PSoufflé: Scaling Exact Probabilistic Logic Inference for Program Analysis

Xuyang Li*
Purdue University
West Lafayette, IN, USA
li5274@purdue.edu

Jiahao Xia*
University of Maryland
College Park, MD, USA
jxia77@terpmail.umd.edu

Ahmed Adnan
East West University
Dhaka, Bangladesh
bsse1131@iit.du.ac.bd

Jingbo Wang
Purdue University
West Lafayette, IN, USA
wang6203@purdue.edu

Artifact Reviewed
10.5281/zenodo.20091940

***Abstract*—Probabilistic extensions of logic programming languages, such as ProbLog, allow users to annotate facts and rules with probabilities and have been useful in domains such as quantitative program analysis. However, exact inference for such programs often fails to scale to program-analysis workloads, where large derivation graphs and many probabilistic variables make decision-diagram construction prohibitively expensive. We present PSoufflé, an exact and scalable inference toolchain built on Soufflé for probabilistic Datalog programs. Its key contribution is a semantics-preserving rewriting of Datalog derivation graphs: by exploiting conditional independence in local derivation structures, PSoufflé replaces these structures with smaller equivalent representations, thereby reducing the number of probabilistic variables exposed to decision-diagram construction. We evaluate PSoufflé on 50 benchmarks from power side-channel analysis, taint analysis, and disassembly analysis. PSoufflé completes every benchmark within the 30-minute timeout, while three state-of-the-art baseline solvers, ProbLog2, vProbLog, and Scallop, complete only 12%, 54%, and 32%, respectively. On benchmarks completed by all tools, PSoufflé achieves average speedups of 219× over ProbLog2, 119× over vProbLog, and 18× over Scallop.**

I. Introduction

Probabilistic logic programming extends rule-based reasoning with uncertainty by associating probabilities with facts, rules, or choices, allowing query answers to be interpreted quantitatively rather than as only true or false. This family includes systems such as ProbLog [1]–[3], which extends Prolog with probabilistic reasoning, as well as probabilistic variants of Datalog [4]–[6] that combine Datalog's finite, fixed-point evaluation model with probabilistic semantics. These languages are useful for applications where uncertainty is inherent, including ranking [7], [8], risk assessment [9], and quantitative program analysis [4]–[6], [10], [11]. In particular, probabilistic Datalog provides a natural foundation for quantitative program analyses: Datalog rules express recursive relational reasoning over program facts, while probabilities allow analyses to rank alarms, estimate risk, and quantify the likelihood of derived properties.

Existing inference tools for probabilistic logic programming face a trade-off between exactness and scalability. Exact engines [2], [3], such as ProbLog, return accurate probabilities but often fail to scale to large quantitative program-analysis workloads. A key bottleneck is knowledge compilation: large derivation graphs induce provenance formulas with many correlated probabilistic variables, making decision-diagram (DD) construction prohibitively expensive. Approximate systems, such as Bingo [4], NESA [5], and Praline [6], improve scalability by relaxing exact inference, but they do not always return exact probabilities and may require additional approximation choices or user guidance. Thus, existing tools either preserve exactness but struggle with large correlated derivation structures, or scale by sacrificing exact guarantees.

Motivated by this trade-off, we present PSoufflé, an *exact* and *scalable* inference engine built on Soufflé [12] for probabilistic Datalog programs arising from large-scale program analyses. The key insight of PSoufflé is to reduce the derivation graph before DD construction. Rather than exposing a large monolithic provenance formula to knowledge compilation, PSoufflé identifies local subgraphs whose internal derivations are conditionally independent of the rest of the graph given their interface atoms. It then applies semantics-preserving rewrites that replace these subgraphs with compact probabilistic representations. These rewrites localize correlations, prune internal derivations, and substantially reduce the number of probabilistic variables exposed to DD construction, making exact inference practical on large-scale analyses.

We evaluated PSoufflé on 50 probabilistic Datalog benchmarks derived from side-channel vulnerability detection [6], [13], taint analysis [4], and disassembly analysis [14]. Across benchmarks completed by both PSoufflé and the corresponding baseline, PSoufflé achieves average speedups of 219×, 119×, and 18× over three prior exact inference baselines. To summarize, this paper makes the following contributions:

- We present PSoufflé, an exact inference toolchain for probabilistic Datalog programs that scales to large program-analysis workloads.
- We introduce semantics-preserving rewrites of derivation graphs that exploit locally visible conditional independence to replace local probabilistic structures with compact, equivalent representations before decision-diagram construction.
- We demonstrate that PSoufflé substantially improves the scalability of exact probabilistic inference on three representative program-analysis workloads.

*Equal contribution.

https://doi.org/10.34727/2026/isbn.978-3-85448-093-8_42

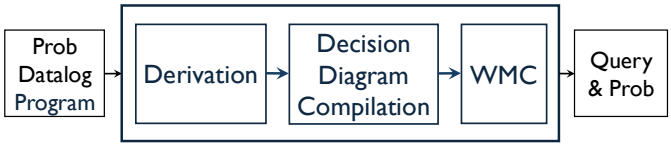


Fig. 1: Overview of probabilistic Datalog solving via derivation-graph construction, decision-diagram compilation, and weighted model counting (WMC).

II. Background on Probabilistic Datalog

A *probabilistic Datalog program* is a pair $P = \langle \mathcal{R}, \mathcal{Q} \rangle$, where $\mathcal{R}$ is a finite set of probabilistic rules and $\mathcal{Q}$ is a finite set of ground query atoms. Each rule $r \in \mathcal{R}$ has the form

$$p :: H(\mathbf{x}) :- \odot_1 B_1(\mathbf{y}_1), \ldots, \odot_n B_n(\mathbf{y}_n), \tag{1}$$

where $p \in (0, 1]$ is the rule probability and $\odot_i \in \{\cdot, \neg\}$ indicates whether B_i appears positively or negated. A rule with an empty body is a probabilistic input fact. We assume that programs are stratified and that all ground rule-instance events, including probabilistic input facts, are mutually independent.

Semantics. Grounding $\mathcal{R}$ yields a finite set of ground rule instances. Each instance is associated with a Boolean event variable $X \in \mathcal{X}$, and the weight map $W : \mathcal{X} \to (0, 1]$ assigns X the probability annotation of the corresponding rule, such as p in Eq. (1). We extend W to literals in the standard way: for $l = X$, $W(l) = W(X)$, and for $l = \neg X$, $W(l) = 1 - W(X)$. A possible world ω is a complete set of literals over $\mathcal{X}$, selecting which ground rule instances fire. Under the independence assumption, $\mathsf{Pr}(\omega) = \prod_{l \in \omega} W(l)$. Let $\mathsf{lfp}(\omega)$ be the least Herbrand model of the deterministic Datalog program induced by ω. A query $q \in \mathcal{Q}$ succeeds in ω, written $\omega \models q$, iff $q \in \mathsf{lfp}(\omega)$. Thus,

$$\mathsf{Pr}(q) = \sum_{\omega \models q} \mathsf{Pr}(\omega)\,.$$

As shown in Figure 1, exact inference for probabilistic Datalog computes this probability by constructing a derivation graph, compiling the query's derivability condition into a decision diagram, and evaluating the resulting formula by weighted model counting.

Definition 1 (Derivation graph)**.** *A* derivation graph *is a directed edge-labeled hypergraph $\mathcal{G} = (N, \mathcal{E}, \mathcal{L}, W)$, where each node $n \in N$ is a ground atom and each hyperedge $e \in \mathcal{E}$ has the form*

$$(B_e^+, B_e^-) \to H_e.$$

Here, B_e^+ and B_e^- are the positive and negative body atoms of the corresponding ground rule instance, and H_e is its head atom. We write $\mathsf{head}(e) = H_e$ and $\mathsf{body}(e) = B_e^+ \cup B_e^-$. The labeling function $\mathcal{L} : \mathcal{E} \to \mathcal{X}$ maps each hyperedge to its event variable, and $W : \mathcal{X} \to (0, 1]$ assigns event probabilities.

Definition 2 (Boolean objective formula)**.** *Given an edge set $E \subseteq \mathcal{E}$, the Boolean objective formula $\phi_E(n)$ characterizes when a node n is derivable using only edges in E:*

$$\phi_E(n) = \bigvee_{\substack{e \in E \\ \mathsf{head}(e) = n}} \left(\mathcal{L}(e) \wedge \bigwedge_{a \in B_e^+} \phi_E(a) \wedge \bigwedge_{b \in B_e^-} \neg \phi_E(b) \right).$$

For an empty-body hyperedge, the disjunct is simply $\mathcal{L}(e)$. Cycles in $\mathcal{G}$ are handled by standard ProbLog-style compilation, so objective formulas are expressed over independent event variables in $\mathcal{X}$.

Decision diagrams and weighted model counting. Given $\phi_E(n)$, exact inference compiles it into a binary DD $D(n)$ under a fixed variable order and computes its probability by weighted model counting.

Definition 3 (Weighted model counting)**.** *Given a Boolean formula ϕ over event variables $\mathcal{X}$ and a weight map W, its weighted model count is*

$$\mathsf{WMC}(\phi, W) = \sum_{\omega \models \phi} \prod_{l \in \omega} W(l),$$

where each satisfying assignment ω is viewed as the complete set of literals it makes true.

Therefore, for a query atom q with corresponding derivation-graph node n_q, its success probability is

$$\mathsf{Pr}(q) = \mathsf{WMC}(\phi_{\mathcal{E}}(n_q), W).$$

III. Rewrite-Driven Inference via SISO Decomposition

The main bottleneck of exact inference for *probabilistic Datalog* is often global DD construction, whose cost can grow exponentially in the number of probabilistic events. To reduce this cost, we prune the derivation graph before global compilation, thereby exposing fewer probabilistic events to the DD. Our pruning method identifies local subgraphs that can be safely rewritten without changing query probabilities.

The central pattern is a *single-input single-output* (`SISO`) region. A `SISO` region has an entry atom n_{in} and an exit atom n_{out}: the outside graph may enter the region only through n_{in} and may observe the region only through n_{out}. All other derivations and event variables are internal to the region. Thus, conditioned on n_{in} being true, the internal derivations are independent of the rest of the graph and affect it only through whether n_{out} is derived. We exploit this conditional independence by replacing the region with one fresh probabilistic edge $n_{\mathsf{in}} \to n_{\mathsf{out}}$, whose weight is the exact local probability of deriving n_{out} from n_{in}.

Figure 2 illustrates the rewrite. For ease of presentation, suppose that all rule edges are deterministic and that only the input facts I_i are probabilistic; we also write I_i for the event variable of the corresponding input fact. In the figure, each $\wedge$ node represents a single hyperedge whose body atoms are conjunctively required, while multiple incoming alternative

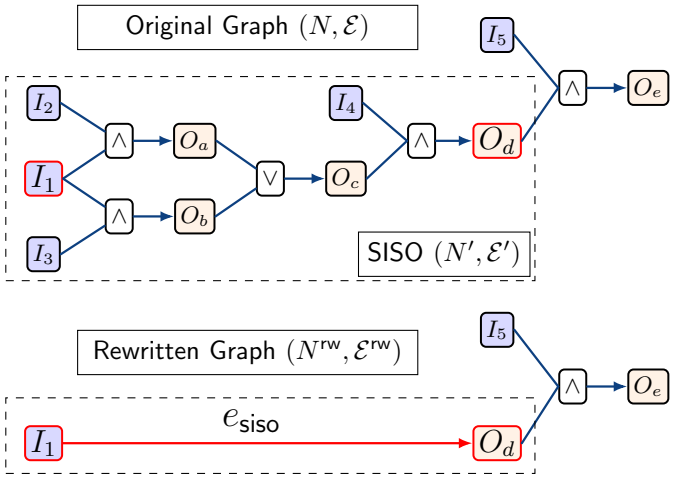


Fig. 2: A SISO summarization rewrite: the dashed region is replaced by a fresh red edge from entry I_1 to exit O_d. Blue and orange nodes denote input and output facts, respectively.

hyperedges are disjoined, corresponding to $\vee$. Thus, in the original graph,

$$\phi_{\mathcal{E}}(O_e) \equiv (I_1 \wedge I_2 \wedge I_4 \wedge I_5) \vee (I_1 \wedge I_3 \wedge I_4 \wedge I_5).$$

The dashed region has entry I_1 and exit O_d, both framed in *red*. Conditioned on I_1 being true, the local formula for deriving O_d is

$$\phi^{I_1}_{\mathcal{E}'}(O_d) \equiv I_4 \wedge (I_2 \vee I_3).$$

The fresh edge $e_{\mathsf{siso}} : I_1 \to O_d$ is assigned the exact local probability

$$p_{\mathsf{siso}} = \mathsf{WMC}\big(\phi^{I_1}_{\mathcal{E}}(O_d), W\big).$$

After rewriting, the edge set becomes $\mathcal{E}^{\mathsf{rw}}$ and the query formula becomes

$$\phi_{\mathcal{E}^{\mathsf{rw}}}(O_e) \equiv \underline{O_d} \wedge I_5 \equiv \underline{I_1 \wedge \mathcal{L}(e_{\mathsf{siso}})} \wedge I_5,$$

where $\mathcal{L}(e_{\mathsf{siso}})$ is a fresh event variable with weight p_{siso}.

A. SISO Regions

Definition 4 (SISO region)**.** *Let $\mathcal{G} = (N, \mathcal{E}, \mathcal{L}, W)$ be a derivation graph and let $G' = (N', \mathcal{E}')$ be a sub-hypergraph with $N' \subseteq N$ and $\mathcal{E}' \subseteq \mathcal{E}$. We say that G' is a* single-input single-output *(SISO) region with interface $(n_{\mathsf{in}}, n_{\mathsf{out}}) \in N' \times N'$ if:*

1) ***Single entry.*** *The only node in N' that may depend on derivations outside the region is n_{in}, i.e., for every $e \in \mathcal{E}$, $\mathsf{head}(e) \in N' \wedge \mathsf{body}(e) \not\subseteq N' \implies \mathsf{head}(e) = n_{\mathsf{in}}$.*
2) ***Single exit.*** *The only node in N' that may be consumed outside the region is n_{out}. Formally, for every $e \in \mathcal{E} \setminus \mathcal{E}'$, $\mathsf{body}(e) \cap N' \neq \emptyset \implies \mathsf{body}(e) \cap N' \subseteq \{n_{\mathsf{out}}\}$.*
3) ***Internal encapsulation.*** *All derivations of non-entry nodes in the region are internal. That is, for every $e \in \mathcal{E}$, $\mathsf{head}(e) \in N' \setminus \{n_{\mathsf{in}}\} \implies e \in \mathcal{E}' \wedge \mathsf{body}(e) \subseteq N'$.*
4) ***Internal relevance.*** *Every edge in $\mathcal{E}'$ lies on a derivation path from n_{in} to n_{out} inside G'.*

Algorithm 1 REWRITESISO$(\mathcal{G}, q)$

```
G^rw ← G                                   ▷ G^rw = (N^rw, E^rw, L^rw, W^rw)
while a valid SISO region exists in G^rw do
    (G', n_in, n_out) ← DETECTSISO(G^rw, q)
    p ← SOLVEREGION(G', n_in, n_out, W^rw)
    Replace G' by a fresh edge e_new : n_in → n_out with
    W^rw(L^rw(e_new)) = p
end while
return G^rw
```

The first two conditions specify the interface of the region: external derivations can enter only through n_{in}, and external uses can observe the region only through n_{out}. Internal encapsulation makes the derivation of every non-entry atom depend only on edges in $\mathcal{E}'$, once n_{in} is fixed. Internal relevance ensures that every local derivation of n_{out} goes through n_{in}, so $\Pr(n_{\mathsf{out}} \mid \neg n_{\mathsf{in}}) = 0$. Hence the region is fully characterized by $\Pr(n_{\mathsf{out}} \mid n_{\mathsf{in}})$ and can be rewritten as one edge $n_{\mathsf{in}} \to n_{\mathsf{out}}$.

B. Rewriting SISO Regions

Given a SISO region $G' = (N', \mathcal{E}')$ with interface $(n_{\mathsf{in}}, n_{\mathsf{out}})$, we compute the exact probability that n_{out} is derived from n_{in} using only the local derivations in $\mathcal{E}'$. Let $\phi^{n_{\mathsf{in}}}_{\mathcal{E}'}(n_{\mathsf{out}})$ denote the objective formula from Definition 2, restricted to $\mathcal{E}'$, with the entry atom treated as already established, i.e., $\phi^{n_{\mathsf{in}}}_{\mathcal{E}'}(n_{\mathsf{in}}) = \mathsf{true}$.

Definition 5 (SISO summarization rewrite)**.** *Let $G' = (N', \mathcal{E}')$ be a SISO region with interface $(n_{\mathsf{in}}, n_{\mathsf{out}})$. Define*

$$p = \mathsf{WMC}(\phi^{n_{\mathsf{in}}}_{\mathcal{E}'}(n_{\mathsf{out}}), W)\,.$$

The SISO *summarization rewrite removes the internal nodes $N' \setminus \{n_{\mathsf{in}}, n_{\mathsf{out}}\}$ and the edges $\mathcal{E}'$, introduces a fresh event variable $X_{\mathsf{new}} \notin \mathcal{X}$ with $W(X_{\mathsf{new}}) = p$, and adds a new hyperedge $e_{\mathsf{new}} : n_{\mathsf{in}} \to n_{\mathsf{out}}$ labeled by $\mathcal{L}(e_{\mathsf{new}}) = X_{\mathsf{new}}$.*

Algorithm 1 gives the rewrite-driven procedure. It repeatedly detects a valid SISO region in the current graph, computes its exact local probability, and replaces the region by a fresh probabilistic edge. When no valid region remains, the resulting graph $\mathcal{G}^{\mathsf{rw}}$ is compiled into a decision diagram and solved by weighted model counting.

SOLVEREGION. For a SISO region $G' = (N', \mathcal{E}')$ with interface $(n_{\mathsf{in}}, n_{\mathsf{out}})$, SOLVEREGION computes the probability p in Definition 5. It uses either pattern-based computation or local DD compilation.

(I). Pattern-based computation. For common local structures, p can be computed directly from edge weights, without constructing a decision diagram. Figure 3 shows representative patterns, with blue arrows denoting rewrite transitions. Linear chains and conjunctive diamonds are summarized by multiplying the encapsulated edge probabilities. Absorbing-fact patterns fold internal probabilistic facts into the replacement edge. Parallel derivations are summarized by a union probability; for independent alternatives with probabilities $p_1, \ldots, p_k$, the replacement weight is $1 - \prod_{i=1}^{k}(1 - p_i)$.

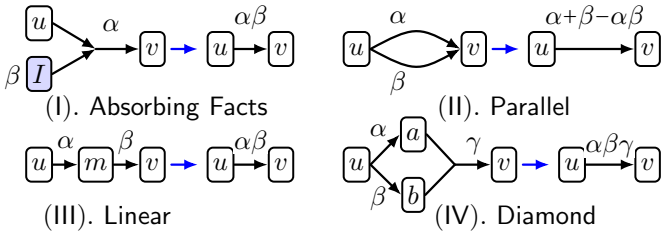


Fig. 3: Representative rewriting patterns.

The patterns in Figure 3 are not sufficient by themselves to justify rewriting. A subgraph matching one of these syntactic patterns can be rewritten only if it satisfies the SISO conditions in Definition 4. These conditions are critical for establishing the correctness theorem: they provide the assumptions on the derivation graph used in the proof of Theorem 1.

In particular, they ensure that the events eliminated by the rewrite are internal to the region, that external derivations can enter the region only through n_{in}, and that the rest of the derivation graph can observe the region only through n_{out}. Under these assumptions, each replacement edge instantiates the SISO summarization rewrite in Definition 5. Theorem 1 then proves that such a rewrite is semantics preserving and yields the same query probability.

(II). Local DD compilation. For a SISO region that does not match a structural pattern, we compile the local formula $\phi^{n_{\mathsf{in}}}_{\mathcal{E}'}(n_{\mathsf{out}})$ into a DD and compute its weighted model count. This compilation involves only the event variables inside the region, and is therefore often much smaller than compiling the global query formula. Each rewrite also removes the region's internal event variables from the global query formula.

Theorem 1 (Correctness of a SISO rewrite)**.** *Let $\mathcal{G} = (N, \mathcal{E}, \mathcal{L}, W)$ be a derivation graph, let q be a query node, and let $\mathcal{G}^{\mathsf{rw}} = (N^{\mathsf{rw}}, \mathcal{E}^{\mathsf{rw}}, \mathcal{L}^{\mathsf{rw}}, W^{\mathsf{rw}})$ be obtained from $\mathcal{G}$ by one SISO summarization rewrite. If q remains in $\mathcal{G}^{\mathsf{rw}}$, then*

$$\mathsf{WMC}(\phi_{\mathcal{E}}(q), W) = \mathsf{WMC}(\phi_{\mathcal{E}^{\mathsf{rw}}}(q), W^{\mathsf{rw}}).$$

Corollary 1 (Correctness of a sequence of rewrites)**.** *Any finite sequence of SISO rewrites preserves $\mathsf{WMC}(\phi_{\mathcal{E}}(q), W)$ for every query node q that remains in the graph.*

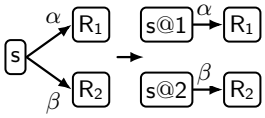
C. Detecting SISO Regions

Although rewriting reduces the number of probabilistic variables exposed to decision diagram compilation, detecting and rewriting SISO regions incurs overhead, especially when a region must be solved by local DD compilation. Therefore, it is important to select regions that provide sufficient reduction to justify this cost. The **DETECTSISO** procedure searches the backward slice of query q for valid SISO regions and prioritizes profitable ones for rewriting. It terminates when no valid SISO region remains or a given budget is exhausted.

Candidate selection. We visit candidate output nodes n_{out} in reverse topological order among nodes that can affect q, prioritizing regions closer to the query. For each candidate n_{out}, we grow a region backward along incoming derivations. Expansion stops if adding a node or edge would introduce another entry, expose an internal node to the outside, or violate the SISO conditions in Definition 4. The grown subgraph is accepted only if it has a unique entry n_{in}, a unique exit n_{out}, and satisfies Definition 4.

Optimization (I): Scoring. When multiple candidates are valid, we score regions by the expected reduction in global DD size, while penalizing large local regions. This heuristic affects runtime but not correctness. Regardless of which valid SISO regions are chosen, the rewritten graph preserves the same query probability by Theorem 1 and Corollary 1. Runtime, however, does depend on the scoring strategy. Although larger or maximal SISO regions may eliminate more probabilistic variables, they also increase detection and local solving overhead. We therefore bias the search toward compact regions that remove many probabilistic variables without making **SOLVEREGION** unnecessarily expensive.

Optimization (II): Splitting. Some rewrite opportunities are obscured by shared nodes. For example, a node s may feed multiple downstream derivations, such as R_1 and R_2. Splitting such a node blindly can introduce redundant aliases, increase graph size, and hurt performance. Therefore, PSoufflé performs splitting selectively. For each shared node (e.g., s), it computes the forward reachable sets of its outgoing branches and splits only branches whose downstream cones are disjoint. These are precisely the cases where separating the uses of the shared node can expose multiple independent SISO regions, thereby enabling additional rewrites and improving performance.



IV. EVALUATION

Implementation. Our proposed tool PSoufflé is implemented on top of Soufflé [12] and uses CUDD [15] as its underlying DD library with dynamic variable reordering enabled.

Our evaluation answers the following three questions: **(RQ1)** How effective is PSoufflé at reducing derivation-graph size? **(RQ2)** How does PSoufflé compare with existing exact probabilistic Datalog tools? **(RQ3)** How much does rewriting contribute to end-to-end runtime within PSoufflé?

Benchmarks. We evaluate PSoufflé on three probabilistic Datalog suites from program-analysis workloads.

SC This suite is derived from side-channel detection benchmarks used in prior probabilistic Datalog work [6], [13]. It analyzes cryptographic implementations such as AES [16], SHA-3, and MAC-Keccak [17], [18], and contains 19 instances.

TA This suite is derived from the Android taint-analysis workload of Bingo [4], originally implemented on top of Soot [19]. We use sliced instances whose queries ask for the probability that a tainted source-to-sink path exists. The suite contains 15 instances.

BS This suite is based on DDISASM [14] for binary symbolization analysis. It identifies candidate data objects in stripped binaries and derives symbolic data references and

labeled addresses, with uncertain facts annotated probabilistically. The suite contains 16 real-world binaries.

Benchmark Statistics. Table I summarizes the sizes of the resulting derivation graphs, where nodes are ground facts and hyperedges are grounded rule applications. As shown in Table I, the TA and BS suites contain hundreds of thousands of nodes and hyperedges, highlighting the scale of exact probabilistic inference required by program analysis workloads.

TABLE I: Derivation graph sizes for the SC, TA, and BS benchmark suites. #nodes denotes the number of ground facts, and #edges denotes the number of hyperedges corresponding to ground rule applications.

	Average		**Max**		**Median**	
	#nodes	#edges	#nodes	#edges	#nodes	#edges
SC [6], [13]	54K	25K	227K	97K	8K	8K
TA [4]	305K	156K	727K	396K	261K	117K
BS [14]	376K	331K	919K	776K	249K	200K

Baselines. We compare PSoufflé with three state of the art baselines for exact probabilistic inference: ProbLog2 [20], vProbLog [3], and Scallop [21]. ProbLog2 and vProbLog are exact inference engines. For Scallop, we set its proof retention limit k high enough to retain all proofs for each instance, which yields exact probabilities. Other probabilistic Datalog solvers, such as Praline [6], Bingo [4], and Beer [22], target approximate inference and are therefore not direct baselines.

Settings. All experiments were run on a machine with an Intel(R) Core(TM) i5-8500T CPU and 52GB of memory, with a timeout of 1800 seconds per instance. Reported runtimes are averaged over five runs.

RQ1: Effect of rewriting on graph size: Table II reports the reduction rate of derivation-graph nodes and hyperedges after rewriting, computed as $1-|G_{\mathsf{after}}|/|G_{\mathsf{before}}|$. Rewriting reduces graph size across all three suites, with especially large reductions on SC and TA. Although the raw-size reduction is smaller for BS, rewriting still simplifies the remaining provenance structure, making subsequent DD compilation more efficient.

TABLE II: Derivation-graph reduction after rewriting.

	Average		**Max**		**Median**	
	#nodes	#edges	#nodes	#edges	#nodes	#edges
SC [6], [13]	0.67	0.60	1.00	1.00	0.73	0.70
TA [4]	0.61	0.59	0.90	0.90	0.56	0.55
BS [14]	0.20	0.12	0.29	0.22	0.22	0.14

RQ2: Baseline Comparison: Table III compares the end-to-end runtime of PSoufflé with ProbLog2 [20], vProbLog [3], and Scallop [21]. ProbLog2 times out on most large instances. vProbLog and Scallop are more scalable due to optimized fixed-point grounding, but still fail on the largest **BS** benchmarks.

All baselines expose large, monolithic provenance formulas to knowledge compilation, forcing the DD backend to reason about many correlated random variables simultaneously. In contrast, PSoufflé rewrites local derivation structures before global DD construction, reducing the number of probabilistic events and achieving substantially better scalability.

TABLE III: End-to-end inference time in seconds. Ours denotes PSoufflé. The fastest runtime in each benchmark is bold. P, vP, and Sp denote ProbLog2, vProbLog, and Scallop, respectively. T/O denotes a timeout after 1800 seconds.

Side-channel Analysis (**SC**)									
	Ours	P	vP	Sp		Ours	P	vP	Sp
S1	**0.4**	T/O	14.9	T/O	S2	**0.04**	1.6	5.1	0.3
S3	**0.05**	2.6	5.2	0.3	S4	**0.07**	4.9	6.1	0.3
S5	**0.08**	29.5	7.6	0.3	S6	**0.1**	48.2	19.3	0.4
S7	**0.1**	34.7	12.9	0.4	S8	**0.4**	T/O	170.5	2.1
S9	**0.4**	T/O	167.6	2.1	S10	**0.5**	T/O	151.0	8.3
S11	**1.1**	T/O	421.9	32.1	S12	**1.9**	T/O	846.8	75.5
S13	**3.3**	T/O	T/O	305.4	S14	**7.7**	T/O	T/O	T/O
S15	**11.1**	T/O	T/O	T/O	S16	**14.7**	T/O	T/O	T/O
S17	**18.1**	T/O	T/O	T/O	S18	**19.6**	T/O	T/O	T/O
S19	**20.4**	T/O	T/O	T/O					
Taint Analysis (**TA**)									
	Ours	P	vP	Sp		Ours	P	vP	Sp
T1	**18.2**	T/O	766.4	T/O	T2	**17.7**	T/O	1036.3	T/O
T3	**18.4**	T/O	241.9	T/O	T4	**19.7**	T/O	204.7	361.4
T5	**48.3**	T/O	136.6	T/O	T6	**16.1**	T/O	1088.3	T/O
T7	**41.4**	T/O	131.1	T/O	T8	**23.4**	T/O	120.6	T/O
T9	**17.7**	T/O	660.9	196.9	T10	**67.5**	T/O	203.5	T/O
T11	**24.2**	T/O	85.6	T/O	T12	**24.3**	T/O	144.9	T/O
T13	**38.0**	T/O	795.9	T/O	T14	**45.6**	T/O	1091.6	T/O
T15	**42.1**	T/O	173.8	T/O					
Binary Symbolization (**BS**)									
	Ours	P	vP	Sp		Ours	P	vP	Sp
B1	**19.5**	T/O	T/O	T/O	B2	**19.9**	T/O	T/O	326.5
B3	**121.1**	T/O	T/O	T/O	B4	**18.3**	T/O	T/O	T/O
B5	**103.8**	T/O	T/O	T/O	B6	**12.0**	T/O	T/O	T/O
B7	**101.8**	T/O	T/O	T/O	B8	**17.6**	T/O	T/O	333.7
B9	**26.4**	T/O	T/O	T/O	B10	**20.2**	T/O	T/O	T/O
B11	**40.4**	T/O	T/O	T/O	B12	**14.3**	T/O	T/O	T/O
B13	**16.9**	T/O	T/O	T/O	B14	**73.8**	T/O	T/O	T/O
B15	**196.0**	T/O	T/O	T/O	B16	**32.3**	T/O	T/O	T/O

RQ3: Ablation Study: Figure 4 compares PSoufflé, denoted by **R**, against its ablated variant without rewriting, denoted by **NR**. Since both configurations use the same Soufflé-optimized evaluator, their difference isolates the impact of rewriting on exact probability computation. The x-axis reports the number of random variables as a proxy for input complexity, and the y-axis reports end-to-end runtime.

Figure 4 shows that **R** consistently outperforms **NR**. In the **BS** suite, the **R** curve appears highest for instances with more than 25K random variables only because **NR** times out on those instances and therefore has no corresponding data points. These results show that rewriting substantially reduces end-to-end runtime, especially on larger instances where DD construction dominates. The speedup is driven by the graph reductions reported in RQ1, as shown in Table II: by pruning the derivation graph, PSoufflé reduces the number of probabilistic events exposed to exact inference.

Such gains rely on exploitable conditional independence, which sparse program-dependence structures often expose through SISO regions. On highly connected graphs with few such regions, rewriting offers limited reduction, leaving

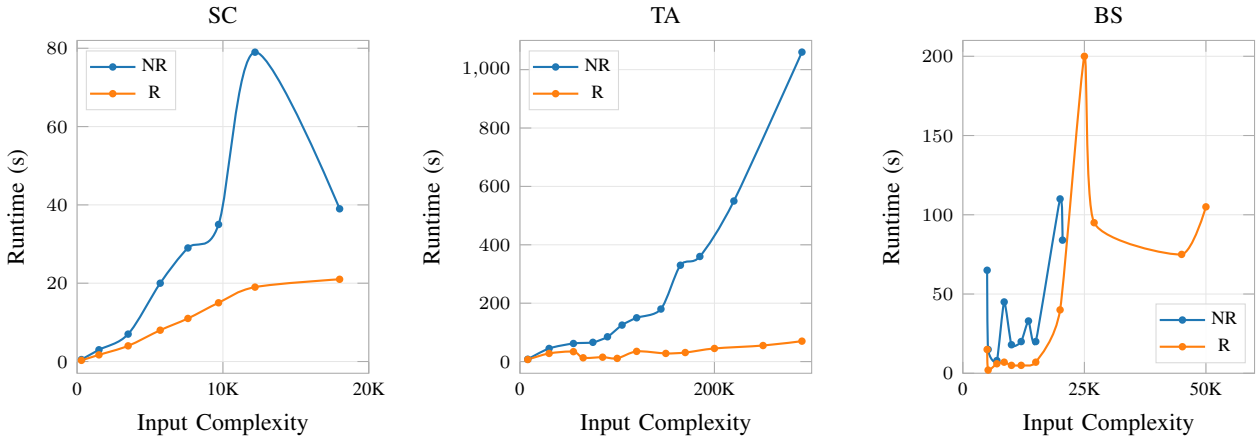


Fig. 4: End-to-end runtime of PSoufflé with rewriting (**R**) and without rewriting (**NR**) across the SC, TA, and BS suites. Among instances completed by both configurations, **R** is consistently faster; missing **NR** points indicate timeouts.

exact inference subject to the worst-case complexity of WMC; approximate inference may then be needed.

V. Related Work

Datalog has long served as a foundation for program analyses, including data-race detection [23], side-channel analysis [24], [25], and taint analysis [26], [27]. Probabilistic logic-programming and Datalog-style systems, such as ProbLog [1], [2], vProbLog [3], PPDL [28], Scallop [21], Bingo [4], Praline [6], and NESA [5], extend rule-based reasoning with uncertainty, enabling quantitative analyses that report the likelihood of alarms rather than only binary answers.

Existing approaches largely fall into two categories. Exact engines, including ProbLog [1], [2] and vProbLog [3], reduce probabilistic reasoning to weighted model counting over decision diagrams and return exact probabilities [1], [2], [20], [29]–[31]. However, they often fail to scale to large program-analysis workloads because monolithic provenance formulas expose many correlated probabilistic variables to knowledge compilation. Approximate techniques improve scalability by relaxing exactness [4]–[6], [10], [32], [33]; for example, Bingo-style methods use belief propagation, while Praline [6] uses interval arithmetic. These methods are scalable for large analyses, but they do not always provide exact probabilities.

PSoufflé bridges this gap by making exact inference scalable for probabilistic Datalog programs. Unlike existing exact engines that directly compile large monolithic provenance formulas, PSoufflé rewrites derivation graphs before decision-diagram construction, reducing the number of probabilistic variables exposed to knowledge compilation while preserving exact query probabilities. Built on Soufflé [12], PSoufflé also benefits from semi-naïve evaluation and Datalog-level optimizations [34]–[36]; however, its key scalability gain comes from derivation-graph rewriting. Although graph reduction has been studied in other areas, such as discrete-time Markov chains [37], those techniques focus on aggregating mutually exclusive paths; our rewrites handle overlapping Datalog derivations that may share probabilistic facts.

A complementary line of work studies incremental probabilistic Datalog inference, which reuses derivation and compilation results across program updates [38]. This direction is orthogonal to PSoufflé's rewrite-based optimization: it exploits reuse across runs, whereas PSoufflé reduces the derivation graph before decision-diagram construction within a single inference run.

VI. Conclusion

We presented PSoufflé, an exact inference toolchain for probabilistic Datalog programs that scales to large program-analysis workloads. The key idea is to reduce derivation graphs before decision-diagram construction by applying semantics-preserving rewrites to local `SISO` structures. Our evaluation on side-channel analysis, taint analysis, and disassembly analysis benchmarks shows that PSoufflé substantially improves scalability over existing exact probabilistic logic programming engines while preserving exact probability results.

Data Availability

The artifact(s) associated with this paper is/are available at Zenodo under the following DOI: 10.5281/zenodo.20091940.

References

[1] L. De Raedt, A. Kimmig, and H. Toivonen, "ProbLog: a probabilistic prolog and its application in link discovery," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, ser. IJCAI'07. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007, pp. 2468–2473.

[2] A. Dries, A. Kimmig, W. Meert, J. Renkens, G. V. den Broeck, J. Vlasselaer, and L. D. Raedt, "ProbLog2: Probabilistic logic programming," in *Machine Learning and Knowledge Discovery in Databases*, vol. 9286. Springer, 2015, pp. 312–315.

[3] E. Tsamoura, V. Gutierrez-Basulto, and A. Kimmig, "Beyond the grounding bottleneck: Datalog techniques for inference in probabilistic logic programs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 6, 2020, pp. 10 284–10 291.

[4] M. Raghothaman, S. Kulkarni, K. Heo, and M. Naik, "User-guided program reasoning using Bayesian inference," *SIGPLAN Not.*, vol. 53, no. 4, pp. 722–735, Jun. 2018.

[5] T. Li and X. Zhang, "Combining formal and informal information in Bayesian program analysis via soft evidences," *Proc. ACM Program. Lang.*, vol. 9, no. OOPSLA1, pp. 1774–1801, Apr. 2025.

[6] J. Wang, S. Halalingaiah, W. Chen, C. Wang, and I. Dillig, "Probabilistic inference for Datalog with correlated inputs," *Proc. ACM Program. Lang.*, vol. 9, no. OOPSLA2, pp. 220–247, Oct. 2025.

[7] J. Li, B. Saha, and A. Deshpande, "A unified approach to ranking in probabilistic databases," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 502–513, Aug. 2009.

[8] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "Probabilistic top-k and ranking-aggregate queries," *ACM Trans. Database Syst.*, vol. 33, no. 3, pp. 13:1–13:54, Sep. 2008.

[9] S. Kabir and Y. Papadopoulos, "Applications of Bayesian networks and Petri nets in safety, reliability, and risk assessments: A review," *Safety Science*, vol. 115, pp. 154–175, 2019.

[10] K. Heo, M. Raghothaman, X. Si, and M. Naik, "Continuously reasoning about programs using differential Bayesian inference," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 561–575.

[11] Y. Shi, Y. Zhang, and X. Zhang, "On abstraction refinement for Bayesian program analysis," *Proc. ACM Program. Lang.*, vol. 9, no. OOPSLA2, pp. 3232–3258, Oct. 2025.

[12] B. Scholz, H. Jordan, P. Subotić, and T. Westmann, "On fast large-scale program analysis in Datalog," in *Proceedings of the 25th International Conference on Compiler Construction*, ser. CC '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 196–206.

[13] J. Zhang, P. Gao, F. Song, and C. Wang, "SCInfer: Refinement-based verification of software countermeasures against side-channel attacks," in *International Conference on Computer Aided Verification*. Springer, 2018, pp. 157–177.

[14] A. Flores-Montoya and E. Schulte, "Datalog disassembly," in *Proceedings of the 29th USENIX Conference on Security Symposium*, ser. SEC'20. USA: USENIX Association, 2020, pp. 1075–1092.

[15] F. Somenzi, "CUDD: CU decision diagram package, release 3.0.0," Software library, 2015. [Online]. Available: https://github.com/cuddorg/cudd

[16] J.-S. Coron, E. Prouff, M. Rivain, and T. Roche, "Higher-order side channel security and mask refreshing," in *International Workshop on Fast Software Encryption*. Springer, 2013, pp. 410–424.

[17] G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, B. Grégoire, and P.-Y. Strub, "Verified proofs of higher-order masking," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 457–485.

[18] M. Rivain and E. Prouff, "Provably secure higher-order masking of AES," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2010, pp. 413–427.

[19] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan, "Soot - a Java bytecode optimization framework," in *Proceedings of the 1999 Conference of the Centre for Advanced Studies on Collaborative Research*, ser. CASCON '99. IBM Press, 1999.

[20] D. Fierens, G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, and L. De Raedt, "Inference and learning in probabilistic logic programs using weighted Boolean formulas," *Theory and Practice of Logic Programming*, vol. 15, no. 3, pp. 358–401, 2015.

[21] J. Huang, Z. Li, B. Chen, K. Samel, M. Naik, L. Song, and X. Si, "Scallop: from probabilistic deductive databases to scalable differentiable reasoning," in *Proceedings of the 35th International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2021, pp. 25 134–25 145.

[22] H. Lin, Z. Yan, and X. Zhang, "Beer: Interactive alarm resolution in bayesian program analysis via exploration-exploitation," *Proceedings of the ACM on Programming Languages*, vol. 10, no. OOPSLA1, pp. 400–426, 2026.

[23] M. Naik, A. Aiken, and J. Whaley, "Effective static race detection for Java," *SIGPLAN Not.*, vol. 41, no. 6, pp. 308–319, Jun. 2006.

[24] J. Wang, C. Sung, M. Raghothaman, and C. Wang, "Data-driven synthesis of provably sound side channel analyses," in *Proceedings of the 43rd International Conference on Software Engineering*, ser. ICSE '21. IEEE Press, 2021, pp. 810–822.

[25] J. Wang, C. Sung, and C. Wang, "Mitigating power side channels during compilation," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, pp. 590–601.

[26] J. Whaley, D. Avots, M. Carbin, and M. S. Lam, "Using Datalog with binary decision diagrams for program analysis," in *Proceedings of the Third Asian Conference on Programming Languages and Systems*, ser. APLAS'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 97–118.

[27] X. Zhang, R. Mangal, R. Grigore, M. Naik, and H. Yang, "On abstraction refinement for program analyses in Datalog," *SIGPLAN Not.*, vol. 49, no. 6, pp. 239–248, Jun. 2014.

[28] V. Bárány, B. T. Cate, B. Kimelfeld, D. Olteanu, and Z. Vagena, "Declarative probabilistic programming with Datalog," *ACM Trans. Database Syst.*, vol. 42, no. 4, pp. 22:1–22:35, Oct. 2017.

[29] Z. Li, J. Huang, and M. Naik, "Scallop: A language for neurosymbolic programming," *Proc. ACM Program. Lang.*, vol. 7, no. PLDI, pp. 1463–1487, Jun. 2023.

[30] F. Riguzzi and T. Swift, "The PITA system: Tabling and answer subsumption for reasoning under uncertainty," *Theory and Practice of Logic Programming*, vol. 11, no. 4–5, pp. 433–449, 2011.

[31] J. Vlasselaer, G. Van den Broeck, A. Kimmig, W. Meert, and L. De Raedt, "Anytime inference in probabilistic logic programs with TP-compilation," in *Proceedings of the 24th International Conference on Artificial Intelligence*, ser. IJCAI'15. AAAI Press, 2015, pp. 1852–1858.

[32] J. Renkens, G. Van den Broeck, and S. Nijssen, "k-optimal: a novel approximate inference algorithm for ProbLog," in *Proceedings of the 21st International Conference on Inductive Logic Programming*, ser. ILP'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 33–38.

[33] J. Renkens, A. Kimmig, G. Van den Broeck, and L. De Raedt, "Explanation-based approximate weighted model counting for probabilistic logics," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, no. 1, 2014, pp. 2490–2496.

[34] I. Balbin, G. S. Port, K. Ramamohanarao, and K. Meenakshi, "Efficient bottom-up computation of queries on stratified databases," *J. Log. Program.*, vol. 11, no. 3–4, pp. 295–344, Aug. 1991.

[35] S. Arch, X. Hu, D. Zhao, P. Subotić, and B. Scholz, "Building a join optimizer for Soufflé," in *Logic-Based Program Synthesis and Transformation: 32nd International Symposium, LOPSTR 2022, Tbilisi, Georgia, September 21–23, 2022, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2022, pp. 83–102.

[36] P. Subotić, H. Jordan, L. Chang, A. Fekete, and B. Scholz, "Automatic index selection for large-scale Datalog computation," *Proc. VLDB Endow.*, vol. 12, no. 2, pp. 141–153, Oct. 2018.

[37] E. Ábrahám, N. Jansen, R. Wimmer, J.-P. Katoen, and B. Becker, "DTMC model checking by SCC reduction," in *2010 Seventh International Conference on the Quantitative Evaluation of Systems*, 2010, pp. 37–46.

[38] X. Li, W. Chen, I. Dillig, and J. Wang, "Incremental inference for probabilistic Datalog," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, E. Darulova, A. W. Lin, and P. Rümmer, Eds., vol. 16684. Springer, Jul. 2026, pp. 169–190.