

A Formal Characterization of Non-Monotonicity in Tensor Cores

Paul Jiang¹, Vivian Zheng², Ganesh Gopalakrishnan³ (Advisor)



Research under REU Site NSF 2244492
Trust and Reproducibility Education
for Undergraduates

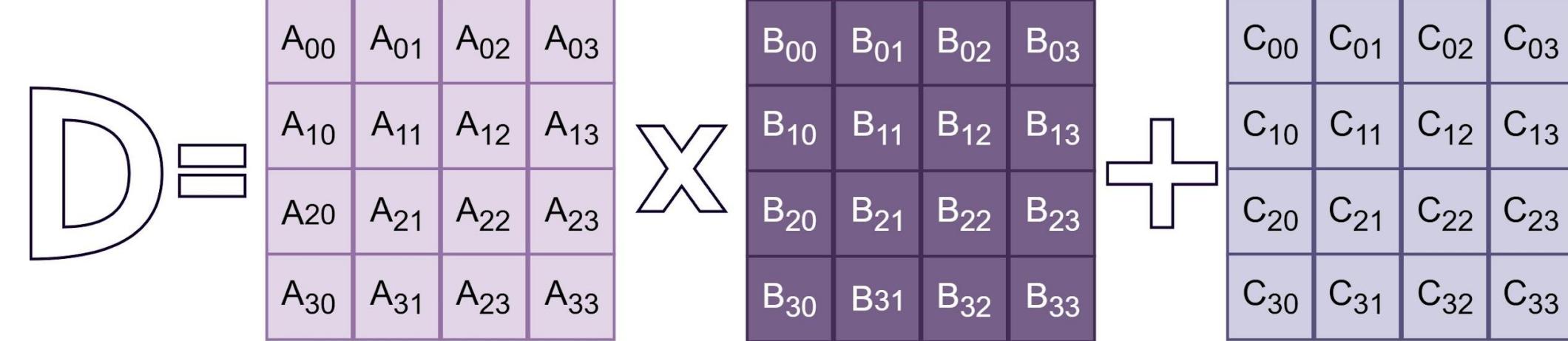
¹Department of Computer Science, Purdue University

²Department of Computer Science, Stony Brook University

³Kahlert School of Computing, University of Utah

INTRODUCTION

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



$$\begin{array}{l} 1.000\ldots000 \mid 00 \\ +0.000\ldots000 \mid 10 \\ +0.000\ldots000 \mid 10 \\ +0.000\ldots000 \mid 10 \\ +0.000\ldots000 \mid 10 \\ \hline 1.000\ldots000 \mid 00 \end{array}$$

$$\begin{array}{l} 0.111\ldots111 \mid 00 \\ +0.000\ldots000 \mid 10 \\ +0.000\ldots000 \mid 10 \\ +0.000\ldots000 \mid 10 \\ +0.000\ldots000 \mid 10 \\ \hline 1.000\ldots001 \mid 00 \end{array}$$

- 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 lead 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

$$\begin{array}{ll} \text{Term A: } 1.1010 \times 2^3 & 1.1010 \\ \text{Term B: } 1.0011 \times 2^1 & 0.010011 \\ \text{Term C: } 1.1100 \times 2^0 & 0.0011100 \\ \text{Term D: } 1.0101 \times 2^{-1} & 0.0001010 \\ \hline & 10.0011 \\ & \downarrow \\ & 1.0001 \rightarrow \text{New Exponent} = \text{Max}(E) + 1 \\ & 10.0010... \end{array}$$

Research Questions

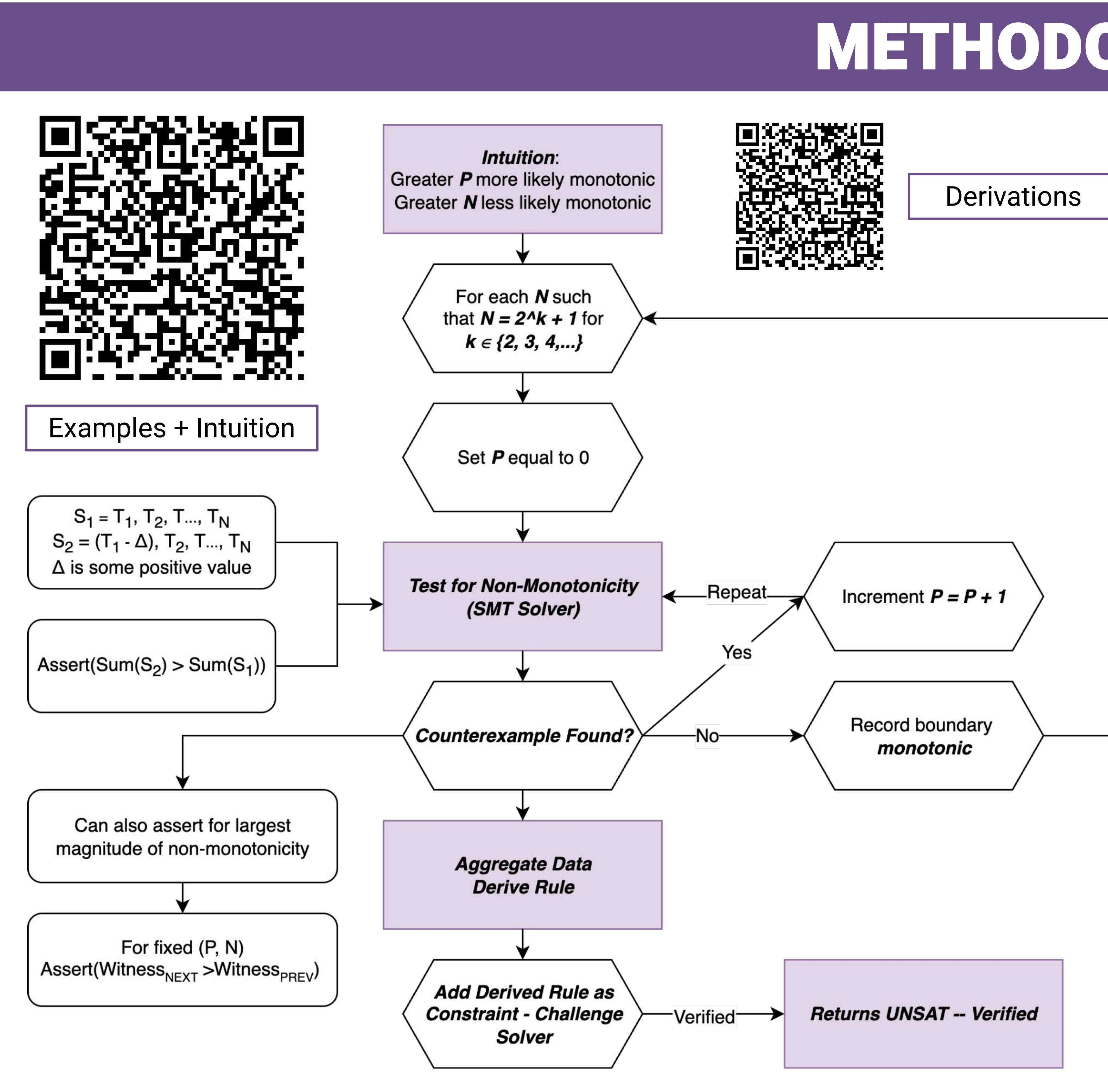
- Can we use formal methods to build an automated and exhaustive framework for verifying numerical properties in hardware accelerators?
- Can this framework identify the specific hardware parameters required to guarantee correct behavior, such as monotonicity?

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 input 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

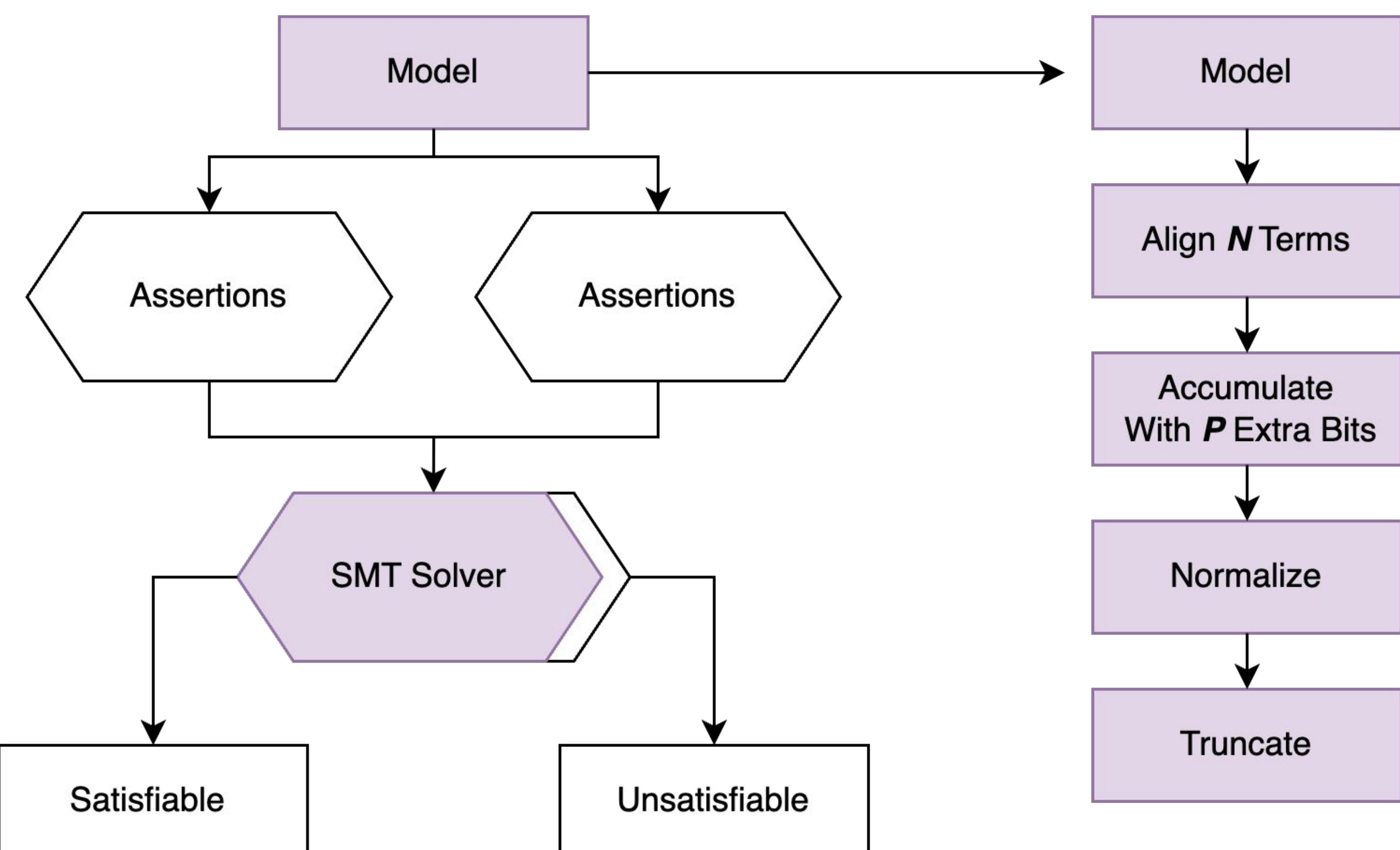
- These conditions are fed back to the SMT solver and challenged
 - If the Solver returns UNSAT, the derived rules are exhaustively and provably correct



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**



RESULTS

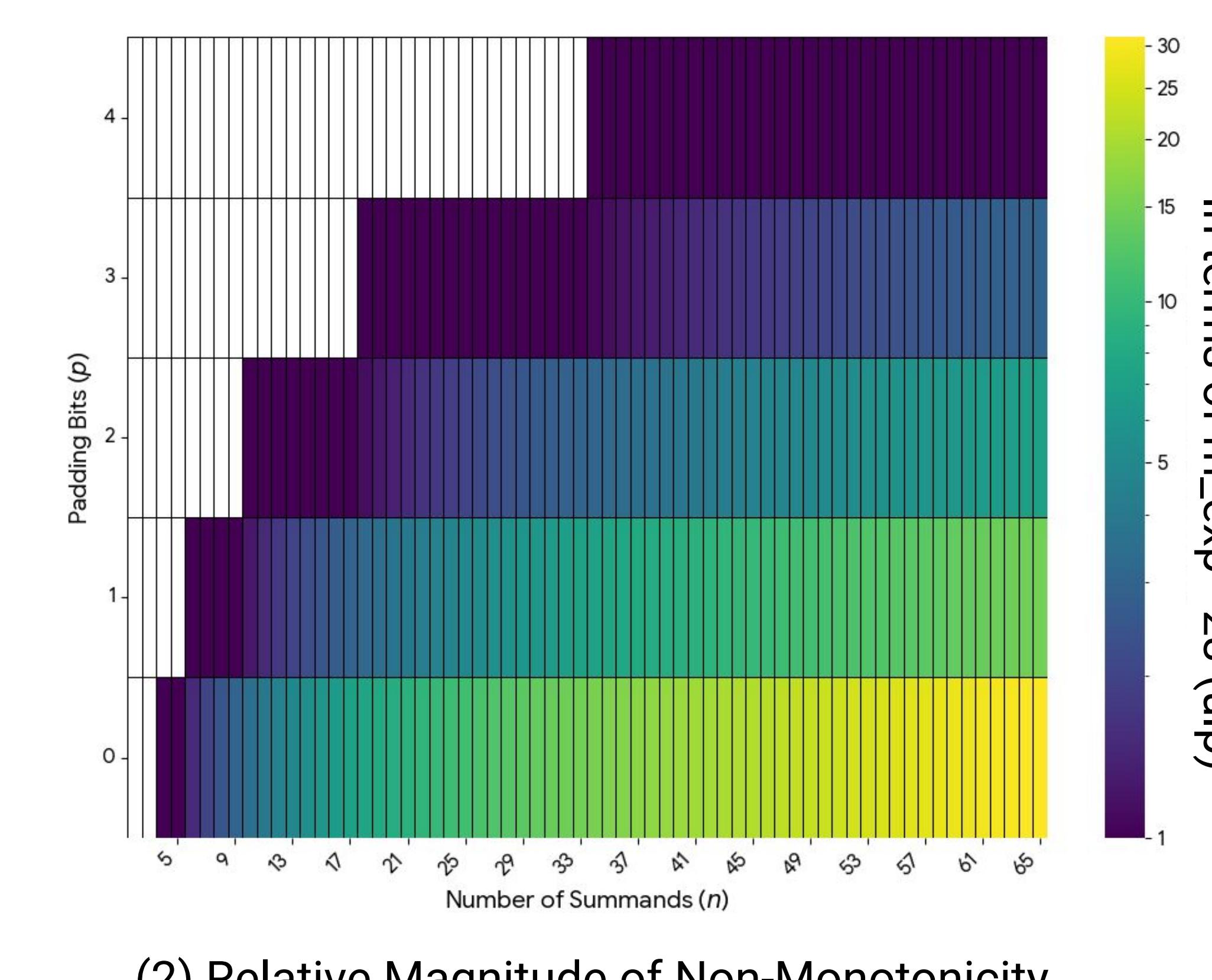
- We reveal that a block-adder is provably monotonic when: $p \leq L(\log_2(n-1)-2)$
- We also derive a formula to calculate the maximum magnitude of non-monotonicity, M, for any given (n, p) configuration and maximum exponent (m_exp) for FP32 accumulation:

$$M(n,p,m_{\text{exp}}) = (n-1) \cdot 2^{(m_{\text{exp}} - (24+p))} - 2^{(m_{\text{exp}} - 23)}$$



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



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 Pranesh. 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. FTN: 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, Sreepathi 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.