

INTRODUCTION

Modern computing relies on hardware accelerators like NVIDIA Tensor Cores for high performance

$$D = \begin{matrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{matrix} \times \begin{matrix} B_{00} & B_{01} & B_{02} & B_{03} \\ B_{10} & B_{11} & B_{12} & B_{13} \\ B_{20} & B_{21} & B_{22} & B_{23} \\ B_{30} & B_{31} & B_{32} & B_{33} \end{matrix} + \begin{matrix} C_{00} & C_{01} & C_{02} & C_{03} \\ C_{10} & C_{11} & C_{12} & C_{13} \\ C_{20} & C_{21} & C_{22} & C_{23} \\ C_{30} & C_{31} & C_{32} & C_{33} \end{matrix}$$

- The internal arithmetic of these accelerators is non-standard and evolves with each generation [1, 3]
 - This can violate the fundamental mathematical property **monotonicity** when summing 4+ terms [4]
- Non-monotonicity can head to numerical errors
 - Violate triangular inequality, interval arithmetic [4]**
- Our work builds upon that of Valpey et al. [5], who previously provided executable formal specifications of these hardware's behaviors

1.000...000		00
+0.000...000		10
+0.000...000		10
+0.000...000		10
+0.000...000		10
1.000...000		00
0.111...111		00
+0.000...000		10
+0.000...000		10
+0.000...000		10
+0.000...000		10
1.000...001		00

Term A: 1.1010×2^3
 Term B: 1.0011×2^1
 Term C: 1.1100×2^0
 Term D: 1.0101×2^{-1}

→ $\begin{matrix} 1.1010 \\ 0.0100 \\ 0.0011 \\ +0.0001 \\ \hline 1.0010 \end{matrix}$ → 1.0001 → New Exponent = Max(E) + 1

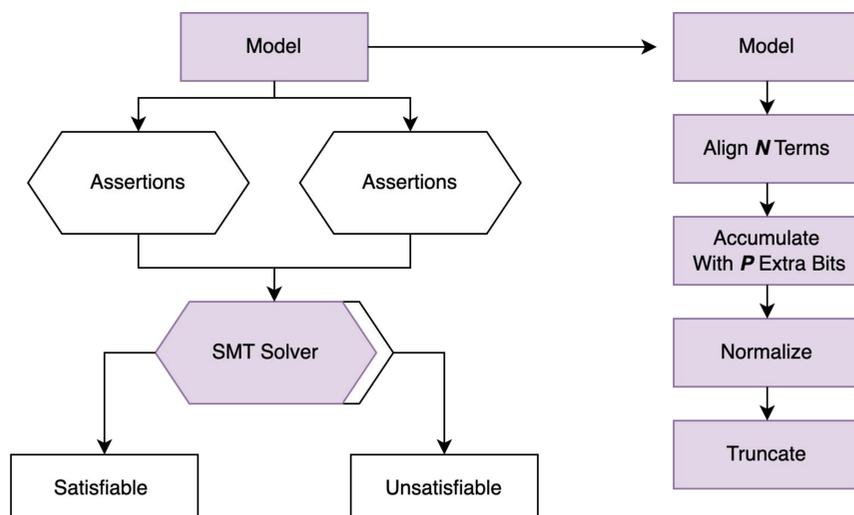
Research Questions

- Can we use formal methods to build an automated and exhaustive framework for verifying numerical properties in hardware accelerators?
- Can this framework identify the specific hardware parameters required to guarantee correct behavior, such as monotonicity?

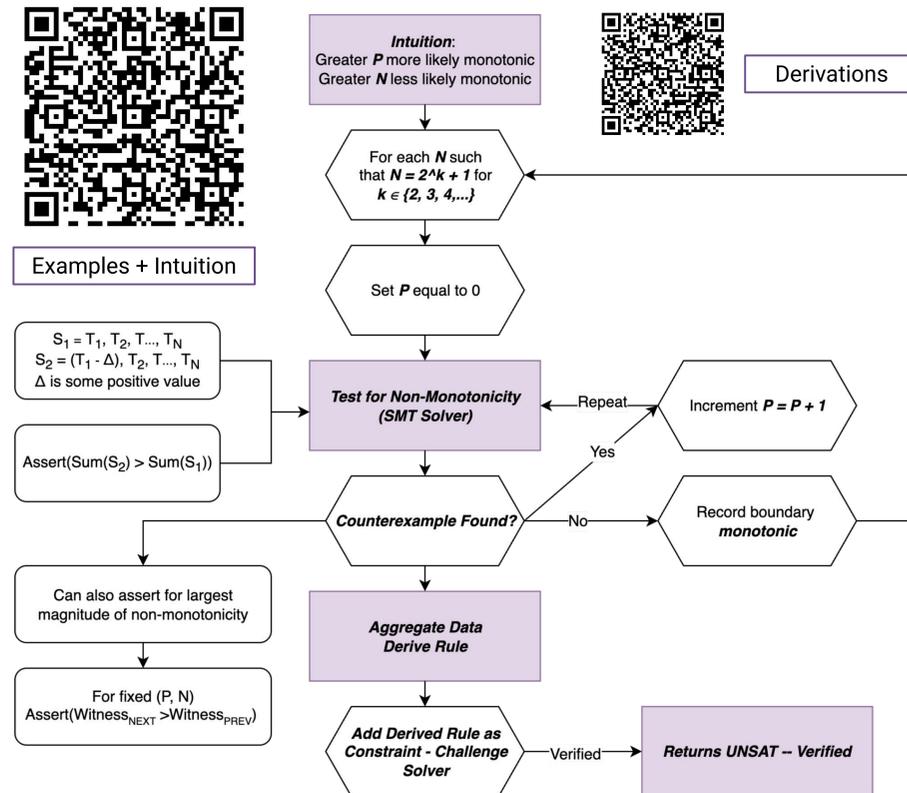
PRELIMINARIES

Satisfiability Modulo Theories (SMT) [2] Solvers are constraint reasoning tools designed **automatically analyze the satisfiability of logical formulas**

- Recently become powerful enough to be practical for floating point
- Standard SMT floating-point libraries are IEEE-754 compliant
- To accurately leverage this tool for non-standard hardware, we create custom encodings built from the ground up with **bit-blasting**



METHODOLOGY



- We primarily investigate monotonicity in addition
- Example P = 0, N = 5:**
 - $2^0 + 2^{-24} + 2^{-24} + 2^{-24} + 2^{-24} = 2^0$
 - $(2^0 - 2^{-24}) + 2^{-24} + 2^{-24} + 2^{-24} + 2^{-24} = 2^0 + 2^{-23}$
- To study this, we translate the property of monotonicity into a logical query for SMT
 - Two inputs sets, $S1 = \{a, \dots, x\}$ and $S2 = \{a, \dots, y\}$ where $y > x$ → Find values such that the hardware sum of S1 > the hardware sum of S2
- Our approach involves systematically varying the model's parameters
 - We adjust the number of terms (**N**), and the number of internal padding bits (**P**)
 - This tells us whether non-monotonicity can arise under these specifications

From this data, we manually derive conditions that define the behavior of this phenomena

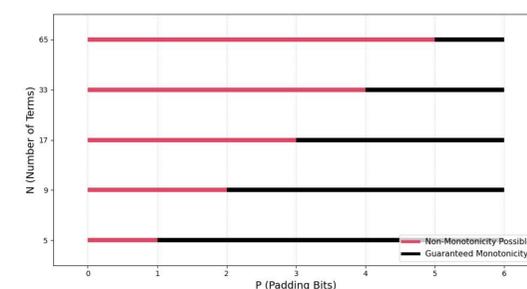
- These conditions are fed back to the SMT solver and challenged
 - If the Solver returns UNSAT, the derived rules are exhaustively and provably correct

RESULTS

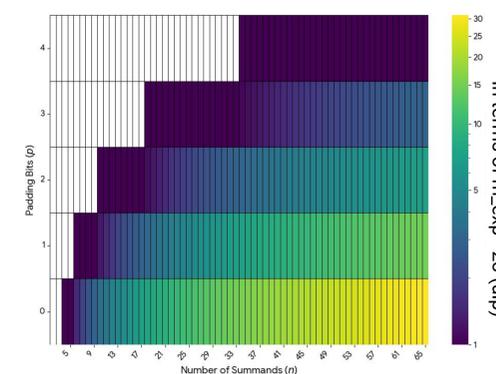
- We reveal that a block-adder is provably monotonic when: $p \leq L(\log_2(n-1)-2)J$
- We also derive a formula to calculate the maximum magnitude of non-monotonicity, M, for any given (n, p) configuration and maximum exponent (m_exp) for FP32 accumulation:
 $M(n, p, m_exp) = (n-1) \cdot 2^{(m_exp - (24+p))} - 2^{(m_exp - 23)}$

Architecture	Summands (n)	Padding (p)	Condition Check: $p \leq \lfloor \log_2(n-1) - 2 \rfloor$	Result
Volta	5	0	$0 \leq 0$	Susceptible
Turing	5	1	$1 \leq 0$	Monotonic
Ampere	9	1	$1 \leq 1$	Susceptible
Hopper	17	2	$2 \leq 2$	Susceptible

- With this, hardware architects can eliminate non-monotonicity at the design stage
- Software engineers and numerical analysts can bound potential error when leveraging existing hardware
 - **enables the development of more robust algorithms**



(1) Required Padding Bits ('p') for Guaranteed Monotonicity



(2) Relative Magnitude of Non-Monotonicity

CONCLUSION

- Our SMT-based approach transforms the process of hardware verification from a laborious, manual task into a systematic, semi-automated, and formal one
- We demonstrate that our framework can be used to identify the exact hardware configurations to guarantee fundamental mathematical properties like monotonicity

REFERENCES

[1] Massimiliano Fasi, Nicholas J Higham, Mantas Mikaitis, and Srikanth Praneeth. Numerical Behavior of NVIDIA Tensor Cores. PeerJ Computer Science, 7:e330, 2021.

[2] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: an efficient SMT solver. In Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems (TACAS'08/ETAPS'08). Springer-Verlag, Berlin, Heidelberg, 337–340.

[3] Xinyi Li, Ang Li, Bo Fang, Katarzyna Swirydowicz, Ignacio Laguna, and Ganesh Gopalakrishnan. FTTN: Feature-Targeted Testing for Numerical Properties of nvidia & AMD Matrix Accelerators. In 2024 IEEE/ACM 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid). IEEE, 2024.

[4] Mantas Mikaitis. Monotonicity of multi-term floating-point adders. IEEE Transactions on Computers, 73(6):1531–1543, 2024.

[5] Benjamin Valpey, Xinyi Li, Sreerathi Pai, and Ganesh Gopalakrishnan. An SMT Formalization of Mixed-Precision Matrix Multiplication: Modeling Three Generations of Tensor Cores. In NASA Formal Methods Symposium, pages 360–379, Cham, Jun 2025. Springer Nature Switzerland.