Good morning everyone,
we are Jacopo Casonatto and Ilaria Zanella, and today we are presenting our project on Knowledge Compilation, focusing on building an end-to-end pipeline from Boolean formulas to sd-DNNF and comparing exact model counting with an approximate method based on SampleSAT.

1. Motivation
The problem we tackle is #SAT: given a Boolean formula, how many assignments satisfy it?
This is a fundamental question in artificial intelligence, probabilistic reasoning, and verification.

But #SAT is #P-complete, which makes it computationally very challenging.
To overcome this, we use Knowledge Compilation: the idea of transforming a formula into a structured form that supports efficient reasoning.

2. The Pipeline
Our pipeline progressively transforms any Boolean formula into sd-DNNF (smooth deterministic decomposable negation normal form).

The stages are:
1. NNF – normalize the formula with negations only on literals.
2. DNNF – enforce decomposability: AND nodes have disjoint variable sets.
3. d-DNNF – enforce determinism: OR branches are mutually exclusive.
4. sd-DNNF – enforce smoothness: OR-children share the same variable set.

This step-by-step refinement guarantees that the final representation can be used for Weighted Model Counting.

3. Deep Dive into Functions

3.1 NNF2DNNF – Decomposability
- Goal: ensure AND nodes do not share variables.
- Method: treat each conjunction as a graph of factors.
* Connected components are variable-disjoint $\rightarrow$ decomposable.
* If only one component, force progress with Shannon expansion.

3.2 DNNF2dDNNF – Determinism
- Goal: ensure OR branches are model-disjoint, avoiding double-counting.
- Method: detect overlaps and apply Shannon expansion on a conflict variable.
- Example: expand on variable s:
$f = (s \wedge f|s=1) \vee (\neg s \wedge f|s=0)$.

3.3 dnnf2sdNNF – Smoothness
- Goal: all OR-children must reference the same variable set.
- Method: structurally smooth by padding missing variables with Tau(v).
- Example: if a branch misses variable b, extend it with Tau(b).

3.4 Orchestrator – compile_to_sdDNNF
- Function: orchestrates the full pipeline end-to-end.
- Steps:
1. to_nnf – normalize formula.
2. NNF2DNNF – enforce decomposability.
3. DNNF2dDNNF – enforce determinism.
4. dnnf2sdNNF – enforce smoothness.

3.5 model_counting_sdDNNF – Weighted Model Counting
- Function: computes the Weighted Model Count using literal weights.
- Process: recursively traverse the structure.
* Constants → 1.0 for True, 0.0 for False.
* Literals → return their weight.
* Tau nodes → return weight(v) + weight(¬v).
* AND nodes → product of children counts.
* OR nodes → sum of children counts.

3.6 SampleSAT and Approximate Counting
Alongside exact counting, we implemented SampleSAT, an approximate approach.

- SampleSAT Algorithm: a stochastic local search with random flips and noise to find satisfying assignments.
- Approximate #SAT Estimation: repeat the process to estimate the count as p_hat * 2^n, where p_hat is the empirical success rate and n the number of variables.
- Exact #SAT via sd-DNNF: provides the ground truth.
- Comparison Framework: both approaches are run on example formulas, producing tables with exact counts, approximations, and relative errors.

This highlights the trade-off: SampleSAT is faster but approximate, while sd-DNNF provides exact and reliable counts.

4. The Demo (Highlight)
The most engaging part of our project is the interactive demo.

- It shows each stage of the pipeline with visual status badges.
- It compares exact vs approximate counts in colorful tables.
- It makes errors and approximations immediately visible.

Through the demo, abstract concepts become tangible: we see formulas being compiled and results compared side by side.

5. Conclusion
To summarize, our project has two main contributions:
1. It demonstrates the theory of Knowledge Compilation, turning complex Boolean formulas into sd-DNNF.
2. It makes this practical and visible through the demo, comparing exact vs approximate reasoning.

The key takeaway is that Knowledge Compilation is not just abstract theory: it provides concrete tools to handle problems that are otherwise computationally intractable.

Thank you for your attention.