 \neq c_4]}_{\phi_3} \\ &\underbrace{\mathbb{1}[c_2 \neq c_3]}_{\phi_4} \underbrace{\mathbb{1}[c_2 \neq c_4]}_{\phi_5} \underbrace{\mathbb{1}[c_3 \neq c_4]}_{\phi_6}. \end{aligned}$$

The corresponding factor graph is shown in Figure 6b. Each variable already participates in three factors within a single row, and in even more once columns and blocks are considered. These overlapping scopes and cycles in the factor graph prevent us from pushing marginalization sums inside the product, as we did in Example D.1.



(a) sum_4 program.



(b) First-row constraint of a 4×4 Sudoku program.

Figure 6: Factor graphs for (a) a tree-like program where marginalization sums can be pushed inside individual factors, and (b) a program with overlapping scopes and cycles where this simplification does not apply.

Despite the overlap, CTSKETCH applies the same per-factor simplification to Sudoku, pushing each marginalization sum to its associated factor as if the leaves had disjoint $\mathbf{c}$-scopes:

$$\sum_{\mathbf{c}} p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x}) \prod_j \phi_j(\mathbf{c}_j) \overset{\text{I,F}}{\approx} \prod_j \Big(\sum_{\mathbf{c}_j} p_{\boldsymbol{\theta}}(\mathbf{c}_j|\mathbf{x})\, \text{AC}(\phi_j)(\mathbf{c}_j) \Big).$$

The right-hand side has exactly the form of the (exact) factorization in Example D.1, but here the overlapping $\mathbf{c}_j$ turn the equality into an approximation: it is the composition of a **factor approximation** – each ϕ_j replaced by its TT $\text{AC}(\phi_j)$ – with an **independence approximation**. The independence step is the source of the unbounded error analyzed in Section C and empirically illustrated in Section E.

E Independence Approximation and Categorical vs. Binary Encoding

We illustrate how the **independence approximation** can yield drastically different model count estimates depending on whether ϕ is encoded over categorical or binary variables, even though the two encodings describe the same set of valid worlds. We use Sudoku as a concrete example, with the same constraint expressed once over categorical cells and once after the propositionalization required by SDD compilation (Section 2.1).

E.1 Independence Approximation under a Uniform Prior

Specializing the independence approximation to a uniform prior $p_{\boldsymbol{\theta}}(\mathbf{c}) = 1/|\mathcal{C}|$, the WMC reduces to the model count $\text{MC} = \sum_{\mathbf{c}} \phi(\mathbf{c})$, and the approximation becomes

$$\widehat{\text{MC}}^{\text{I}} \;=\; |\mathcal{C}| \cdot \prod_{j=1}^{N} p_j, \qquad p_j \;:=\; \frac{1}{|\mathcal{C}_j|} \sum_{\mathbf{c}_j \in \mathcal{C}_j} \phi_j(\mathbf{c}_j), \tag{11}$$

where p_j is the fraction of assignments over the scope $\mathbf{c}_j$ that satisfy ϕ_j. Both factors of Equation (11) depend on the encoding only through trivial counting, but their product is dominated by very different magnitudes in the two encodings.

E.2 Sudoku 4x4: Categorical Encoding

The categorical formulation has variables $c_i \in \{1, \ldots, 4\}$ for $i = 1, \ldots, 16$, giving $|\mathcal{C}| = 4^{16}$. The constraint ϕ decomposes into $N = 56$ pairwise inequalities $\phi_j(c_i, c_l) = \mathbb{1}[c_i \neq c_l]$ over cell pairs (i, l) sharing a row, column, or box. Each factor has $|\mathcal{C}_j| = 16$ joint assignments of which 12 are satisfying, so $p_j = 3/4$. Plugging into Equation (11),

$$\widehat{\text{MC}}^{\text{I}}_{\text{cat}} \;=\; 4^{16} \cdot \left(\tfrac{3}{4}\right)^{56} \;\approx\; 432.9,$$

which overestimates the true MC of 288 by a factor of $\approx 1.5\times$.

E.3 Sudoku 4x4: Binary Encoding

SDD compilation requires propositionalizing ϕ via indicators $b_{i,v} = \mathbb{1}[c_i = v]$ for $i = 1, \ldots, 16$ and $v \in \{1, \ldots, 4\}$, resulting in 64 Boolean variables and $|\mathcal{C}| = 2^{64}$. The constraint becomes a conjunction of $N = 336$ CNF clauses of four types:

- *At-least-one* (16 clauses): $b_{i,1} \vee b_{i,2} \vee b_{i,3} \vee b_{i,4}$ per cell, with $p_j = 1 - (1/2)^4 = 15/16$.
- *At-most-one* (96 clauses): $\neg b_{i,v} \vee \neg b_{i,v'}$ for each cell i and value pair $v < v'$, with $p_j = 3/4$.
- *Distinct values across rows and columns* ($96 + 96$ clauses): $\neg b_{i,v} \vee \neg b_{l,v}$ for each row- or column-pair (i, l) and each value v, with $p_j = 3/4$.
- *Distinct values across boxes* (32 clauses): same form, restricted to box-pairs not already covered by rows or columns, with $p_j = 3/4$.

Plugging into Equation (11),

$$\widehat{\mathrm{MC}}^{\mathrm{I}}_{\mathrm{bin}} \;=\; 2^{64} \cdot \left(\tfrac{15}{16}\right)^{16} \cdot \left(\tfrac{3}{4}\right)^{96+96+96+32} \;\approx\; 6.87 \times 10^{-22},$$

underestimating the true MC of 288 by roughly 24 orders of magnitude.

E.4 Discussion

The two estimates differ by approximately 24 orders of magnitude despite describing the same 288 valid boards. It is tempting to attribute this entirely to the N scaling of Equation (11), as propositionalization inflates the factor count from 56 to 336, and each additional sub-unit p_j pushes the product toward zero. Our inference experiments in Section 5 show that this picture is misleading: even when the SDD pipeline reduces the propositionalized factors back to a count K comparable to the categorical N via exact multiplications, the binary estimate continues to underestimate the true MC while the categorical one continues to overestimate it, and the two remain orders of magnitude apart.

The deeper issue is that the independence approximation, viewed as a fuzzy semantics with the product t-norm, is unbounded and *syntax-dependent*: its value depends not only on the set of models of ϕ but on how ϕ is written. As a minimal illustration, consider a single Boolean $a \in \{0, 1\}$. The formulas $\phi_1 := a \wedge a$ and $\phi_2 := a$ have the same single satisfying assignment (MC $= 1$), yet $\widehat{\mathrm{MC}}^{\mathrm{I}}(\phi_2) = 2 \cdot \frac{1}{2} = 1$ and $\widehat{\mathrm{MC}}^{\mathrm{I}}(\phi_1) = 2 \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$.

Categorical and binary encodings of Sudoku express the same underlying constraint with very different syntactic profiles, and the product t-norm amplifies these differences in opposite directions: even after multiplying propositionalized factors back together, the residual factor structure is not the same as the categorical one and produces a systematically smaller $\prod_j p_j$. This contrasts with fuzzy semantics built on other t-norms (Gödel, Łukasiewicz), which have more established bounds and structural guarantees. [58] Neither, however, corresponds to a probabilistic relaxation of Equation (1), which is what motivates the use of the product t-norm as the **independence approximation** in the first place.

F Tensor Train Algebra

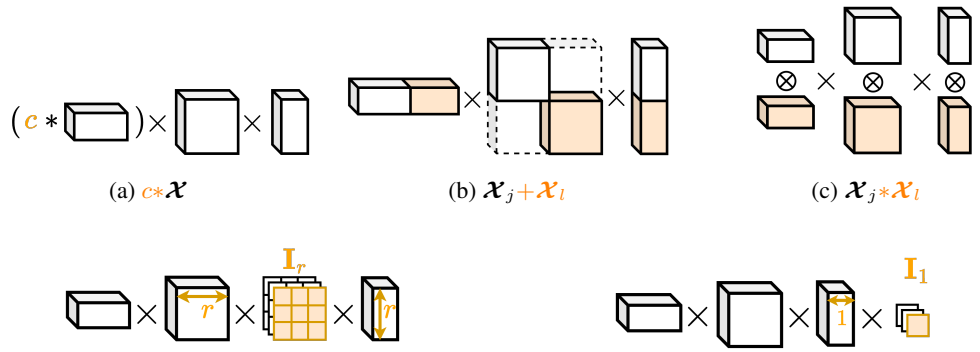
F.1 Addition, Scalar Multiplication and Hadamard Product

We briefly introduce the closed-form algebra of TTs [43]. Figure 7 provides a graphical illustration. Let $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ denote two TTs over the same d variables in the same core-to-variable ordering, with cores $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}},k}$ and $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{B}},k}$ of TT ranks r^A_k and r^B_k respectively. We write $n = \max_k n_k$ for the maximal mode size and assume for simplicity that input TTs share a common rank r.

Scalar multiplication (Figure 7a). For $c \in \mathbb{R}$, the TT $c \cdot \boldsymbol{\mathcal{A}}$ is obtained by absorbing c into any single core, e.g. $\boldsymbol{\mathcal{T}}_{c\boldsymbol{\mathcal{A}},1}(c_1) = c\,\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}},1}(c_1)$. The ranks are unchanged and the resulting TT has the same $\mathcal{O}(dnr^2)$ storage as $\boldsymbol{\mathcal{A}}$.

Addition (Figure 7b). The sum $\boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{B}}$ is a TT whose cores stack the input cores block-diagonally,

$$\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}}+\boldsymbol{\mathcal{B}},k}(c_k) = \begin{pmatrix} \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}},k}(c_k) & 0 \\ 0 & \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{B}},k}(c_k) \end{pmatrix}, \qquad 1 < k < d,$$



(a) $c*\boldsymbol{\mathcal{X}}$ (b) $\boldsymbol{\mathcal{X}}_j+\boldsymbol{\mathcal{X}}_l$ (c) $\boldsymbol{\mathcal{X}}_j*\boldsymbol{\mathcal{X}}_l$

(d) Broadcast$_{c_1c_2c_3\to c_1c_2c_4c_3}$ with cost $\mathcal{O}(nr^2)$ (e) Broadcast$_{c_1c_2c_3\to c_1c_2c_3c_4}$ with cost $\mathcal{O}(n)$

Figure 7: Closed algebra over tensor trains: (a) scalar multiplication, (b) addition, (c) Hadamard product, and (d-e) broadcasting with two different insertion positions. The cost of broadcasting is given as the size of the inserted *identity cores*.

with the boundary cores stacked horizontally ($k = 1$) and vertically ($k = d$) so that $r_0 = r_d = 1$ is preserved. The TT ranks add: $r_k^{A+B} = r_k^A + r_k^B$, giving total storage $\mathcal{O}(dn(r_k^A + r_k^B)^2) = \mathcal{O}(dnr^2)$ for two equal-rank inputs.

Hadamard product (Figure 7c). The element-wise product $\boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}$ has cores given by the per-site Kronecker product of Equation (6),

$$\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}}*\boldsymbol{\mathcal{B}},k}(c_k) = \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}},k}(c_k) \otimes \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{B}},k}(c_k),$$

with multiplied ranks $r_k^{A*B} = r_k^A \cdot r_k^B$ and storage $\mathcal{O}(dn(r_k^A r_k^B)^2) = \mathcal{O}(dnr^4)$ for two equal-rank inputs. This quadratic per-product blow-up is the exponential rank growth discussed in Section 3.2 and motivates the recompression strategies of Section 3.3.

F.2 Broadcasting

The cores in Equation (6) assume that $\boldsymbol{\mathcal{X}}_j$ and $\boldsymbol{\mathcal{X}}_l$ are TTs over (i) the same scope and (ii) the same core-to-variable ordering. Assumption (ii) is easily enforced by fixing a global ordering σ of the variables in $\mathbf{c}$ and decomposing every factor in that ordering. Assumption (i) requires *broadcasting*: any TT whose scope $\mathbf{c}_j \subsetneq \mathbf{c}$ must be lifted to the common scope $\mathbf{c}$ before it can be multiplied with another TT.

Identity cores. For a TT $\boldsymbol{\mathcal{X}}_j$ with scope $\mathbf{c}_j$ decomposed in the ordering induced by σ, broadcasting to $\mathbf{c}$ inserts, for every missing variable $c_k \notin \mathbf{c}_j$, an *identity core* $\boldsymbol{\mathcal{I}}_k$ at the position dictated by σ:

$$\boldsymbol{\mathcal{I}}_k(c_k) = \mathbf{I}_{r_{k-1}}, \qquad c_k = 1, \ldots, |\mathcal{C}_k|,$$

where r_{k-1} is the rank of the bond at which $\boldsymbol{\mathcal{I}}_k$ is inserted, so that $\boldsymbol{\mathcal{I}}_k$ has shape $r_{k-1} \times |\mathcal{C}_k| \times r_{k-1}$. Since the slices of $\boldsymbol{\mathcal{I}}_k$ are constant in c_k, the broadcasted TT evaluates to the same value as $\boldsymbol{\mathcal{X}}_j$ for every assignment of the missing variables, as required.

Position matters. The position at which an identity core is inserted directly controls its size, and hence the storage and downstream multiplication cost of the broadcasted TT. If c_k is inserted *within the span* of $\mathbf{c}_j$ in σ – i.e., between two original cores – then $\boldsymbol{\mathcal{I}}_k$ has size $\mathcal{O}(nr^2)$, where r is the local rank of $\boldsymbol{\mathcal{X}}_j$ at that bond. If instead c_k is inserted *outside the span* – before the first or after the last variable of $\mathbf{c}_j$ – the surrounding bond rank is $r_0 = r_d = 1$ and $\boldsymbol{\mathcal{I}}_k$ collapses to a $1 \times |\mathcal{C}_k| \times 1$ core of size $\mathcal{O}(n)$, leaving the storage essentially unchanged. Figures 7d and 7e contrast the two cases: inserting the same variable c_4 between c_2 and c_3 costs $\mathcal{O}(nr^2)$, while appending it after c_3 costs only $\mathcal{O}(n)$.

Choosing a global variable ordering. Different global orderings σ result in different broadcasting costs because they change which missing variables fall inside or outside each factor's span. We seek a σ that minimizes the total cost of broadcasting all factors to their pairwise (or full) common

scopes. We assume for simplicity a uniform rank r and uniform mode size n across all TTs. [5] The broadcasting cost of factor ϕ_j under σ scales with its number of *interior gaps*: the number of positions inside the span of $\mathbf{c}_j$ that are occupied by variables outside $\mathbf{c}_j$. Concretely, denoting by $\mathrm{pos}_\sigma(c_i)$ the position of c_i in σ, the interior gap length of ϕ_j is

$$\mathrm{gaplen}_j(\sigma) = \Big(\max_{c_i \in \mathbf{c}_j} \mathrm{pos}_\sigma(c_i) - \min_{c_i \in \mathbf{c}_j} \mathrm{pos}_\sigma(c_i) + 1 \Big) - |\mathbf{c}_j|, \tag{12}$$

where the parenthesised term is the span of $\mathbf{c}_j$ in σ and $|\mathbf{c}_j|$ is the number of own variables it contains. We seek the ordering minimising the total gap length,

$$\min_\sigma \sum_{j=1}^{N} \mathrm{gaplen}_j(\sigma). \tag{13}$$

Spectral ordering. Equation (13) is a variant of the (NP-hard) minimum linear arrangement problem on the hypergraph whose hyperedges are the factor scopes $\mathbf{c}_j$. We solve it heuristically via *spectral seriation*, which is fast, parameter-free, and reliably produces orderings that are good enough for our pipelines.

We construct a weighted co-occurrence graph on the variables $\mathbf{c}$ with adjacency

$$\mathbf{W}_{ii'} = \sum_{j : c_i, c_{i'} \in \mathbf{c}_j} 1,$$

i.e., the number of factors in which c_i and $c_{i'}$ jointly appear. Variables that co-occur in many factors should sit close together in σ to keep factor spans short. Letting $D = \mathrm{diag}(\mathbf{W}\mathbf{1})$ be the degree matrix and $L = D - \mathbf{W}$ the graph Laplacian, the *Fiedler vector* – the eigenvector associated with the second-smallest eigenvalue of L – provides a one-dimensional embedding of the variables that approximately minimises the weighted sum of squared distances between adjacent vertices [3]. We sort the variables by their Fiedler-vector entries to obtain σ.

This ordering is computed once per WMC instance, before any TT is built, and is then reused for all factors.

G Multiplication Schedules

The bottom-up compilation pipeline of Section 4 pairs factors at two distinct stages: (i) the optional *pre-multiplication* of Step I, which groups dense factor tensors $\boldsymbol{\mathcal{X}}_j$ and multiplies them in dense space before applying TT-SVD (TT pipelines only), and (ii) the *circuit multiplication* of Step II, which iteratively pairs and multiplies the N compiled factor circuits into the $K \leq N$ circuits of Step III. In both stages, we intuitively aim to prioritize the multiplication of factors which share the most dependencies (that would otherwise be approximated by Step III) under some space constraints. The factors grow differently in the different stages, and we therefore use different strategies.

G.1 Dense Pre-multiplication: Hypergraph Partitioning

Before converting factors to TTs, we may multiply several sufficiently small factors exactly as dense tensors. This reduces the number of circuits passed to Step II and ensures that dependencies among factors in the same group are represented before any TT approximation is introduced. The grouping must balance two objectives: factors sharing variables should preferably be placed together, but the dense tensor produced by each group must fit within a prescribed memory budget B.

Grouping objective. Let $\mathcal{P}_1, \ldots, \mathcal{P}_K$ be a partition of the N initial factors. The factors in group $\mathcal{P}_k$ are replaced by their dense Hadamard product

$$\boldsymbol{\mathcal{X}}_{\mathcal{P}_k} = \prod_{j \in \mathcal{P}_k} \boldsymbol{\mathcal{X}}_j, \qquad \mathbf{c}_{\mathcal{P}_k} = \bigcup_{j \in \mathcal{P}_k} \mathbf{c}_j.$$

[5] In practice, TT-SVD yields non-uniform ranks, but estimating the exact post-broadcast cost would require recompiling every factor for every candidate σ, which is prohibitive. The uniform-rank approximation is enough to capture the essential combinatorial structure of the problem.

The resulting tensor contains one mode for every variable in the union of the grouped scopes and therefore has size$(\boldsymbol{\mathcal{X}}_{\mathcal{P}_k}) = \prod_{c_i \in \mathbf{c}_{\mathcal{P}_k}} |\mathcal{C}_i|$ entries. We require this size to remain below B.

We seek groups that share as few variables as possible with one another while satisfying this constraint. Equivalently, factors containing the same variable should be kept in the same group whenever the memory budget permits.

Hypergraph representation. This grouping problem has a natural hypergraph representation. We create one node for each factor and one hyperedge for each variable. The hyperedge for variable c_i connects exactly the factors whose scopes contain c_i:

$$e_i = \{j : c_i \in \mathbf{c}_j\}.$$

If the connected factors are assigned to different groups, then e_i is cut. A cut hyperedge therefore corresponds directly to a variable shared by multiple output groups. Minimizing the number of cut hyperedges implements the dependency objective above.

Memory constraint. The exact dense size of a group depends on the union of its factor scopes and is not additive across factors. Standard hypergraph partitioners, however, impose balance constraints using additive node weights. We therefore give factor j the conservative weight

$$w_j = \log\left(\prod_{c_i \in \mathbf{c}_j} |\mathcal{C}_i|\right).$$

For a group $\mathcal{P}_k$,

$$\sum_{j \in \mathcal{P}_k} w_j = \log\left(\prod_{j \in \mathcal{P}_k} \prod_{c_i \in \mathbf{c}_j} |\mathcal{C}_i|\right)$$

upper-bounds the logarithm of its actual dense size. The bound may count a shared variable more than once, whereas the dense product contains that variable only once. Consequently, enforcing

$$\sum_{j \in \mathcal{P}_k} w_j \leq \log B$$

is sufficient, though potentially conservative, for keeping the dense product within budget.

The resulting optimization is

$$\min_{\mathcal{P}_1,\ldots,\mathcal{P}_K} \left|\{\, i : e_i \text{ is cut} \,\}\right| \qquad \text{s.t.} \qquad \sum_{j \in \mathcal{P}_k} w_j \leq \log B \quad \forall k. \tag{14}$$

Thus, the objective keeps factors coupled by shared variables together, while the constraint limits the cost of materializing each resulting dense tensor.

Implementation. We approximately solve this constrained partitioning problem with KaHyPar [49]. Starting from $K = 1$, we increase the number of groups until the partition satisfies the dense-size budget. We use $B \in \{10^6, 10^9\}$ entries in our experiments. The partitioning overhead is negligible relative to dense multiplication and circuit compilation.

G.2 Circuit Multiplication Schedules

After Step I, the pipeline contains N compiled circuits, which Step II reduces to K through pairwise multiplications. The multiplication schedule balances two objectives. First, factors with important shared dependencies should be combined before the pipeline stops and approximates the remaining groups as independent. Second, intermediate products must remain small enough to store and, for recompressed TT multiplication, simple enough to approximate accurately under the available rank budget.

This scheduling problem is closely related to the *contraction-order* problem studied for tensor networks [22]. Standard tensor-network contraction normally continues until the entire network has been contracted, so the order is chosen primarily to minimize the sizes of intermediate tensors. Our setting introduces an additional concern: Step II may stop at $K > 1$, after which dependencies between the remaining groups are ignored by the fuzzy reduction. The schedule must therefore decide

not only how to control intermediate size, but also which dependencies should already have been incorporated when multiplication stops. TT recompression adds a further consideration: even when two schedules have comparable memory costs, they may expose different intermediate functions to rounding and consequently result in different approximation errors.

We explore three different scheduling strategies, each which their own focus within these objectives and considerations. These strategies are heuristic. Just as tensor network literature shows no single merge order strategies across problems [22, 21], we empirically find different strategies to perform better in different experiments.

Dependency-first matching. When no more specific program structure is available, we prioritize the first objective directly. At each level of a binary multiplication tree, we pair circuits with the largest scope intersections. Scope overlap serves as a proxy for shared dependencies: multiplying highly overlapping factors first keeps those dependencies inside the same circuit if the pipeline stops early, rather than leaving them to be discarded by the final independence approximation. We choose the pairs jointly through maximum-weight matching [18], with edge weights $w_{jl} = |\mathbf{c}_j \cap \mathbf{c}_l|$. We use this strategy in our Sudoku experiments.

Variable-elimination scheduling. When a program contains auxiliary variables, we can minimize its intermediate size by quickly eliminating these variables. Auxiliary variables encode dependencies between factors, but they are not queried and ultimately receive uniform weights. Once all factors containing such a variable have been multiplied, it can therefore be marginalized exactly. Doing so immediately can sharply and predictably reduce the scope and ranks of the intermediate circuit before the following multiplications. The variable-elimination schedule consequently prioritizes pairs whose multiplication makes the largest number of auxiliary variables available for marginalization. Note that this still serves the same dependency objective: all factors coupled through an eliminated variable are combined before that variable is summed out, so its induced dependency is retained exactly within the resulting factor. We use this strategy in our experiments with auxiliary variables (MNIST arithmetic and CLEVR-CountDiff).

Ordered contraction. Finally, we explore a strategy inspired by sequential boundary contraction in tensor-network algorithms [41, 50] which helps in intermediate TT rounding steps. Boundary-MPS methods progressively contract a tensor network from one side and compress the resulting boundary after each step. Their guiding principle is that each truncation approximates the complete network prefix contracted so far, rather than an arbitrary subset of its tensors. We transfer this principle to factor multiplication under the variable order of the TT. Each factor is assigned to the last variable in its scope. Factors assigned to the same variable are multiplied together, and the resulting groups are incorporated from left to right into a running product. After processing variable c_i, the accumulator therefore contains every factor whose scope is contained in the prefix $(c_1, \ldots, c_i)$. Consequently, each TT rounding acts on a complete factor prefix. Dependencies whose factors close within that prefix can interact before ranks are truncated, whereas an arbitrary binary tree may round disconnected or incomplete collections of factors and retain distinctions that later factors would make redundant. This schedule does not guarantee smaller ranks for every program, but it aligns intermediate approximation with the sequential decomposition underlying the TT. CLEVR-Diff uses this schedule.

G.3 Final contraction

Compilation generally leaves two TT factors (when feasible), which are contracted jointly with the query weights without materializing their product TT. This final contraction is exact and introduces no independence approximation or additional truncation. Because it avoids storing the inflated product cores, the inner product of m TTs with rank r is $O(r^m)$ rather than $O(r^2 m)$; with two TTs stored in $O(r^2)$ memory, this final contraction of two TTs is therefore virtually free in space complexity.

H Experiments (cont.)

H.1 Conditional WMC

In the conditional WMC setting, we still use a parameterized model $p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})$ over the symbolic concepts, but the logical constraint $\phi(\mathbf{c}, \mathbf{y}) : \mathcal{C} \times \mathcal{Y} \to \{0, 1\}$ now also depends on output variables

$\mathbf{y}$. The goal is to use the formula to derive a probability of $\mathbf{y}$ given $\mathbf{x}$:

$$p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) := \frac{\sum_{\mathbf{c}\in\mathcal{C}} p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})\phi(\mathbf{c},\mathbf{y})}{\sum_{\mathbf{y}'\in\mathcal{Y},\mathbf{c}\in\mathcal{C}} p_{\boldsymbol{\theta}}(\mathbf{c}|\mathbf{x})\phi(\mathbf{c},\mathbf{y}')} \tag{15}$$

which, given a label $\mathbf{l} \in \mathcal{Y}$, is optimized by maximizing $p_{\boldsymbol{\theta}}(\mathbf{l}|\mathbf{x})$.

The denominator is a normalizing constant that ensures $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$ is a valid distribution over $\mathbf{y}$. If, for every $\mathbf{c} \in \mathcal{C}$, there exists one and only one $\mathbf{y} \in \mathcal{Y}$ such that $\phi(\mathbf{c}, \mathbf{y}) = 1$, then the denominator simplifies to 1. Many works [8, 59, 62] take this assumption and write the symbolic program as a function $\mathbf{y} = f(\mathbf{c})$. We note however that this is generally not the case for Sudoku completion: some partial boards $\mathbf{c}$ could be completed in multiple ways, and some could not admit any completion. However, the SatNet dataset [66] is constructed so that every $\mathbf{c}$ has only one valid completion, such that in practice works that optimize the numerator alone [8] still achieve good results.

H.2 Sudoku compilation ablation

H.2.1 Evaluated pipelines

The Sudoku ablation plot in Figure 3 records the APRICOT compilation pipelines using

$$N \xrightarrow{p\times} \xrightarrow{R_{\text{init}}} \xrightarrow{e\times} \xrightarrow{t\times} K.$$

Here N is the initial factor count, p counts dense pre-multiplications, e counts exact TT merges, and t counts recompressed merges. The TT-SVD arrow records the initial rank setting. K is the resulting number of factors; if $K > 1$, the baseline uses an independent reduction. Note that some of these operations are optional (e.g., pre-multiplication, recompressions, ...); when they are absent, the arrow is omitted. We test different dense pre-multiplication budgets (none, 10^6, or 10^9 entries, see also Section G.1), initial ranks $R_{\text{init}} \in \{10, 30, 500\}$, recompression strategies from Section I with a maximum rank of 500, and various reduction targets K.

SDD pipelines are recorded as

$$N \xrightarrow{\text{KC}} \xrightarrow{e\times} K$$

where the KC arrow denotes exact compilation of the initial clauses, followed by any exact merge arrows. As for the TT pipelines, if $K > 1$, fuzzy approximations are used.

H.2.2 Multiplication schedule and rank budget

We further ablate the multiplication schedule and maximum rank on Sudoku 6×6, using TT-ROUND for recompressed TT multiplication and stopping at five factors before fuzzy reduction. The dependency-first schedule from Section G.2 prioritizes factors with overlapping scopes. Its intermediate products can therefore be more difficult to represent within the available rank budget, resulting in fewer exact merges than under a random schedule (169 rather than 183 at rank 500). Similarly, reducing the maximum rank to 50 also leaves fewer merges within the exact rank budget (144). This behavior is expected from the compilation algorithm; the relevant comparison is the accuracy and cost of the resulting pipelines.

Pipeline	Schedule	Rank	Time (s)	$\lvert\log_{10}(\text{MC}/\text{MC}_{\text{ref}})\rvert \downarrow$
$216 \xrightarrow{R=6} \xrightarrow{169\times} \xrightarrow{42\times} 5$	Dependency-first	500	42.4	1.30
$216 \xrightarrow{R=6} \xrightarrow{183\times} \xrightarrow{28\times} 5$	Random	500	94.7	3.41
$216 \xrightarrow{R=6} \xrightarrow{144\times} \xrightarrow{67\times} 5$	Dependency-first	50	5.1	1.97

Table 1: Ablation of the multiplication schedule and maximum TT rank on Sudoku 6×6. All pipelines use TT-ROUND and reduce the 216 initial factors to five before fuzzy reduction.

At rank 500, dependency-first matching is substantially more accurate than the random schedule. Although the random schedule admits more exact merges, its five remaining factors preserve less of the shared dependency structure before fuzzy reduction. The dependency-first matching is also surprisingly faster, perhaps because handling dependencies early on while the TTs are still small is more efficient than in later stage. Comparing the two dependency-first pipelines, rank 50 is much

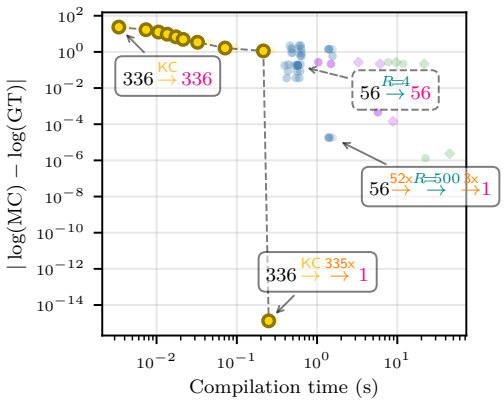


Figure 8: Pareto front of log inference error vs compilation time for Sudoku 4x4.

faster, consistent with the cubic dependence of TT multiplication on the rank, whereas rank 500 is more expressive and yields a more accurate model count. Notably, dependency-first matching at rank 50 remains more accurate than the random schedule at rank 500.

H.2.3 Sudoku 4x4

On Sudoku 4x4, we confirm that both SDD and APRICOT pipelines can derive the exact model count within a tiny ϵ error. As SDD can exactly reduce the formula into a single circuit, we also expect that approximate compilation can achieve a reduction to $K = 1$ with negligible error, which is what we observe in practice. We also notice that, as introducing approximate multiplications is slower than exact ones due to the overhead of recompression (though, they still compile within seconds), they are not beneficial in such small scale settings, and the best point on the Pareto front is achieved by the SDD pipelines.

H.3 Neurosymbolic tasks

We describe the Visual Sudoku, arithmetic and scene-understanding tasks used in our experiments. Perception predicts distributions over input MNIST digits or object attributes, and supervision is provided only on the program output. Auxiliary symbolic variables express intermediate computations and are unweighted: compiled pipelines marginalize them jointly, while independence reduction marginalizes them separately within each factor.

H.3.1 Definitions

ViSudo Classification. Let $z_{r,c} \in \{1, \ldots, 9\}$ denote the digit in row r and column c. Sudoku requires all digits within each row, column, and 3×3 box to be distinct. Writing E for the set of unordered pairs of distinct cells sharing at least one of these units, the constraint is

$$\phi(\mathbf{z}) := \bigwedge_{\{(r,c),(r',c')\} \in E} [z_{r,c} \neq z_{r',c'}].$$

Each factor is a 9×9 Boolean tensor with zeros on the diagonal and ones elsewhere. This factorization introduces no auxiliary symbolic variables.

ViSudo Completion. For completion, clue-cell digits are the perceptual concepts $\mathbf{c}$ and missing-cell digits are the output $\mathbf{y}$. The same constraint is evaluated on the combined board, giving $\phi(\mathbf{c}, \mathbf{y})$.

MNISTAdd. Two L-digit numbers A and B are represented by $2L$ MNIST images, and the output is the $L+1$ digits of their sum Y. Indexing digits from least to most significant, the constraints describe the rules of column addition as:

$$A_0 + B_0 = Y_0 + 10C_0, \qquad A_i + B_i + C_{i-1} = Y_i + 10C_i \quad (1 \leq i < L), \qquad Y_L = C_{L-1}$$

where C_i are binary carries. Each factor involves one column and its adjacent carries, giving a chain of local computations. The variable ordering of both SDD and APRICOT follows this structure.

MNISTMul. The inputs are again $2L$ MNIST images, and the output is the $2L$ digits of $A * B$. A coarse factorization would connect input digits and carries as

$$\sum_{i+j=k} A_i B_j + C_{k-1} = Y_k + 10C_k, \qquad 0 \le k < 2L,$$

with $C_{-1} = C_{2L-1} = 0$. However, in this setting, each factor still involves a large number of input digits which exceed our GPU memory at $L = 3$ in our implementation. We therefore derive a finer factorization that introduces partial products $P_{ij} = A_i B_j$ and computes each column sum $S_k = \sum_{i+j=k} P_{ij}$ through a chain of running partial sums. With these auxiliary variables and factors, the carry logic becomes

$$S_k + C_{k-1} = Y_k + 10C_k, \qquad 0 \le k < 2L.$$

with $S_{2L-1} = 0$. Notice that this is not possible for the fuzzy baseline, where an independent reduction of the finer factorization has ineffective learning gradients. We therefore retain the coarser carry formulation for fuzzy (explaining why fuzzy runs out of memory at $L = 3$).

CLEVR-Diff. Each of the m objects has a key K_i, predicted by the perception and consisting of the shape and color of the object. We use $D = 3 \times 8 = 24$ possible keys. To check whether the m objects are different, we evaluate a conjunction of factors that indicate whether all keys differ:

$$\phi(\mathbf{K}) := \bigwedge_{1 \le i < j \le m} K_i \neq K_j.$$

This constraint consists of $\binom{m}{2}$ pairwise inequalities and requires no auxiliary symbolic variables.

CLEVR-CountDiff. Here, we use keys consisting of the shape, material, and size of the object, with $D = 3 \times 2 \times 2 = 12$ possibilities. The output is the number of distinct keys, $Y = |\{K_1, \ldots, K_m\}|$ in the scene. The factorization uses auxiliary binary variables $S_{v,i}$ that record whether key v occurs among the first i objects. Then, running counts T_v accumulate the number of keys present:

$$\begin{aligned} &S_{v,0} = 0, \quad S_{v,i} \leftrightarrow S_{v,i-1} \vee [K_i = v], \\ &T_{-1} = 0, \quad T_v = T_{v-1} + S_{v,m}, \qquad Y = T_{D-1}, \end{aligned}$$

for $0 \le v < D$ and $1 \le i \le m$.

H.3.2 Datasets

Visual Sudoku. Both tasks use 9×9 boards with observed digits represented by MNIST images. ViSudo-Classification uses ViSudo-PC [4]: the label indicates whether a board satisfies the Sudoku rules. Training uses only valid boards, while evaluation includes valid and invalid boards. ViSudo-Completion uses the SatNet dataset [66], with the data-leak correction of Chang et al. [6]. Observed clue cells provide the image inputs, and the digits in the remaining cells are the supervised targets. We use the benchmark training and test splits.

Arithmetic. We construct examples separately from the MNIST training and test splits: disjoint groups of $2L$ images are used for the input numbers, and their digit labels determine the arithmetic output. This gives $\lfloor 60{,}000/(2L) \rfloor$ training and $\lfloor 10{,}000/(2L) \rfloor$ test examples (therefore, increasing the input length L also reduces the number of available training examples). The generated datasets are shared across methods and run seeds.

Scene-understanding. We extract object crops from annotated CLEVR-Hans3 scenes [54], resize them to 32×32, and assemble sets of m objects as newly constructed scenes. For CLEVR-Diff, we sample distinct keys and, with probability $1/2$, replace one by a duplicate of another, balancing the two output classes. For CLEVR-CountDiff, keys are drawn independently and uniformly. Each key is instantiated with a randomly selected crop of that type. Training and test examples use crops exclusively from their respective source splits. We generate 10,000 training and 2,000 test examples for each task size, using fixed datasets across methods and runs. Recombining crops permits larger object sets than the original scenes and keeps the per-object sampling distribution fixed across m.

H.4 WMC evaluation

For each arithmetic and CLEVR circuit, we evaluate 64 random weightings, shared across methods using a fixed seed. In each draw, every non-auxiliary variable receives an independently sampled

$\mathrm{Dirichlet}(\mathbf{1})$ weight vector, uniform over its probability simplex. These include input digits or object keys and, where represented explicitly, output digits or counts. Auxiliary variables receive unit weights and are summed out. No evidence is fixed: the query sums over all satisfying assignments, including all output values. CLEVR-Diff evaluates the all-different constraint without an explicit output variable. These random queries assess circuit fidelity rather than performance under learned neural predictions or observed-label evidence.

Let WMC_b and $\mathrm{W}\,\widehat{\mathrm{M}}\mathrm{C}_b$ denote the reference and estimated WMC for draw b. For weight draw b, we measure the absolute log-ratio error

$$e_b = \left| \log_{10} \frac{\mathrm{W}\,\widehat{\mathrm{M}}\mathrm{C}_b}{\mathrm{WMC}_b} \right|.$$

We report the median of the 64 resulting errors,

$$E_{\mathrm{WMC}} = \mathrm{median}\{e_1, \ldots, e_{64}\}.$$

Thus, $E_{\mathrm{WMC}} = 1$ means that, for a typical sampled weighting, the estimated WMC differs from the reference by a factor of 10.

Exact references. In tasks where exact compilation breaks in the harder settings (MNISTMul, CLEVR-Diff, CLEVR-CountDiff), the exact references WMC_b are derived using task-specific solutions developed separately for each benchmark. These specialized solutions used dynamic programming, inclusion-exclusion, or brute-force enumeration; they are not general-purpose WMC algorithms and are not included as competing methods. The procedures were developed with the assistance of AI and then checked against exact SDD evaluation on the easier instances for which SDD compilation was feasible.

- **MNISTMul.** For every pair of L-digit input numbers (A, B), we compute the unique product $Y = AB$. The contribution of this assignment is the product of the weights assigned to the digits of A, B, and Y. Summing these contributions over all 10^{2L} input pairs gives the exact WMC. This direct enumeration is practical for the evaluated instances, where $L \leq 3$.
- **CLEVR-Diff.** The required WMC is the total weight of assignments in which all m objects receive different keys. We process the D possible key values one at a time. A dynamic-programming state records the subset S of objects that have already been assigned distinct processed keys. For the next key, we either leave it unused or assign it to one object not in S, multiplying by that object's weight for the key. After all keys have been processed, the state containing all objects is the exact all-different WMC. The algorithm has $\mathcal{O}(2^m)$ states and costs $\mathcal{O}(Dm2^m)$.
- **CLEVR-CountDiff.** Here we require the weighted distribution of the number of distinct keys. For every subset B of the D possible key values, we first compute

  $$q(B) = \prod_{i=1}^{m} \sum_{v \in B} p_i(v),$$

  the total weight of assignments in which every object chooses a key from B. These quantities include assignments using all of B as well as assignments using only a smaller subset. Inclusion–exclusion removes this repeated counting and recovers the total weight of assignments using exactly t distinct keys, for every t. The output weights are then applied to this exact distribution. The computation is exponential in the key-domain size D and linear in the number of objects m, making it practical here because $D = 12$.

H.5 Training and evaluation details

APRICOT configurations. The grid of APRICOT configurations evaluated in the Sudoku compilation ablation is detailed in Section H.2. For the Visual Sudoku learning experiments, the selected compilation stage uses dense pre-multiplication groups of at most 10^6 entries, leaving 113 initial factors, an initial TT-SVD rank 30, and a maximum rank of 500 during multiplications. For arithmetic and CLEVR, the selected APRICOT C configurations skip pre-multiplications and use TT-ROUND

for product recompression, with maximum ranks below of 500 for all experiments, except MNIST-Mul where we allow a maximum rank of 1200. The multiplication schedules are different in each experiment and detailed in Section G.2.

Compute budgets. The reported configuration uses one NVIDIA A10G GPU with approximately 22 GiB memory. TT compilation runs on the GPU. SDD compilation runs on the CPU process with a 64 GiB memory cap and a two-hour time limit for the compilation study. Training has a six-hour wall-clock limit. Runs exceeding these limits are reported as out of memory or timed out, without increasing the budget. Results are averaged over ten runs.

Learning objectives. We maximize the WMC in Equation (1) and conditional WMC in Equation (15) by setting the weights of the concepts $\mathbf{c}$ to predicted neural probabilities and the weights of the output variables $\mathbf{y}$ to the given one-hot labels, and by marginalizing the circuit. Auxiliary variables receive unit weights. In practice, we maximize the logarithm of the resulting WMC.

Inference. In the conditional WMC setting (Section H.1), two strategies can be used to predict the output $\mathbf{y}$ from the latent concepts $\mathbf{c}$ at test time. The first, *greedy decoding*, selects the most likely concept prediction from the neural predictor $\operatorname{argmax}_{\mathbf{c}} p(\mathbf{c}|x)$. The concepts are then fed to a symbolic solver to evaluate the constraint or derive output symbols $\mathbf{y}$, e.g., by running a Sudoku solver on the partially filled concepts or computing ordinary exact addition/multiplication for arithmetic. Under this strategy, the WMC is not used at inference time; only during learning. We use greedy decoding for the Visual Sudoku experiments. The second, *reasoning-based decoding*, uses the WMC at inference to predict $\operatorname{argmax}_{\mathbf{y}} \sum_{\mathbf{c}} p(\mathbf{c}|x)\phi(\mathbf{c}, \mathbf{y})$ – or doing this independently for each y_k. This approach can pick up on subtle inconsistencies in the predictions and correct them based on the constraints. We use it for the arithmetic and CLEVR experiments.

Architectures and hyperparameters. We use conventional CNNs to process each input image independently (an MNIST image or an object crop) and produce a softmax distribution over its symbol. Architecture details are provided in the repository. We use Adam [26] with the following settings, together with usual techniques such as gradient clipping. There is no validation-based selection or early stopping: metrics use the final epoch.

Task	Learning rate	Batch size	Epochs
MNISTAdd	10^{-3}	32 (8 for $L = 100$)	30
MNISTMul	10^{-3}	32	10
CLEVR-Diff / CountDiff	10^{-3}	32	40
ViSudo-Completion	10^{-5}	8	20
ViSudo-Classification	10^{-3}	20	2000

I Approximation Algorithms for TT Multiplication

In this appendix, we detail the four recompression strategies introduced in Section 3.3. All algorithms take as input two compatible TTs $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ (same scope, same core ordering, same mode dimensions n_k) and a target maximum rank R or Frobenius tolerance ϵ, and return a TT approximation $\tilde{\boldsymbol{\mathcal{X}}}$ of the Hadamard product $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}$. We denote by $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}},k} \in \mathbb{R}^{r^{\mathcal{A}}_{k-1} \times n_k \times r^{\mathcal{A}}_k}$ the k-th core of $\boldsymbol{\mathcal{A}}$, and similarly for $\boldsymbol{\mathcal{B}}$ and $\tilde{\boldsymbol{\mathcal{X}}}$.

Notation. We follow einsum and einops conventions for tensor operations [47]. Given tensors $\boldsymbol{\mathcal{T}}_1, \boldsymbol{\mathcal{T}}_2$:

- EINSUM(`"abc, cde -> abde"`, $\boldsymbol{\mathcal{T}}_1, \boldsymbol{\mathcal{T}}_2$) denotes Einstein summation, contracting indices that appear in both inputs but not in the output.
- REARRANGE(`"(a b) c -> a b c"`, $\boldsymbol{\mathcal{T}}$) denotes a reshape/permutation, where parentheses on the left split a single axis into multiple axes (with explicit sizes given as side arguments) and parentheses on the right merge multiple axes into a single axis.

I.1 TT-ROUND

TT-ROUND [43] computes $\tilde{\boldsymbol{\mathcal{X}}} \approx \boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}$ by interleaving the Kronecker product of cores with a sweep of truncated SVDs. The naive variant first materializes the inflated cores $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},k} = \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}},k} \otimes \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{B}},k}$ of

Algorithm 1 Hadamard TT-ROUND

Require: TTs $\mathcal{A}, \mathcal{B}$ with cores $\{\mathcal{T}_{\mathcal{A},k}\}_{k=1}^{d}$, $\{\mathcal{T}_{\mathcal{B},k}\}_{k=1}^{d}$; max rank R; tolerance ϵ.
Ensure: TT approximation $\tilde{\mathcal{X}} \approx \mathcal{A} * \mathcal{B}$.

1: $\mathcal{A} \leftarrow$ RIGHTORTHOGONALIZE($\mathcal{A}$)
2: $\mathcal{B} \leftarrow$ RIGHTORTHOGONALIZE($\mathcal{B}$)
3: $\mathcal{L} \leftarrow \mathbf{1} \in \mathbb{R}^{1\times 1\times 1}$ ▷ initial left environment
4: **for** $k = 1, \ldots, d$ **do**
5: $\quad \mathcal{W} \leftarrow$ EINSUM("r ra rb, ra n ra', rb n rb' -> r n ra' rb'", $\mathcal{L}$, $\mathcal{T}_{\mathcal{A},k}$, $\mathcal{T}_{\mathcal{B},k}$)
6: $\quad \mathcal{W} \leftarrow$ REARRANGE("r n ra' rb' -> (r n) (ra' rb')", $\mathcal{W}$)
7: $\quad$ **if** $k < d$ **then**
8: $\quad\quad \mathbf{U}, \mathbf{s}, \mathbf{V}^\top \leftarrow$ TRUNCATEDSVD($\mathcal{W}$; ϵ, R)
9: $\quad\quad \mathcal{T}_{\tilde{\mathcal{X}},k} \leftarrow$ REARRANGE("(r n) r' -> r n r'", $\mathbf{U}$)
10: $\quad\quad \mathcal{L} \leftarrow$ REARRANGE("r' (ra' rb') -> r' ra' rb'", $\mathrm{diag}(\mathbf{s})\,\mathbf{V}^\top$)
11: $\quad$ **else**
12: $\quad\quad \mathcal{T}_{\tilde{\mathcal{X}},d} \leftarrow$ REARRANGE("(r n) 1 -> r n 1", $\mathcal{W}$)
13: $\quad$ **end if**
14: **end for**
15: **return** TT $\tilde{\mathcal{X}}$ with cores $\{\mathcal{T}_{\tilde{\mathcal{X}},k}\}_{k=1}^{d}$.

rank $r_k^A r_k^B$ and then applies a standard rounding sweep, paying $\mathcal{O}(dnr^4)$ memory for the inflated cores. The *single-pass* variant presented here exploits the Matrix-Kronecker structure of the inflated cores to avoid this materialization, reducing the memory cost to $\mathcal{O}(r^3)$.

The algorithm proceeds in two phases:

1. **Right-orthogonalization.** Both $\mathcal{A}$ and $\mathcal{B}$ are first right-orthogonalized so that the right unfolding of every core is row-orthonormal. This ensures that the subsequent local SVD truncations are quasi-optimal with respect to the global Frobenius norm.

2. **Left-to-right SVD sweep.** A left environment $\mathcal{L} \in \mathbb{R}^{r_{k-1}\times r^A_{k-1}\times r^B_{k-1}}$ accumulates the already-compressed prefix. At each site k, the algorithm (i) contracts $\mathcal{L}$ with the (implicit) Kronecker slice $\mathcal{T}_{\mathcal{A},k} \otimes \mathcal{T}_{\mathcal{B},k}$ via a single einsum, producing a working tensor $\mathcal{W}$; (ii) flattens $\mathcal{W}$ into a matrix and performs a truncated SVD with target rank R (or tolerance ϵ); (iii) stores the left singular vectors as the new core $\mathcal{T}_{\tilde{\mathcal{X}},k}$ and absorbs $\mathrm{diag}(\mathbf{s})\mathbf{V}^\top$ into the next left environment.

The last core requires no SVD truncation since its rightmost rank is already 1 by definition.

Randomized per-site SVD. When the target rank R is small relative to the working tensor dimensions, the truncated SVD on $\mathcal{W}$ (Algorithm 1, line 8) can be replaced by a randomized low-rank SVD [23]: $\mathcal{W}$ is multiplied by a Gaussian sketch of width $R + r_{\text{oversample}}$, and the dominant left singular vectors are recovered from the resulting tall-and-skinny matrix. We emphasize that this is a purely *local*, matrix-level acceleration — a fresh sketch is drawn independently at each site, and the sweep structure, orthogonalization, and per-site target ranks are unchanged. This is conceptually distinct from the TT-level sketching used by HATT (Section I.2), which draws a single random TT once and reuses it across all sites by precomputing right environments, so that no unfolding of $\mathcal{X}$ is ever formed.

Complexity. At site k, $\mathcal{W}$ has shape $r_{k-1}n_k \times r_k^A r_k^B$ and its truncated SVD costs $\mathcal{O}(nr^4)$ for uniform ranks r. Summed over d sites, this gives $\mathcal{O}(dnr^4)$ time. Memory is dominated by storing $\mathcal{W}$ and the left environment, both of which are $\mathcal{O}(r^3)$ since the Kronecker factors are never explicitly formed. When randomized per-site SVDs are used in place of deterministic ones, the per-site cost drops to $\mathcal{O}(nr^3)$ since the SVD operates on a sketched matrix of column dimension $\mathcal{O}(r)$ rather than on the full $r_k^A r_k^B = r^2$ unfolding, giving an overall time complexity of $\mathcal{O}(dnr^3)$ — matching that of HATT (Section I.2).

Algorithm 2 Sketched right environments for $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}$

Require: TT cores $\{\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}},k}\}_{k=1}^{d}, \{\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{B}},k}\}_{k=1}^{d}$; sketch TT cores $\{\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{S}},k}\}_{k=1}^{d}$.
Ensure: Right environments $\{\boldsymbol{\mathcal{R}}_k\}_{k=1}^{d-1}$.
1: $\boldsymbol{\mathcal{R}}_d \leftarrow \mathbf{1} \in \mathbb{R}^{1\times 1\times 1}$
2: **for** $k = d, d-1, \ldots, 2$ **do**
3: $\quad \boldsymbol{\mathcal{R}}_{k-1} \leftarrow \text{EINSUM}\big(\texttt{"ra n ra', rb n rb', s n s', ra' rb' s' -> ra rb s"},$
$\qquad\qquad \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}},k},\ \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{B}},k},\ \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{S}},k},\ \boldsymbol{\mathcal{R}}_k\big)$
4: **end for**
5: **return** $\{\boldsymbol{\mathcal{R}}_k\}_{k=1}^{d-1}$

I.2 HATT

HATT [55] replaces the deterministic SVD-based truncation of TT-ROUND with a randomized sketching approach inspired by randomized linear algebra [23, 10]. Like TT-ROUND, it exploits the Matrix-Kronecker structure of the inflated cores $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}},k} \otimes \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{B}},k}$ to avoid materializing them. Unlike TT-ROUND— whose per-site SVDs (deterministic or randomized) still operate on full unfoldings of the inflated working tensor $\boldsymbol{\mathcal{W}}$ — HATT projects $\boldsymbol{\mathcal{X}}$ against a TT sketch $\boldsymbol{\mathcal{S}}$ of width $s = R + r_{\text{oversample}}$ first and applies SVDs only on small sketched matrices of column dimension s.

The algorithm proceeds in two phases.

Phase 1: Sketched right environments. A random TT sketch $\boldsymbol{\mathcal{S}}$ with cores $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{S}},k} \in \mathbb{R}^{s_{k-1}\times n_k \times s_k}$ of width s is drawn (e.g., with Gaussian or Rademacher entries) and left-orthogonalized for numerical stability. A right-to-left sweep (Algorithm 2) then produces the right environments $\boldsymbol{\mathcal{R}}_k \in \mathbb{R}^{r_k^A \times r_k^B \times s_k}$ for $k = 1, \ldots, d-1$. Each $\boldsymbol{\mathcal{R}}_k$ represents the contraction of the inflated cores at positions $k+1, \ldots, d$ projected against the right tail of $\boldsymbol{\mathcal{S}}$.

As an extension to the original HATT formulation, we additionally allow the user to provide a TT $\boldsymbol{\mathcal{G}}$ as an initial guess of the solution, which is then used in place of a random sketch. This can leverage prior knowledge about $\boldsymbol{\mathcal{X}}$ – for instance, the result of a previous Hadamard product in a bottom-up compilation – to bias the sketch towards the relevant subspace.

Phase 2: Left-to-right projection sweep. A left-to-right sweep analogous to that of TT-ROUND constructs the cores of $\tilde{\boldsymbol{\mathcal{X}}}$. At each site k, the algorithm contracts the left environment $\boldsymbol{\mathcal{L}}$, the inflated core $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}},k} \otimes \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{B}},k}$, and the sketched right environment $\boldsymbol{\mathcal{R}}_k$ into a small working tensor $\mathbf{M}$ of shape $r_{k-1} \times n_k \times s_k$, where r_{k-1} is the running compressed rank. A truncated SVD on the unfolding of $\mathbf{M}$ then yields the left singular vectors used as the new core $\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},k}$. The left environment is updated by contracting the new core back against the inflated cores, and the loop proceeds. As in TT-ROUND, the last core requires no SVD since $s_d = 1$.

Complexity. The dominant cost at each site is the truncated SVD on $\mathbf{M}$ of shape $r_{k-1}n_k \times s_k$, costing $\mathcal{O}(nr^2 s)$ under uniform ranks. Summed over the d sites, this gives $\mathcal{O}(dnr^2 s)$ time. Memory is dominated by storing the right environments, at $\mathcal{O}(r^2 s)$ each. The right-environment sweep (Algorithm 2) has the same asymptotic cost of $\mathcal{O}(dnr^2 s)$. For $s \approx r$, the overall time becomes $\mathcal{O}(dnr^3)$.

I.3 TCI

Tensor Cross Interpolation (TCI) [42, 39] approximates the Hadamard product $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}$ by a TT $\tilde{\boldsymbol{\mathcal{X}}}$ that interpolates $\boldsymbol{\mathcal{X}}$ at a small set of carefully chosen entries. Unlike TT-ROUND and HATT, TCI never materializes a Kronecker core nor performs an SVD on a full unfolding of $\boldsymbol{\mathcal{X}}$: at each site, a small *supercore* is formed by contracting a w-site window of the input cores with their interpolated environments, and an interpolative decomposition picks the rows or columns of that supercore that best interpolate it. We present the algorithm with two-site sweeps ($w = 2$); the generalization to $K > 2$ factors and larger windows is straightforward.

Pivots and shared environments. Let $p \leq R$ denote the current number of pivots at a given bond. TCI maintains per-factor environments that share the pivot axis across factors:

Algorithm 3 Hadamard HATT

Require: TTs $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}$; max rank R; tolerance ϵ; sketch width $s = R + r_{\text{oversample}}$; (optional) initial guess TT $\boldsymbol{\mathcal{G}}$.
Ensure: TT approximation $\tilde{\boldsymbol{\mathcal{X}}} \approx \boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}$.

$\boldsymbol{\mathcal{A}} \leftarrow$ RIGHTORTHOGONALIZE($\boldsymbol{\mathcal{A}}$)
$\boldsymbol{\mathcal{B}} \leftarrow$ RIGHTORTHOGONALIZE($\boldsymbol{\mathcal{B}}$)
$\boldsymbol{\mathcal{S}} \leftarrow \boldsymbol{\mathcal{G}}$ if provided, else RANDOMTT(dims, s)
$\boldsymbol{\mathcal{S}} \leftarrow$ LEFTORTHOGONALIZE($\boldsymbol{\mathcal{S}}$)
$\{\boldsymbol{\mathcal{R}}_k\}_{k=1}^{d-1} \leftarrow$ SKETCHEDRIGHTENVS($\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}, \boldsymbol{\mathcal{S}}$) ▷ Algorithm 2
$\boldsymbol{\mathcal{R}}_d \leftarrow \mathbf{1} \in \mathbb{R}^{1\times1\times1}$
$\boldsymbol{\mathcal{L}} \leftarrow \mathbf{1} \in \mathbb{R}^{1\times1\times1}$ ▷ initial left environment
for $k = 1, \ldots, d$ **do**
    $\mathbf{M} \leftarrow$ EINSUM("r ra rb, ra n ra', rb n rb', ra' rb' s' -> r n s'", $\boldsymbol{\mathcal{L}}$, $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}},k}$, $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{B}},k}$, $\boldsymbol{\mathcal{R}}_k$)
    if $k < d$ **then**
        $\mathbf{M} \leftarrow$ REARRANGE("r n s' -> (r n) s'", $\mathbf{M}$)
        $\mathbf{U}, \cdot, \cdot \leftarrow$ TRUNCATEDSVD($\mathbf{M}$; ϵ, R)
        $\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},k} \leftarrow$ REARRANGE("(r n) r' -> r n r'", $\mathbf{U}$)
        $\boldsymbol{\mathcal{L}} \leftarrow$ EINSUM("r n r', r ra rb, ra n ra', rb n rb' -> r' ra' rb'", $\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},k}$, $\boldsymbol{\mathcal{L}}$, $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}},k}$, $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{B}},k}$)
    else
        $\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},d} \leftarrow \mathbf{M}$ ▷ $s_d = 1$, so $\mathbf{M}$ already has shape $r_{d-1} \times n_d \times 1$
    end if
end for
return TT $\tilde{\boldsymbol{\mathcal{X}}}$ with cores $\{\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},k}\}_{k=1}^{d}$.

- Left environments $\boldsymbol{\mathcal{L}}_i^{\boldsymbol{\mathcal{A}}} \in \mathbb{R}^{p\times r_{i-1}^A}$ and $\boldsymbol{\mathcal{L}}_i^{\boldsymbol{\mathcal{B}}} \in \mathbb{R}^{p\times r_{i-1}^B}$ holding p contracted slices of the left tail of $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ respectively.
- Right environments $\boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{A}}} \in \mathbb{R}^{r_i^A\times p}$ and $\boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{B}}} \in \mathbb{R}^{r_i^B\times p}$ holding p contracted slices of the right tails.

The shared pivot axis encodes the Hadamard product structure without ever forming Kronecker cores: at each bond, both factors are interpolated at the *same* set of physical-pivot indices.

Interpolative primitives. We use two primitives for selecting pivots and interpolating from them.

- MAXVOL [20]: given a matrix $\mathbf{M}$, MAXVOL($\mathbf{M}$; R) returns up to R column indices of $\mathbf{M}$ that form a near-maximum-volume submatrix.
- PRRLDU [48]: a partially rank-revealing LU decomposition. Its row-pivoted variant $\text{PRRLDU}_{\text{row}}(\mathbf{M}; \epsilon, R)$ returns an interpolation matrix $\boldsymbol{\mathcal{Z}}$ and a pivot set such that $\mathbf{M} \approx \boldsymbol{\mathcal{Z}} \cdot \mathbf{M}[\text{pivot}, :]$. The column-pivoted variant $\text{PRRLDU}_{\text{col}}$ analogously yields $\mathbf{M} \approx \mathbf{M}[:, \text{pivot}] \cdot \boldsymbol{\mathcal{Z}}$.

Phase 1: Maxvol warm-up. A right-to-left sweep (Algorithm 4) initializes the right environments. At each site, per-factor supercores $\boldsymbol{\Phi}_{\boldsymbol{\mathcal{A}}}, \boldsymbol{\Phi}_{\boldsymbol{\mathcal{B}}}$ are formed by contracting the current right environment with the corresponding core; their joint Hadamard matrix $\boldsymbol{\Phi}$ has rows that range over the outer product of the per-factor left bonds and columns that range over the shared physical-pivot indices. MAXVOL selects up to R such columns, and the picked columns of each $\boldsymbol{\Phi}_{\boldsymbol{\mathcal{A}}}, \boldsymbol{\Phi}_{\boldsymbol{\mathcal{B}}}$ become the next right environments.

Phase 2: Alternating sweeps. Sweeps of Algorithm 5 (L2R) and Algorithm 6 (R2L) refine the pivots and emit the cores of $\tilde{\boldsymbol{\mathcal{X}}}$. At each interior site of an L2R sweep, per-factor supercores are formed by contracting the left environment, two consecutive cores, and the right environment. They are combined element-wise into the joint supercore $\boldsymbol{\Phi}$, flattened into a matrix, and decomposed by $\text{PRRLDU}_{\text{row}}$ into an interpolation matrix $\boldsymbol{\mathcal{Z}}$ and a row pivot. The interpolation matrix becomes

Algorithm 4 Maxvol right environments for $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}$

Require: TTs $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}$; max rank R.
Ensure: Right environments $\{\boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{B}}}\}_{i=1}^{d-1}$.

$\boldsymbol{\mathcal{R}}_d^{\boldsymbol{\mathcal{A}}} \leftarrow \mathbf{1} \in \mathbb{R}^{1\times 1}$; $\boldsymbol{\mathcal{R}}_d^{\boldsymbol{\mathcal{B}}} \leftarrow \mathbf{1} \in \mathbb{R}^{1\times 1}$
for $i = d, d-1, \ldots, 2$ **do**
  for $\boldsymbol{\mathcal{Y}} \in \{\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}\}$ **do**
    $\boldsymbol{\Phi}_{\boldsymbol{\mathcal{Y}}} \leftarrow \text{EINSUM}\big(\texttt{"r n r', r' p -> r (n p)"},\ \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{Y}},i},\ \boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{Y}}}\big)$
  end for
  $\boldsymbol{\Phi} \leftarrow \text{EINSUM}\big(\texttt{"ra q, rb q -> (ra rb) q"},\ \boldsymbol{\Phi}_{\boldsymbol{\mathcal{A}}},\ \boldsymbol{\Phi}_{\boldsymbol{\mathcal{B}}}\big)$
  pivot $\leftarrow \text{MAXVOL}(\boldsymbol{\Phi};\ R)$
  for $\boldsymbol{\mathcal{Y}} \in \{\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}\}$ **do**
    $\boldsymbol{\mathcal{R}}_{i-1}^{\boldsymbol{\mathcal{Y}}} \leftarrow \boldsymbol{\Phi}_{\boldsymbol{\mathcal{Y}}}[:, \text{pivot}]$
  end for
end for
return $\{\boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{B}}}\}_{i=1}^{d-1}$

Algorithm 5 Left-to-right TCI sweep

Require: TTs $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}$; left envs $\{\boldsymbol{\mathcal{L}}_i^{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{L}}_i^{\boldsymbol{\mathcal{B}}}\}_{i=1}^{d-1}$; right envs $\{\boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{B}}}\}_{i=1}^{d}$; max rank R; tolerance ϵ.
Ensure: Cores $\{\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},i}\}_{i=1}^{d}$ and updated left envs.

$\boldsymbol{\mathcal{L}}_1^{\boldsymbol{\mathcal{A}}} \leftarrow \mathbf{1} \in \mathbb{R}^{1\times 1}$; $\boldsymbol{\mathcal{L}}_1^{\boldsymbol{\mathcal{B}}} \leftarrow \mathbf{1} \in \mathbb{R}^{1\times 1}$
for $i = 1, \ldots, d-1$ **do**
  for $\boldsymbol{\mathcal{Y}} \in \{\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}\}$ **do**
    $\boldsymbol{\Phi}_{\boldsymbol{\mathcal{Y}}} \leftarrow \text{EINSUM}\big(\texttt{"p r, r n r', r' m r'', r'' q -> p n m q"},\ \boldsymbol{\mathcal{L}}_i^{\boldsymbol{\mathcal{Y}}},\ \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{Y}},i},\ \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{Y}},i+1},\ \boldsymbol{\mathcal{R}}_{i+1}^{\boldsymbol{\mathcal{Y}}}\big)$
  end for
  $\boldsymbol{\Phi} \leftarrow \boldsymbol{\Phi}_{\boldsymbol{\mathcal{A}}} * \boldsymbol{\Phi}_{\boldsymbol{\mathcal{B}}}$ ▷ element-wise Hadamard
  $\boldsymbol{\Phi} \leftarrow \text{REARRANGE}\big(\texttt{"p n m q -> (p n) (m q)"},\ \boldsymbol{\Phi}\big)$
  $(\boldsymbol{\mathcal{Z}},\ \text{pivot}) \leftarrow \text{PRR}\text{LDU}_{\text{row}}(\boldsymbol{\Phi};\ \epsilon,\ R)$
  $\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},i} \leftarrow \text{REARRANGE}\big(\texttt{"(p n) r' -> p n r'"},\ \boldsymbol{\mathcal{Z}}\big)$
  for $\boldsymbol{\mathcal{Y}} \in \{\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}\}$ **do**
    $\boldsymbol{\mathcal{L}}_{i+1}^{\boldsymbol{\mathcal{Y}}} \leftarrow \text{REARRANGE}\big(\text{EINSUM}\big(\texttt{"p r, r n r' -> (p n) r'"},\ \boldsymbol{\mathcal{L}}_i^{\boldsymbol{\mathcal{Y}}},\ \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{Y}},i}\big)\big)[\text{pivot}, :]$
  end for
  if $i = d-1$ **then**
    $\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},d} \leftarrow \text{REARRANGE}\big(\texttt{"r' n -> r' n 1"},\ \boldsymbol{\Phi}[\text{pivot}, :]\big)$
  end if
end for
return cores $\{\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},i}\}_{i=1}^{d}$, updated $\{\boldsymbol{\mathcal{L}}_i^{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{L}}_i^{\boldsymbol{\mathcal{B}}}\}_{i=1}^{d}$

the new core $\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},i}$, and the pivot updates the left environments of $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$. The R2L sweep is symmetric, processing sites from right to left and using $\text{PRR}\text{LDU}_{\text{col}}$ to update right environments.

Boundary cores. The L2R sweep additionally extracts the rightmost core $\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},d}$ directly from the boundary supercore at $i = d-1$ (whose right rank is already $r_d = 1$). The R2L sweep mirrors this for the leftmost core.

Complexity. At each interior site, the dominant cost is forming the joint supercore $\boldsymbol{\Phi}$ and applying PRRLDU to its $(p \cdot n) \times (n \cdot p)$ unfolding. The per-factor four-tensor contractions cost $\mathcal{O}(n^2 p r^2)$, the elementwise Hadamard costs $\mathcal{O}(n^2 p^2)$, and PRRLDU costs $\mathcal{O}(n^2 p^3)$ in the worst case. With $p \le R \approx r$, the per-site cost is $\mathcal{O}(n^2 r^3)$. Summing over the d sites and a constant number of sweeps gives an overall time complexity of $\mathcal{O}(d n^2 r^3)$ and memory $\mathcal{O}(n^2 r^2)$, dominated by $\boldsymbol{\Phi}$. The maxvol warm-up has lower per-site cost, $\mathcal{O}(nr^3)$, and is dominated by the sweep phase.

Algorithm 6 Right-to-left TCI sweep

Require: TTs $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}$; left envs $\{\boldsymbol{\mathcal{L}}_i^{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{L}}_i^{\boldsymbol{\mathcal{B}}}\}_{i=1}^{d}$; right envs $\{\boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{B}}}\}_{i=1}^{d-1}$; max rank R; tolerance ϵ.
Ensure: Cores $\{\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},i}\}_{i=1}^{d}$ and updated right envs.

$\boldsymbol{\mathcal{R}}_d^{\boldsymbol{\mathcal{A}}} \leftarrow \mathbf{1}$; $\boldsymbol{\mathcal{R}}_d^{\boldsymbol{\mathcal{B}}} \leftarrow \mathbf{1}$
for $i = d, d-1, \ldots, 2$ **do**
  for $\boldsymbol{\mathcal{Y}} \in \{\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}\}$ **do**
    $\boldsymbol{\Phi}_{\boldsymbol{\mathcal{Y}}} \leftarrow \text{EINSUM}\big(\texttt{"p r, r n r', r' m r'', r'' q -> p n m q"},$
    $\boldsymbol{\mathcal{L}}_{i-1}^{\boldsymbol{\mathcal{Y}}},\ \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{Y}},i-1},\ \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{Y}},i},\ \boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{Y}}}\big)$
  end for
  $\boldsymbol{\Phi} \leftarrow \boldsymbol{\Phi}_{\boldsymbol{\mathcal{A}}} * \boldsymbol{\Phi}_{\boldsymbol{\mathcal{B}}}$
  $\boldsymbol{\Phi} \leftarrow \text{REARRANGE}\big(\texttt{"p n m q -> (p n) (m q)"},\ \boldsymbol{\Phi}\big)$
  $(\boldsymbol{\mathcal{Z}},\ \text{pivot}) \leftarrow \text{PRRLDU}_{\text{col}}(\boldsymbol{\Phi};\ \epsilon,\ R)$
  $\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},i} \leftarrow \text{REARRANGE}\big(\texttt{"r' (m q) -> r' m q"},\ \boldsymbol{\mathcal{Z}}\big)$
  for $\boldsymbol{\mathcal{Y}} \in \{\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}\}$ **do**
    $\boldsymbol{\mathcal{R}}_{i-1}^{\boldsymbol{\mathcal{Y}}} \leftarrow \text{REARRANGE}\big(\text{EINSUM}\big(\texttt{"r m r', r' q -> r (m q)"},\ \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{Y}},i},\ \boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{Y}}}\big)\big)[:, \text{pivot}]$
  end for
  if $i = 2$ **then**
    $\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},1} \leftarrow \text{REARRANGE}\big(\texttt{"n r' -> 1 n r'"},\ \boldsymbol{\Phi}[:, \text{pivot}]\big)$
  end if
end for
return cores $\{\boldsymbol{\mathcal{T}}_{\tilde{\boldsymbol{\mathcal{X}}},i}\}_{i=1}^{d}$, updated $\{\boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{B}}}\}_{i=1}^{d}$

Algorithm 7 TCI for $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}$

Require: TTs $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}$; max rank R; tolerance ϵ; number of sweeps T.
Ensure: TT approximation $\tilde{\boldsymbol{\mathcal{X}}} \approx \boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}$.

$\boldsymbol{\mathcal{A}} \leftarrow \text{RIGHTORTHOGONALIZE}(\boldsymbol{\mathcal{A}})$; $\boldsymbol{\mathcal{B}} \leftarrow \text{RIGHTORTHOGONALIZE}(\boldsymbol{\mathcal{B}})$
$\{\boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{B}}}\} \leftarrow \text{MAXVOLRIGHTENVS}(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}; R)$ ▷ Algorithm 4
for $t = 1, \ldots, T$ **do**
  $(\text{cores},\ \{\boldsymbol{\mathcal{L}}_i^{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{L}}_i^{\boldsymbol{\mathcal{B}}}\}) \leftarrow \text{L2R-TCI}(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}, \{\boldsymbol{\mathcal{L}}\}, \{\boldsymbol{\mathcal{R}}\}; R, \epsilon)$ ▷ Algorithm 5
  $(\text{cores},\ \{\boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{B}}}\}) \leftarrow \text{R2L-TCI}(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}, \{\boldsymbol{\mathcal{L}}\}, \{\boldsymbol{\mathcal{R}}\}; R, \epsilon)$ ▷ Algorithm 6
end for
return TT $\tilde{\boldsymbol{\mathcal{X}}}$ with cores cores

I.4 RSI

Randomized Sketching with Interpolation (RSI) [38] replaces TCI's maxvol-based right environment construction with random sketching. Like HATT, RSI sketches each input TT independently rather than evaluating exact entries; like TCI, it then performs interpolative-decomposition sweeps to construct $\tilde{\boldsymbol{\mathcal{X}}}$. Compared to HATT, RSI differs in two ways: (i) it uses a Khatri-Rao product (KRP) sketch rather than a TT sketch, and (ii) it uses interpolative decompositions (prrLDU) rather than SVDs to truncate at each site. Because the sketching is per-factor, RSI side-steps the combinatorial maxvol search of TCI (which is specific to the Hadamard product) and immediately extends to any non-linear operation over multiple TTs.

The original RSI formulation uses a single L2R sweep after sketching. We extend it to alternating L2R/R2L sweeps, which follow naturally from the TCI connection: once a first L2R sweep produces interpolative left environments, subsequent sweeps refine the pivots in both directions exactly as in TCI.

KRP sketches. The original RSI paper uses a Khatri-Rao product sketch: a set of d matrices $\{\boldsymbol{\mathcal{S}}_i \in \mathbb{R}^{n_i \times s}\}_{i=1}^{d}$ with i.i.d. random entries (e.g., Gaussian or Rademacher), where $s = R + r_{\text{oversample}}$ is the sketch width. Unlike the TT sketch used in HATT— whose sketch cores have an internal bond structure $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{S}},k} \in \mathbb{R}^{s_{k-1} \times n_k \times s_k}$ — the KRP sketch has a single "flat" sketch axis s shared across all d sites. This makes the contractions in the right-env sweep structurally simpler.

Algorithm 8 KRP-sketched right environments for a TT $\boldsymbol{\mathcal{A}}$

Require: TT cores $\{\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}},i}\}_{i=1}^{d}$; KRP sketch matrices $\{\boldsymbol{\mathcal{S}}_i \in \mathbb{R}^{n_i \times s}\}_{i=1}^{d}$.
Ensure: Right environments $\{\boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{A}}}\}_{i=1}^{d-1}$.

$\boldsymbol{\mathcal{R}}_d^{\boldsymbol{\mathcal{A}}} \leftarrow \mathbf{1} \in \mathbb{R}^{1\times s}$ ▷ all-ones vector along the sketch axis
for $i = d, d-1, \ldots, 2$ **do**
    $\boldsymbol{\mathcal{R}}_{i-1}^{\boldsymbol{\mathcal{A}}} \leftarrow \text{EINSUM}\big(\texttt{"r n r', n s, r' s -> r s"}, \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{A}},i}, \boldsymbol{\mathcal{S}}_i, \boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{A}}}\big)$
end for
return $\{\boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{A}}}\}_{i=1}^{d-1}$

Algorithm 9 RSI for $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}$

Require: TTs $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}$; max rank R; tolerance ϵ; sketch width $s = R + r_{\text{oversample}}$; number of sweeps $T \geq 1$.
Ensure: TT approximation $\tilde{\boldsymbol{\mathcal{X}}} \approx \boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}$.

$\boldsymbol{\mathcal{A}} \leftarrow \text{RIGHTORTHOGONALIZE}(\boldsymbol{\mathcal{A}})$; $\boldsymbol{\mathcal{B}} \leftarrow \text{RIGHTORTHOGONALIZE}(\boldsymbol{\mathcal{B}})$
Draw KRP sketches $\{\boldsymbol{\mathcal{S}}_i \in \mathbb{R}^{n_i \times s}\}_{i=1}^{d}$ with i.i.d. random entries
$\{\boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{A}}}\} \leftarrow \text{KRP-RIGHTENVS}(\boldsymbol{\mathcal{A}}, \{\boldsymbol{\mathcal{S}}_i\})$ ▷ Algorithm 8
$\{\boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{B}}}\} \leftarrow \text{KRP-RIGHTENVS}(\boldsymbol{\mathcal{B}}, \{\boldsymbol{\mathcal{S}}_i\})$
for $t = 1, \ldots, T$ **do**
    (cores, $\{\boldsymbol{\mathcal{L}}_i^{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{L}}_i^{\boldsymbol{\mathcal{B}}}\}$) $\leftarrow$ L2R-TCI($\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}, \{\boldsymbol{\mathcal{L}}\}, \{\boldsymbol{\mathcal{R}}\}; R, \epsilon$) ▷ Algorithm 5
    (cores, $\{\boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{B}}}\}$) $\leftarrow$ R2L-TCI($\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}, \{\boldsymbol{\mathcal{L}}\}, \{\boldsymbol{\mathcal{R}}\}; R, \epsilon$) ▷ Algorithm 6; original RSI omits this and stops after the first L2R
end for
return TT $\tilde{\boldsymbol{\mathcal{X}}}$ with cores cores

Phase 1: Sketched right environments. A right-to-left sweep (Algorithm 8) computes the per-TT right environments $\boldsymbol{\mathcal{R}}_i^{\boldsymbol{\mathcal{Y}}} \in \mathbb{R}^{r_i^{Y} \times s}$ for $\boldsymbol{\mathcal{Y}} \in \{\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}\}$. At each site, the previous right environment is updated by contracting the core with the per-site sketch matrix on the physical dimension and broadcasting on the shared sketch axis.

Phase 2: Alternating sweeps. The L2R and R2L sweeps of TCI (Algorithms 5 and 6) are reused unchanged: the sketched right environments play the same role in the supercore contractions as TCI's maxvol-selected ones. After the first L2R sweep, the left environments become interpolative (subset-based) just as in TCI; from then on, the algorithm is identical to TCI. The original RSI corresponds to stopping after this first L2R sweep, before any R2L.

Complexity. Forming the KRP-sketched right environments costs $\mathcal{O}(dnr^2 s)$ per TT, dominated by the per-site einsum. The subsequent TCI sweeps inherit the per-sweep complexity of $\mathcal{O}(dn^2 r^3)$ in time and $\mathcal{O}(n^2 r^2)$ in memory. With $s \approx r$ and a constant number of sweeps, the overall cost matches TCI: $\mathcal{O}(dn^2 r^3)$ time, $\mathcal{O}(n^2 r^2)$ memory. The savings over TCI come not from asymptotic complexity but from avoiding the combinatorial maxvol search in the warm-up; the savings over HATT come from per-site sketches replacing TT sketches in the right-env sweep.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The contributions are listed explicitly at the end of the Introduction. The framework and tensor-train methods are developed in Sections 2 to 4, and the inference and learning claims are evaluated in Section 5. In particular, Section 5.3 distinguishes tasks where independence reduction remains competitive from tasks with shared auxiliary variables, where preserving dependencies through compilation substantially improves learning.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: A dedicated "Limitations" paragraph follows the Conclusions and discusses the suitability of current recompression methods for sparse Boolean tensors as well as the need to study the effect of program factorization and auxiliary-variable structure across more neurosymbolic methods.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Theoretical results (e.g. Theorem C.1 and the corresponding corollary) are stated with all assumptions and proved in Section C; supporting derivations for the independence approximation are given in Section E.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We describe the framework and pipeline choices in Section 4; provide tensor-train operations, multiplication schedules, and recompression algorithms in Sections F, F.2, G and I; define the six learning tasks and their factorizations in Section H.3; and report compilation settings, compute budgets, learning objectives, decoding procedures, and hyperparameters in Section H.5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code and instructions for reproducing the experiments are available through the public repository linked in the paper. Dependencies are pinned and reproducibly installable with the `uv` package manager.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5 describes the benchmarks, baselines, metrics, and main experimental setup. Sections H.3 to H.5 specify the task construction and splits, weighted-inference protocol, compilation configurations, optimizer, learning rate, batch size, number of epochs, and decoding procedure.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: Results are averaged over ten runs with fixed datasets across methods and runs, capturing variations across training and compute randomness. Figures report the mean and standard-deviation error bars across runs.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section H.5 specifies the NVIDIA A10G GPU and its memory, the CPU memory cap for SDD compilation, and the compilation and training wall-clock limits. Compilation times are reported in the inference figures, and runs exceeding the stated time or memory budgets are marked as timed out or out of memory.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and the research conforms to it; the work involves no human subjects, no personal data, and does not pose ethical concerns.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: The paper is foundational research on tractable inference primitives for neurosymbolic systems, with no direct path to positive or negative societal applications beyond the general impact of more reliable probabilistic reasoning tools.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper releases no pre-trained generative models, image generators, or scraped datasets with risk of misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite the original works for all datasets, baselines, and external libraries used (e.g., ViSudo datasets, SatNet, KaHyPar, NetworkX), and use them under their standard research licenses.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: The paper does not release new datasets or pre-trained models. The code is research code provided with reproducibility instructions but does not consist in an asset.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper involves no crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper involves no human subjects research.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: We include a voluntary AI use statement in the acknowledgements. LLMs are not part of the core methodology of this paper. Use of generative AI tools was limited to help in coding some experiments, not the model, and was reviewed by the authors.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.